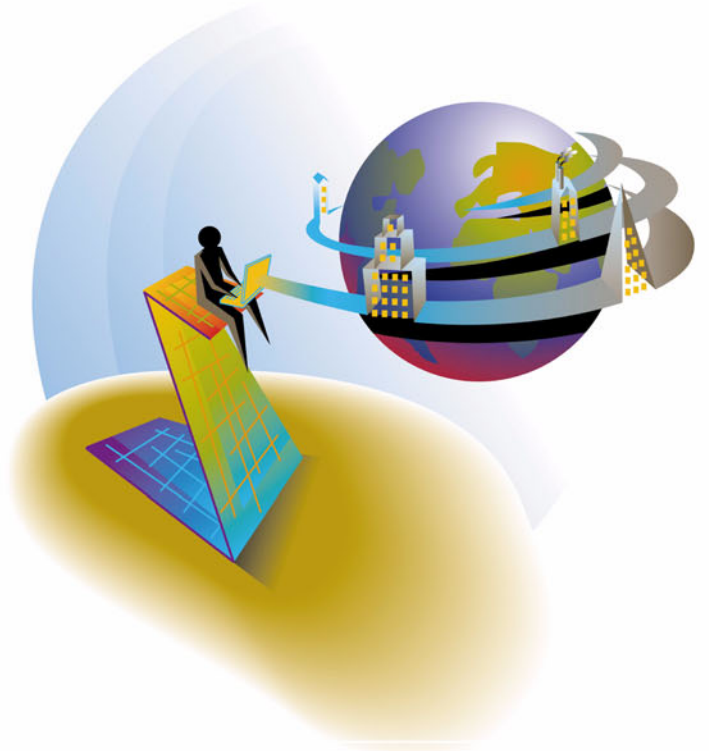


eDeveloper V9.4



Reference Guide

The information in this manual/document is subject to change without prior notice and does not represent a commitment on the part of Magic Software Enterprises Ltd.

Magic Software Enterprises Ltd. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.

The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms and conditions of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.

No part of this manual and/or databases may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or information recording and retrieval systems, for any purpose other than the purchaser's personal use, without the prior express written permission of Magic Software Enterprises Ltd.

All references made to third-party trademarks are for informational purposes only regarding compatibility with the products of Magic Software Enterprises Ltd.

Unless otherwise noted, all names of companies, products, street addresses, and persons contained herein are part of a completely fictitious scenario or scenarios and are designed solely to document the use of Magic.

Magic® is a registered trademark of Magic Software Enterprises Ltd.

Btrieve® and Pervasive.SQL® are registered trademarks of Pervasive Software, Inc.

IBM®, Topview™, iSeries™, pSeries®, xSeries®, RISC System/6000®, DB2®, and WebSphere® are trademarks or registered trademarks of IBM Corporation.

Microsoft®, FrontPage®, Windows™, WindowsNT™, and ActiveX™ are trademarks or registered trademarks of Microsoft Corporation.

Oracle® and OC4J® are registered trademarks of the Oracle Corporation and/or its affiliates.

Linux® is a registered trademark of Linus Torvalds.

UNIX® is a registered trademark of UNIX System Laboratories.

GLOBETrotter® and FLEXIm® are registered trademarks of Macrovision Corporation.

Solaris™ and Sun ONE™ are trademarks of Sun Microsystems, Inc.

HP-UX® is a registered trademark of the Hewlett-Packard Company.

Red Hat® is a registered trademark of Red Hat, Inc.

WebLogic® is a registered trademark of BEA Systems.

Interstage® is a registered trademark of the Fujitsu Software Corporation.

JBoss™ is a trademark of JBoss Inc.

Clip art images copyright by Presentation Task Force®, a registered trademark of New Vision Technologies Inc.

This product uses the FreeImage open source image library. See <http://freeimage.sourceforge.net> for details.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

This product includes software developed by Computing Services at Carnegie Mellon University (<http://www.cmu.edu/computing/>).

Copyright © 1989, 1991, 1992, 2001 Carnegie Mellon University. All rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

This product includes software that is Copyright © 1998, 1999, 2000 of the Thai Open Source Software Center Ltd. and Clark Cooper.

This product includes software that is Copyright © 2001-2002 of Networks Associates Technology, Inc All rights reserved.

This product includes software that is Copyright © 2001-2002 of Cambridge Broadband Ltd. All rights reserved.

This product includes software that is Copyright © 1999-2001 of The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved.

All other product names are trademarks or registered trademarks of their respective holders.

eDeveloper Reference Guide V9.4 SP6

September 2005

Contents

1 Introduction

- Documentation..... 35
- Online Resources..... 35
- Installation 35
- Typographical Conventions 36
- Key Combinations..... 37
- eDeveloper Workspace 37
 - Navigator and Workspace Panes..... 38
 - Multiple Document Interface (MDI) Client Edge..... 38
 - Property Sheets 39
 - Comments..... 39
 - Checker Results 39
 - Switch Panes 40
 - Combined Panes..... 40
 - Repositories..... 42

2 Settings

- The Settings Menu..... 53
- New Applications 55
- Application Settings 56
 - Application Repository..... 56
 - Application Properties - Outside the Application 60
 - Application Properties - Within the Application..... 63
- Environment Settings..... 69
 - System 71
 - Multi-User 84

Preferences	88
International.....	103
External	105
Server	116
eDeveloper Defaults	124
Advanced Toolkit Settings.....	125
JAVA Settings	125
The MAGIC_JAVA Section	125
Color Settings	126
The Color Repository Settings	126
Saving Changes to the Color Repository	129
Font Settings	131
The Font Repository Settings	132
Font Assignment Window	133
Saving Changes to the Font Repository	133
Keyboard Mapping Settings.....	134
eDeveloper Actions.....	134
State Qualifications to eDeveloper Actions	135
eDeveloper Action Example.....	136
The Keyboard Mapping Repository	136
The Keyboard Mapping Repository Settings.....	137
Servers Settings.....	140
The Server Repository Settings	141
Services Settings.....	144
The Service Repository Settings	144
The Services Properties Dialog	145
Visual Connection Settings.....	146
Communication Settings.....	148
The Communication Repository Settings.....	148

DBMS Settings	149
The DBMS Repository Settings	151
DBMS Properties	153
Variable MCF Record Length.....	158
Database Settings	159
The Database Repository Settings.....	161
The Database Properties Dialog.....	164
Logical Names Settings	172
The Logical Name Repository Settings	173
Logical Names Usage	173
Language Settings.....	174
The Language Repository Properties	175
Printer Settings	177
The Printer Repository Properties.....	178
HTML Style Settings.....	179
The HTML Style Properties	179
Print Attribute Settings.....	180
Secret Name Settings.....	180
Connecting to an LDAP Server.....	180
User Groups Settings	181
User ID Settings.....	181
Logon Settings	181
The Logon Properties	182
System Logon Setting.....	183
The Magic.ini File.....	183
The Magic.ini File Format	185
Saving Server Information to the Magic.ini File.....	187
Command Line Options	187
Specifying Command Line Options	187

Command Line Options Examples	189
Application Launch via the OS Command Line	190
Command Line Options and Magic.ini Values	190
Environment Properties and Command Line Values	192

3 Models

Model Repository	200
Columns	200
Classes	200
Properties	201
Data Items	203
Data Item Qualifiers	204
Attributes	204
Storage Field Models	208
Pictures	208
Functional Directives	210
Positional Directives	218
Mask Characters	224
Syntax Rules for Constructing Pictures	225
Field Class Properties	226
Model	226
Details	226
ActiveX and OLE	227
Input	228
Appearance	228
Style	229
Def/Null	229
Storage	230
SQL	230

Control Properties.....	231
Details	231
Input	233
Appearance	234
Navigation.....	236
OLE	236
Form Properties.....	237
Model	237
Details	237
Input	239
Appearance	239
Navigation.....	240
Help Properties.....	241
Model: (default)	241
Details	241
Input	242
Appearance	242
Navigation.....	242
Working with Models	242
Creating a Model	242
Deleting a Model	243
Breaking Model Properties	243
Selecting a Different Model for an Object	243
Removing a Model from an Object	243
Expressions	244
Rights.....	244

4 Tables

Table Repository.....	246
Table Repository Columns	248
Table Properties	251
Table Properties - Advanced.....	251
Table Properties - SQL	253
Resident Tables.....	256
Table Conversion Utility.....	257
Column Repository	258
Column Repository Fields	258
Column Properties	260
Index Repository	266
Index Repository Columns	268
Index Segment List Columns.....	268
Index Properties.....	270
Index Properties - Advanced	270
Index Properties - SQL	271
Foreign Key Repository	273

5 Application Engine

Tasks.....	277
Engine Levels.....	278
Operation Repositories	280
Event Handling.....	283
Event Types.....	283
Interactive Task Event Handling	284
Batch Task Event Handling	285
Handlers	287
User-Defined Events	292

#	292
Description	293
Trigger Type	293
Trigger.....	293
Force Exit.....	293
Public Name	294
Expose.....	294
Information about the Engine	295
The 14 eDeveloper Operations	295
The Task Dataview	299
The Effect of Modes of Operation on Task Flow	301
End-User Screen Interaction	304
Engine Execution Rules	305
Task Cycle Levels	305
Task Cycle.....	307
The Record/Row Loop	308
The Control Level	310
Group Levels	312
How the Engine Executes Group Levels	313
Group Levels Example.....	313
Record/Row Loop Flowcharts.....	314
Record/Row Loop in Online Tasks	314
Record/Row Loop in Batch Tasks	316
Engine by Record Level	318
Controlling the Execution of an Operation	322
End-User Screen Interaction	328

6 Programs

Program Repository	331
Properties of the Program Repository	331
Tasks	332
Menu Options for Tasks	332
Task Properties Dialog	333
Direct SQL Command	346
Using Direct SQL Command	347
Direct SQL Task Elements	348
SQL Command Automatic Program Generator	352
Behavior of Direct SQL SELECT Statements	352
Restrictions on Using Direct SQL	354
Binding Variables	355
Allow DSQL in a Deferred Transaction	356
Task Control	357
Task Control Properties as Conditions	357
Task Control Properties Dialog	357
Local Variable Repository	364
Properties of the Variable Repository	364
Local Variable Properties Sheet	365
Expression Rules Repository	366
Expressions	367
Form Repository	369
Form Repository Columns	371
Working with Forms	374
DB Table Repository	380
Properties of the DB Table Repository	380
I/O File Repository	383

Properties of the I/O File Repository.....	384
I/O Properties Dialog	389
Sort Repository	391
Properties of the Segment Area.....	392
Event Repository	394
Properties of the Event Repository	395
Range and Locate Properties	398
Range/Locate Tab	398
SQL Where Tab	401
Task Execution Repository	406
The Structure of the Handler Repository	407
Handler Repository Properties	408
Main Program	411
Main Program Access and Usage.....	411
Main Program Characteristics	412
Toolkit Characteristics and Behavior.....	414

7 Operations

Alphabetical Index to Operations	420
Introduction.....	420
Remark	422
Purpose	422
Usage	422
Remark Operation Property	422
Select	422
Purpose	422
Usage	423
Placement	423
Select Operation Properties.....	424

Verify	433
Purpose	433
Usage	434
Verify Operation Properties	434
Link	437
General Information about eDeveloper's Link	437
Link Usage	442
Placement	442
Link Operation Properties	442
Link Types.....	443
Link Properties	450
End Link	454
Purpose	454
Usage	455
Placement	455
End Link Operation Properties	455
Block	456
Purpose	456
Usage	456
Block Operation Types	456
End Block	458
Purpose	458
Usage	459
End Block Operation Properties	459
Call Operations.....	459
Purpose	459
Passing Arguments	460
The Argument List	460
How eDeveloper Passes Arguments.....	462

Call Operation Qualifiers	463
Call Task	465
Call Program and Call Exp.....	469
Call a Public Program	471
Call UDP.....	475
Call COM	477
Call Remote.....	477
Call Web Service	478
Evaluate.....	483
Purpose	483
Usage	483
Evaluation Operation Properties.....	485
Update.....	486
Usage	486
Update Operation Properties	486
Output Form	492
Purpose	492
Usage	493
Output Form Operation Properties	493
Input Form	497
Purpose	497
Usage	497
Input Form Operation Properties	497
Browse.....	500
Purpose	500
Usage	500
Browse Operation Properties	501
Exit	503
Purpose	503

Usage	503
Exit Operation Properties	503
Raise Event	505
Raise Event Properties	505
Raise Public Event Runtime Behavior	509

8 Expression Rules

Literals.....	511
Operators	515
Mathematical Operators	515
Logical Operators	515
String Operator	516
Variables	516
Functions	517
Dynamic Data Exchange.....	517
Buffer Management	518
Vector Data	518
XML Namespaces	519
Function Summary	523
Alphabetical Directory of Functions	553

9 Display Forms

Browser Forms	765
Browser Subforms	766
Browser Form Properties	768
HTML Control Repository	770
Browser Control Properties.....	772
Browser Edit Control Properties	772
Browser Radio Button Control Properties.....	775
Browser Hyper-Text Control Properties	776

Browser Push Button Control Properties.....	778
Browser Check Box Control Properties.....	781
Browser List Box Control Properties	783
Browser Combo Box Control Properties.....	785
Browser Image Control Properties.....	787
Browser Table Control Properties	788
Browser IFRAME Control Properties.....	790
Browser Opaque Control Properties.....	790
GUI Display Forms.....	791
GUI Display Form Properties	791
GUI Display Commands.....	798
GUI Display Color Palette	804
GUI Display Controls.....	804
Static Controls	807
Choice Controls.....	807
Slider Controls	809
Editing, Action, and Image Controls	809
Table Controls.....	813
Tree Control	815
Drag and Drop	820
Drag Begin Event	820
Drop Event	821
Drag and Drop Limitations and Environment Settings	822
GUI Display Control Properties	824
Radio Button Control Properties.....	825
Rectangle Control Properties	831
Table Control Properties	835
Edit Control Properties	842
Column Control Properties	848

Tab Control Properties	849
Ellipse Control Properties	855
Image Control Properties	859
Text Control Properties	864
List Box Control Properties	868
Line Control Properties	875
OLE Control Properties	878
Push Button Control Properties	882
Combo Box Control Properties	888
Slider Control Properties	893
Rich Text Control Properties	896
Check Box Control Properties	901
Group Control Properties	905
Rich Edit Detail Control Properties	909
Tree Control Properties	913

10 Output Forms

HTML Forms	921
HTML Form Display	921
HTML Control Placement	921
HTML Form Properties	922
HTML Style Repository	924
Hyperlink Settings	925
Context Variables	927
The HTML Command Palette	929
HTML Static Table Command Palette	932
HTML Control Palette	935
Fonts and the HTML Controls	937
HTML Control Properties	937

Static Table Control Properties	943
Frame Set Forms	945
Frame Set Form Properties	946
The Frame Set Command Palette.....	947
Frame Set Control Properties	949
HTML Merge Forms	950
HTML Merge Form Properties	951
Web Online Event Handlers.....	952
The Merge Command Table	955
The Merge Command List	957
HTML Template File Tags	958
HTML Merge Tags	959
HTML Merge Syntax Rules	961
HTML Merge Runtime Behavior	961
HTML File Merge Example	962
Web Online Page	964
Web Online Response	964
Upload Capability of the Requester	965
Report Forms	966
GUI Table Control Functionality	966
Multi-Line Edit Printing	967
Printer Attribute Support	968
Printer Settings	969
Print Styles.....	970

11 Data Management

Transaction Processing	977
Transactions and Execution Levels	977
Physical Transactions at Task Level.....	979

Physical Transactions for the Group Level	980
Physical Transactions for the Record Level	980
Deferred Transactions	982
Transaction Begin.....	982
Locking Strategy Property	983
SQL Range Statement.....	983
Direct SQL.....	983
Numeric Field Updates	983
Update/Delete Statements.....	984
Record Update Fail Before Call	985
Nested Transactions	985
Transaction Tree	987
Open Transaction	987
Close Transaction	988
Runtime Tree Sample	989
Transaction Processing Recovery	990
eDeveloper's Internal Transactions.....	990
Mapping Transactions to Databases	990
Deadlocks and Transaction Processing	991
Rollback	992
Rollback Behavior for Browser-Based Programs	992
eDeveloper Cache.....	993
What Can Be Cached	993
When is the Cache Used	994
Activating the Cache Size	994
Changes to Program Behavior	995
Cache and Resident Tasks	995
Cache and The Rollback Operation	996
Cache and Client/Server.....	996

eDeveloper Cache Internal Implementation.....	996
Error Handling.....	997
Error Handling Mechanism	997
Error Behavior Strategies	997
Error Handlers	1000
Error Information	1008
Runtime Error Handling.....	1009
Task Range According to a Record's Position	1009
Applications from Previous Versions	1011

12 End-User Menus & Help

Menu Formats.....	1014
Pulldown Menus	1015
Context Menus.....	1015
Menu Repository.....	1016
Menu Name	1017
Menu Type	1017
Menu Definition Repository	1017
Entry Types	1018
Entry Text	1018
Entry Name	1018
Menu Parameters	1019
Menu Access Key.....	1020
Menu Authorization Options	1021
The Menu Properties Dialog.....	1021
Properties Tab	1022
Toolbox Tab.....	1024
States Tab.....	1025
Help Screen Repository	1026

Help Types	1027
Internal Helps	1028
Prompts	1029
Windows WinHelp Connections	1030
Tooltips.....	1032
URLs.....	1033

13 Authorization System

Magic Security and People's Roles	1035
Getting Started as Supervisor	1036
Rights Repository	1036
Description of the Rights Repository.....	1036
Rights Assignment Dialogs	1039
Model Repository Rights Assignment	1040
Table Repository Rights Assignment	1041
Program Repository Rights Assignment	1042
Help Screens Repository Rights Assignment	1044
Menu Definition Rights Assignment	1045
Component Repository Rights Assignment	1046
Application Properties Dialog Rights Assignment	1047
Application Access Key	1048
Public Rights Access Key	1048
The Super Right Key	1049
Force MVCS Key	1049
Restricting Import and Export	1050
User ID Repository	1051
Description of the User ID Repository.....	1051
User Group Repository	1053
Description of the User Group Repository.....	1053

The Secret Name Repository	1054
Data Security	1054
Restricting Access to Application Data Tables	1054
Restricting Access to the Application File Itself.....	1055
HTTP Authentication.....	1056

14 Components

Component Frameworks	1058
Component Repository	1059
Loading a New Component	1060
Deleting a Component.....	1060
Magic Component Properties	1061
Objects Connected to the Magic Component Interface	1063
Component Runtime Behavior	1063
INIPut Function Behavior	1064
Component Interface Builder Repository	1066
Magic Component Builder Properties	1067
Item Type Repository.....	1067
Environment Repository	1068
Adding an Item	1068
Generating the Magic Component Interface File	1069
Sample MCI File	1069
Web Service Interface Builder	1070
Web Service Programs Repository.....	1071
WSDL File Settings	1073
Generating a WSDL File.....	1074
The Created WSDL File.....	1074
Enterprise JavaBean Interface Builder	1074
EJB Programs Repository	1075

EJB Settings	1076
EJB Environment Variable Path Settings	1076
Generating an EJB Component File	1077
Additional Generated Jar Files	1077
Java Generator.....	1078
XML Generator	1078

15 COM Object Support

OLE and ActiveX	1080
Defining COM Object Fields	1080
Attribute	1080
Object Name and Type Library Settings	1080
Calling a COM Object	1081
Handling ActiveX Events	1084
Runtime Behavior	1086
Passing Objects as Arguments.....	1086
Manual Object Instantiation	1087
Referring to an Already Created Object.....	1087
Retrieving COM Related Error	1088
Placing an ActiveX Control on a Form	1088
OLE Variable and BLOB Variable of OLE Content.....	1088
COM Interface Builder	1088
COM Interface Builder Repository	1089
Object Settings	1092
Generating a COM Object	1094
Registering the Object.....	1095
Local COM Object Runtime Behavior	1095
Remote COM Object Runtime Behavior	1097
COM Object Errors and Troubleshooting.....	1099

16 Java Integration

Java Terminology	1104
Java and EJB Functions	1105
Code Pages.....	1107
Type Signatures	1109
Runtime Engine Behavior	1110
Life Cycle	1110
Multi-Threading.....	1110
Browser-Based Programs.....	1111
Conversion Tables	1111
Returning Pseudo-Reference Values	1112
Errors and Exception Handling.....	1112
Garbage Collection Mechanism	1113
Environment	1113
Java Component Generator.....	1114
Specifying the Java Class or Enterprise JavaBeans Type	1114
The Java Object Browser	1116
Java Class Structure	1117
The Generated Java Component	1118
Created Files	1119

17 XML Component Generator

XCG Wizard	1121
XCG Main Options.....	1122
Modifying a Component.....	1122
XML Schema Details	1124
View XSD	1125
XML Schema Interface Details.....	1126

Data Types	1127
Component Details	1129
XCG Programs.....	1130
Count Program.....	1130
DbDel Program	1130
Get Program	1131
Put Program	1131
Read Program	1131
Search Program	1131
Write Program	1132
Generating the Component	1132
Output Files.....	1133
Changing the XCG Directory	1134
Namespace Support.....	1134

18 Connecting Magic to External Applications

Dynamic Data Exchange	1136
Functions	1137
Magic & OLE Automation	1143
Implementing OLE Automation.....	1144
Parameter Type String	1144
OLE Automation Functions	1145
Call to a DLL	1152
Call to a 3rd Generation Language	1152
Call UDP Operation Parameters	1153
UDP Functions	1154

19 Distributed Application Architecture

The Enterprise Server General Scheme.....	1156
Uses of Distributed Application Architecture.....	1157

Application Partitioning	1157
Internet/Intranet Applications	1158
Enterprise Server Setup	1158
Runtime Engine Behavior	1158
Loading a Middleware Gateway	1159
Supported Middleware	1160
Magic Request Broker - MRB	1160
Magic Request Broker Behavior	1168
eDeveloper Requesters	1173
Requester Settings	1173
Internet Requester	1180
How You Can Use eDeveloper on the Internet	1181
Application Development Concepts	1181
Setting Up an Internet Requester	1182
Other Web Servers	1183
Internet Application Paradigms	1184
SOAP Server Requests	1184
Browser Client Applications	1184
Creating a Browser Task	1185
Creating a Browser Client Program with the Automatic Program Genera- tor (APG)	1185
Writing the Logic for the Browser Task	1185
BLOB Support	1188
Explicit Handling for a Browser Task	1189
Help Action Support	1190
Creating the Browser Task Interface	1191
Recompute	1199
Creating a Batch-Based HTML Program	1201
The Benefits of Application Partitioning	1203

The Call Remote Command.....	1204
Synchronous Execution vs. Asynchronous Execution.....	1206
Dynamic Assignment of Partitions	1207
Command Line Requester	1214
Multi-Threading.....	1216
eDeveloper Monitor Application.....	1219
Request-Related Functions.....	1220

20 Utilities

Application Wizard	1223
Automatic Program Generator	1223
Program Generator Properties for Database Tables	1224
Program Generator Properties for a Program Entry.....	1229
Check Syntax Utility.....	1231
Checker Message Categories.....	1231
Checker Results	1232
The Check Syntax Process	1233
Checker Messages Table.....	1234
Get Definition Utility.....	1235
Loading Tables.....	1235
Cross Reference Utility	1237
The Location From Where to Cross Reference an Object....	1238
Deleting a Cross Reference	1243
Searching for a Cross Reference	1243
Saving Cross References	1244
Printing Cross References	1244
Changing the Maximum Number of Cross-Referenced Results	1244
Export-Import Utility.....	1245
The Export/Import Dialog Box	1246

Flow Monitor/Debugger	1250
Flow Monitor Toolbar	1251
Flow Monitor Message Group Filters	1253
Flow Monitor Properties	1254
Flow Monitor Utility for a Server	1255
Flow Monitor Support for the Browser Client.....	1257
The Remote Flow Monitor	1257
The Profiler	1266
Profiler Operation	1266
Profiler Output	1267
Program Execution Trace File	1267
Opened Files Trace File	1269
The OEM2ANSI Utility.....	1270
The ODBC Check Driver Utility	1271
The MakeKey Utility	1271
The Table Conversion Utility.....	1271
Magic Flat File	1273
Print Data Wizard	1274
Runtime Operations.....	1274
Delimiters and String Identifiers	1276
Runtime Behavior.....	1276
XML Template Structure	1277
XSD Data Type	1278
HTML Template Structure	1282
Tools Infrastructure	1284
Building the Menu.....	1284
Tools Menu Example	1286
Operation Commands	1287
Global Parameters Information.....	1296

Automatic Processing	1299
Monitor Utility	1300
Enterprise Servers	1300
Contexts	1301
Requests	1301
Statistics	1302
Applications	1302
Window Displays	1302
Monitoring Servers	1303
The Documentation Template Facility	1303
Producing Template Documentation	1303
Syntax - Documentation Template File	1305
Documentation Report Sections	1309
Keywords	1313

21 Simple Network Management Protocol

SNMP Implementation	1372
eDeveloper Requester Settings	1372
Magic Request Broker Settings	1373
Environment Settings	1374
SNMPNotify Function	1374
Other Traps	1374
Network Management Station Query from eDeveloper	1375
Enterprise Servers (QUE=RT)	1375
Requested Query (QUE=QUE)	1376
Loaded Query (QUE=LOAD)	1376
NMS Management Options	1377
Installation and Configuration	1379
Supported SNMP Agents	1380

22 J2EE Integration

Terminology.....	1382
The Component Builder	1383
Defining the EJB.....	1383
EJB Component Builder Screen.....	1385
EJB Settings	1388
Creating the JAR file	1389
EJB Configuration	1390
EJB Environment Definitions	1390
Resources	1391
eDeveloper Configuration and Deployment.....	1391
Environment.....	1391
Runtime	1392
Generic Messaging Layer (mgrqgnrc.dll)	1393
Broker Configuration and Deployment	1395
Query-Only Enterprise Servers	1395
Termination.....	1395
Loading enterprise servers.....	1395
Command Line Requester	1395
Termination.....	1395
Queries	1395
Remote calls from other requesters (not EJBs)	1396
J2EE and eDeveloper Installation	1396
Connection Difficulties.....	1396

23 Multi-User Considerations

Definitions	1399
Isolation Level	1399
Locks.....	1401

Process	1401
Transactions	1401
Concurrency	1401
Locking	1402
Identify Modified Row	1402
Transactions	1404
Locking Strategy	1406
Task Nesting and Locking	1408
Isolation Level	1409
Differential Update	1409
Table Modes	1410
Access Mode	1410
Share Mode	1411
Multi-User Considerations When Defining Table Modes.....	1411
How to Define Table Modes.....	1413
Table Sharing Interaction	1413
Setting the Multi-User Environment.....	1414

24 Workgroup Development

Workgroup Options	1418
Activate Team Development	1418
MVCS Snapshot File	1418
MVCS Lock File Path	1418
The Workgroup (MVCS) Menu	1421
Team Development	1422
Requirements for Team Development.....	1422
Activation of Team Development	1422
Snapshot File	1423
Concurrency	1423

Modifications to the Program Repository	1423
Lock File	1424
The Synchronization Process	1426
Application Access and Share Modes	1427
Station Lock File.....	1428

25 SQL Considerations

Configure and Define the eDeveloper Environment	1430
Windows Operation Systems	1430
Unix Operating Systems	1430
Naming Conventions - eDeveloper Gateways	1431
Gateway Name Structure	1431
eDeveloper's API Implementation and Versions	1432
Data Definition Rules	1434
Configuration and Performance.....	1435
Transactions	1435
Locking	1436
Null Value	1443
Index Definition and Usage	1444
Range Definition.....	1447
Sorting	1448
Stored Procedures	1448
Reducing Network Traffic.....	1449
Incremental Locate.....	1450
Direct SQL.....	1450
String Time Attribute Mapping.....	1450
Properties Supported by Various Gateways	1451
The eDeveloper Database Gateway for Oracle.....	1453
eDeveloper Data Types	1453

Blob Mapping Flag	1456
Oracle Data Types	1456
Long and Long RAW Data Types	1457
Hints	1457
Database Information	1458
Table Locking.....	1459
Physical Locking	1459
Views	1459
Unique Identifier	1460
NLSSORT Support	1460
Stored Procedures	1461
MSSQL Server Database Gateway	1461
eDeveloper Data Types	1461
MS-SQL Data Types.....	1464
Text Data Type	1466
Physical Locking	1466
Hints	1466
Identity Column	1467
Views	1468
Temporary Tables.....	1468
Cursors and DB Commands	1470
Database Information	1471
Informix Database Gateway	1473
eDeveloper Data Types	1473
Informix Data Types	1476
Views and Fragmented Tables	1477
Table Locking.....	1477
Physical Locking	1478
Text and Byte Data Types.....	1478

DB2 Database Gateway	1478
eDeveloper Data Types	1478
DB2 Data Types	1481
Views	1482
Physical Locking	1482
Using DB2 Handles	1483
ODBC Database Gateway.....	1484
eDeveloper Data Types	1484
ODBC Data Types.....	1487
Locking.....	1489
Troubleshooting	1489
Database Default Values.....	1489
Sort/Temporary Database	1489
Direct SQL.....	1490
ODBC Check Driver Utility.....	1490
ODBC Gateway - Data Source Information.....	1491

Index

Entries	1515
---------------	------

Welcome to eDeveloper Version 9, Magic's rapid development and deployment tool for enterprise applications. To get started, install eDeveloper and then familiarize yourself with eDeveloper's dynamic interface, as described in this chapter.

In this chapter:

- | |
|-----------------------------|
| • Documentation |
| • Online Support |
| • Installation |
| • Typographical Conventions |
| • Key Combinations |
| • eDeveloper Workspace |

Documentation

In addition to the *Reference Guide*, eDeveloper comes with the following documentation provided in PDF format:

- *How To Speak eDeveloper* - A glossary of eDeveloper terminology.
- *How To... Working with eDeveloper* - A hands-on users guide for developing eDeveloper Version 9 applications.
- *iSeries Guide* - A guide to deploying eDeveloper applications for iSeries.

Online Resources

eDeveloper includes the following online resources:

Books Online displays the complete documentation set in PDF format. Online books can be read and printed by using Adobe Acrobat Reader. Check Magic's web site, www.magicsoftware.com, for updates to eDeveloper online documentation located under *Services/Downloads/More Downloads*.

eDeveloper Context Sensitive Help provides immediate information about the fields in eDeveloper repositories, dialog boxes, and property sheets. Glossary and How To Help topics are also included.

Technical Support Online enables you to browse the knowledge base and add your own articles, post to and search user discussion forums, and submit support requests. You can access these services from the Magic web site under *Services/Support*.

Installation

The *Guide to Installation & License Management* is included with eDeveloper to provide information on different issues of installation and licensing procedures.

Typographical Conventions

The typographical conventions used throughout this book are described in the table below.

Type Style	Used for...
<i>italic</i>	Menu or folder paths. For example: <i>Workspace/Switch Panes</i> New term introduced for the first time. Names of publications, such as the <i>Reference Guide</i> .
<code>courier</code>	Any value that you must enter into a field, syntax, warnings or error messages, or environment settings. For example, <code>Appl=applic 1</code> or <code>Connection to Broker Refused</code> .
bold	Menu names and Option names. For example, select the Cancel option from the Edit menu. Buttons, such as OK , Delete , or Save .
SMALL CAPS	The names of keys on your keyboard or keyboard combinations, such as ENTER or CTRL+P .
KEY1+KEY2	A plus sign (+) between key names means hold down the first key while you press the second key. Then release both keys. For example, press ALT+F4 to exit eDeveloper and return to the Operating System.
KEY1, KEY2	A comma (,) between key names means press and release each key, one after the other.

Key Combinations

eDeveloper supports key combinations for all F keys, F1 to F12. Supported combinations are: **CTRL+F[X]**, **ALT+F[X]**, and **SHIFT=F[X]**. Key combinations are used for:

- Keyboard mapping
- System handlers
- User event triggers
- KDB literals
- Shortcuts

eDeveloper Workspace

eDeveloper Version 9 provides dynamic development repositories, a navigator, property sheets, and comments, as displayed in Figure 1-1.

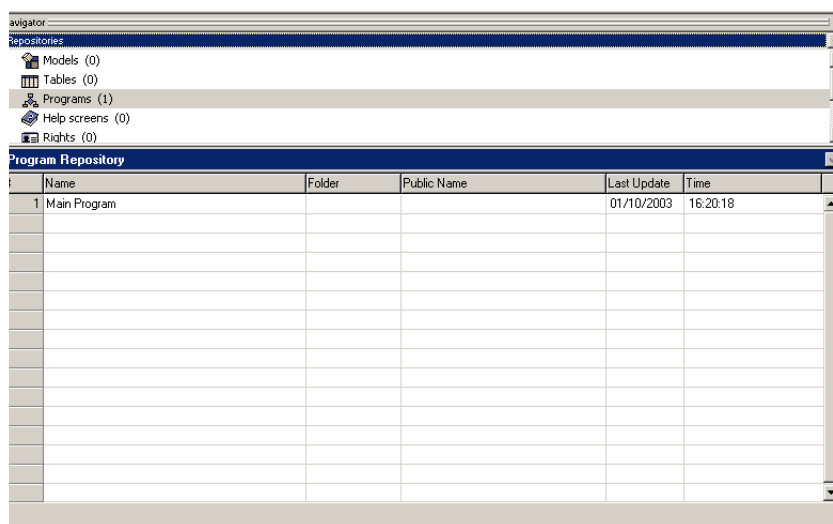


Figure 1-1 Navigator and Workspace Panes

eDeveloper uses a graphical user interface that is fully compatible with Microsoft Windows 95, 98, 2000, or NT.

Navigator and Workspace Panes

The Navigator pane is a Multiple Document Interface (MDI) screen that lets you navigate through the application.

You can click *Workspace/Navigator* to either open or close the Navigator pane. The Navigator pane automatically closes when you switch from toolkit to runtime, or when you invoke a program.

The Navigator pane displays four sections:

- Repositories - The main repositories of the application.
- Tasks - The tasks of a selected program.
- X-ref - The result list of the most current cross-reference search.
- Bookmarks - Objects that have bookmarks in an application.

Multiple Document Interface (MDI) Client Edge

The Multiple Document Interface (MDI) client appears with the Client Edge style, a 3D display with a border and a sunken edge. In some screen designs where the form is fitted to the MDI and the form contains 3D controls that are closely positioned to the form border, having the Client Edge style may result in excessive borders. You can use this environment setting to turn the Client Edge style on or off.

This environment setting should be defined in the [MAGIC_ENV] section of the Magic.ini file: MDIClientEdge = [Y/N]. The default value for this setting is Yes.

When you enter Yes, the MDI is displayed with a 3D style frame. When you enter No, the MDI will not have its 3D style frame.

Property Sheets

From the **Workspace** menu, click **Property Sheet** to display the specific properties alphabetically and by category of the selected object. Property sheets are available for fields, local variables, forms, controls, and help screens.

Comments

From the **Workspace** menu, click **Comments** or press **F10** to attach a comment box to an object in the application. You can attach a comment to the object types listed below:

Columns	Helps	Ranges
Components	Indexes	Rights
Direct SQL	I/O files	Tables
Events	Local Fields	Tasks
Foreign Keys	Models	
Handlers	Programs	

The comment is displayed in the Comment pane. Click **ALT+F10** on the repository entry to open the Comment pane.

Comments have the following characteristics:

- The Comment box can be resized to hold up to 1,000 characters.
- Comments for the object are stored in the eDeveloper Control File. The Comment text cannot be exported in the binary file of an eDeveloper Flat File.

Checker Results

From the **Workspace** menu, click **Checker Result** or press **ALT+F3** to display the current checker results generated by the Check Syntax utility. You can

specify the error level and how messages are grouped from the Toolkit Checker Minimal Level and Group Checker Messages environment settings.

The Checker Result pane can be attached to the eDeveloper MDI screen, displayed as an independent floating window, or combined in a window with the navigator, property sheet, and comment box.

For more information about checker results, see the Checker Syntax Utility section in Chapter 20, Utilities.

Switch Panes

Select the *Workspace/Switch Panes* to move the cursor from the Navigator pane to the Workspace pane.

You can click **CTRL+TAB** to cycle through every opened pane and the eDeveloper workspace. The cycle order is navigator, property sheet, comment box, and workspace.

When panes are placed together, only the front pane is in focus.

Combined Panes

The navigator, property sheet, and comments box can either appear as attached to the Workspace pane or as detached.

You can combine the objects by dragging them on top of each other, as shown in Figure 1-2. Alternately, you can select the objects from the Workspace menu.

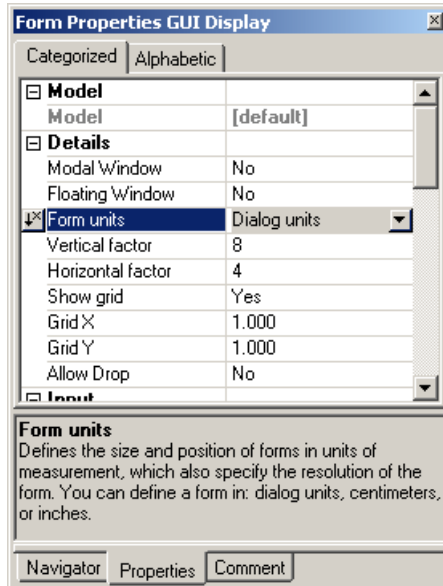


Figure 1-2 Combined Objects

You can move from object to object by clicking:

- **ALT+F1** for the navigator
- **ALT+F2** for the property sheet
- **ALT+F10** for the comment box

To separate combined objects, simply drag the title of the current option while pressing the **CTRL** key.

The combined window can either be attached to one of the eDeveloper window borders or kept as a floating window.

You can close an object by pressing the object's key combination, such as **ALT+F1**, when the object is selected.

Repositories

A repository is eDeveloper's basic screen. Each eDeveloper repository is divided into a grid, similar to a spreadsheet. Every repository has columns and rows. From certain columns, you can zoom to an option list or a details window to select data options. When the insertion point is positioned on a zoomable field, the word ZOOM appears on the message line. Figure 1-3 shows an example of an eDeveloper repository and its property sheet.

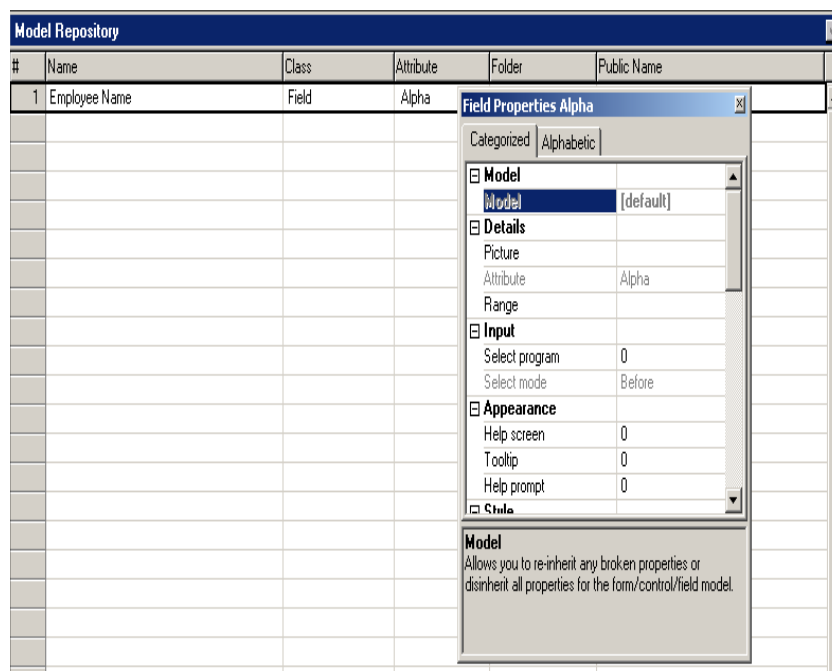


Figure 1-3 Repository and Property Sheet

You can type a data option directly into a field rather than selecting it from a list. The insertion point shows where the next text character will be placed.

Moving the Insertion Point

Information of how to move the insertion point in a repository is described in the table below.

To Move To...	Press...
The beginning of the column	HOME
The end of the column	END
The previous page	PGUP
The next page	PGDN
The top of the repository or dialog	CTRL+HOME
The bottom of the repository or dialog	CTRL+END
The end of the row	ALT+→
The beginning of the row	ALT+←

Command Options

eDeveloper provides line editing capabilities for you to enter and update data in your repositories. Whenever your insertion point is positioned on a line that can be edited, you can access the *Edit* menu or press **ALT+E** to see the line editing items available.

Edit Option	Key	Means
Cancel	F2, ESC	Cancel the previous edits. The extent of this cancellation is context-dependent. Cancel also closes the repository.
Undo Editing	ALT+BACKSPACE	Cancel edit while in the column editor.

Edit Option	Key	Means
Zoom	F5	An action that lets you jump from a cell to a selection list, dialog, combo box, details window, or property sheet. A zoomable cell is indicated on the status bar.
Wide	F6	Opens a window wide enough to view the entire column.
Go to Top	CTRL+F9	Go to the top of the development area. A development area is either in the Program repository or in the Table repository, explained in later chapters.
Jump to Row	CTRL+J	Jump to the repository row entered in the Jump dialog box. Jump to Row is useful when editing large repositories.
Create Line	F4	Add a line after this one. If your insertion point is positioned just above the first line of a repository, this will create a first line. Lines below the created line are renumbered.
Delete Line	F3	Delete this line. Confirmation will be requested. Lines below the deleted line are renumbered. In Development mode, the Create Line command also adds a line after the last line in a file.
Properties	CTRL+P	Display the property sheet for the selected repository, form, or control.

Edit Option	Key	Means
Comment	F10	Display a text box that can be used for internal messages about a selected object. You can open created comment boxes by pressing ALT+F10 .
Find Text		Lets you search for text in a repository.
Replace Text		Lets you find and replace text in a repository.
Repeat Entry	CTRL+R	With the insertion point immediately above the target line position, you can duplicate the entry of a row. Lines below that point will be renumbered.
Move Entry	CTRL+M	When the insertion point is immediately above the target line position, you can move a row from one entry line to another. Lines will be renumbered.
Overwrite Entry	CTRL+W	Replace the current entry with a copy of the line with the number you have entered in the Overwrite Entry dialog box. The original line is not changed.
Table Locate	CTRL+L	Find the first row with entries that match the search mask you have entered in the template row. You can use an asterisk (*) as a wildcard to represent any number of characters, and a dollar sign (\$) or question mark (?) as a wildcard for a single character.

Edit Option	Key	Means
Table Locate Next	CTRL+N	Continue searching the repository using the same template defined with the Table Locate command.
Next Checker MSG	CTRL+Y	Jump to the next checker message. eDeveloper highlights the current message in the Checker Results window and selects the field where the syntax error is located.
Cut Text	SHIFT+DEL	Cuts selected text.
Copy Text	CTRL+INS	Copies selected text.
Paste Text	SHIFT+INS	Pastes selected text at the current insertion point position.
Select All	CTRL+A	Select all objects.

Column and Key Internal Sequence Numbers (ISNs)

eDeveloper lets you determine how the internal sequence numbers (ISNs) should be calculated for an overwritten table entry as displayed in Figure 1-4.

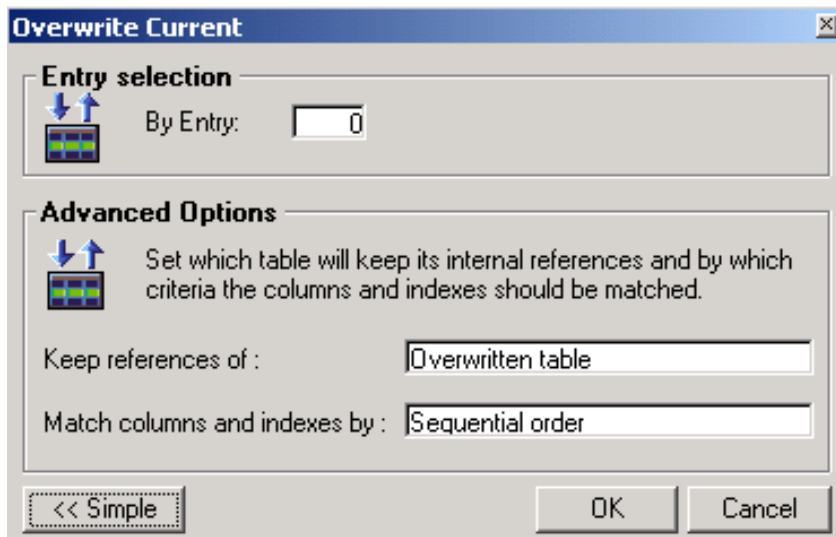


Figure 1-4 Overwrite Tables Advanced Options

When you click the **Advanced** button, the following fields appear:

- Keep references of - You can select either Overwritten table or Overwriting table.
 - Overwritten table - Keeps the ISNs that have already been defined in the overwritten table. The existing ISNs will be set for the column and indexes of the overwriting table by the value option selected from the **Match columns and indexes by** field.
 - Overwriting table - Keeps the ISNs of the overwriting table.
- Matched columns and indexes by - The options listed below affect how ISNs are set.
 - Sequential order - ISNs are set for overwriting columns and indexes by the overwritten counter-objects matched by their sequential order.

- DB name - ISNs are set for overwriting columns and indexes by the overwritten counter-objects matched by their DB name. This option is available for SQL tables only.
- Description - ISNs are set for the overwriting columns and indexes by the overwritten counter-objects matched by their Description field.

Folders

Folders are displayed under their repository in the Repository section of the navigator. Each folder represents a group of related repository objects. You can create a folder for a selected repository, by pressing **F4**.

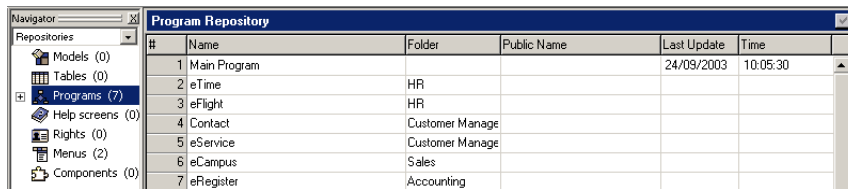


Figure 1-5 The Folder Tree for the Program Repository

The navigator lets you:

- Display or hide folders
- Enter a folder name
- View the number of objects in a folder

A folder is an object that lets you organize entries in a repository. For example, a folder may contain tables 10 to 13, and another folder may contain tables 14 to 27.

Each repository can have numerous folders. Each folder must represent a sequential range of entries in the repository. When a user drags and drops an entry from one folder to another, eDeveloper automatically rennumbers the tables to maintain an unbroken sequence of listed entries.

An entry does not need to be stored in a folder. Individual entries appear at the top of the entry list and are visible when the entire repository is visible.

#	Name	Folder	Public Name	Last Update	Time
1	Main Program			24/09/2003	10:05:30
2	eTime	HR			
3	eFlight	HR			
4	Contact	Customer Manage			
5	eService	Customer Manage			
6	eCampus	Sales			
7	eRegister	Accounting			

Figure 1-6 Programs Displayed by Folder Category

You can create folders for the following repositories:

- Table
- Program
- Help
- Rights
- Models
- Components

Click the Folder column in the selected repository to specify the folder where the entry is stored.

Folders have the following characteristics:

- You can create (**F4**), delete (**F3**), or rename a folder by parking the cursor on a folder in the Navigator pane. eDeveloper does not allow duplicate folder names.
- You can move an item from one folder to another by clicking the folder name displayed in the Folder column. Choose another folder from the list box.
- When you open the Locate dialog within a folder, eDeveloper prompts you to enter the name of the program, its folder, public name, and so on.

- Selecting an entry in another folder does not change the current folder. You must select another folder from the Folder column to change folders.
- When you generate a program from the Table repository, the Folder property lets you specify the folder where the program is placed.

Bookmarks

You can create a bookmark for an entry, shown in Figure 1-7, by clicking **Bookmark** from the **Options** menu or press **CTRL+B**.

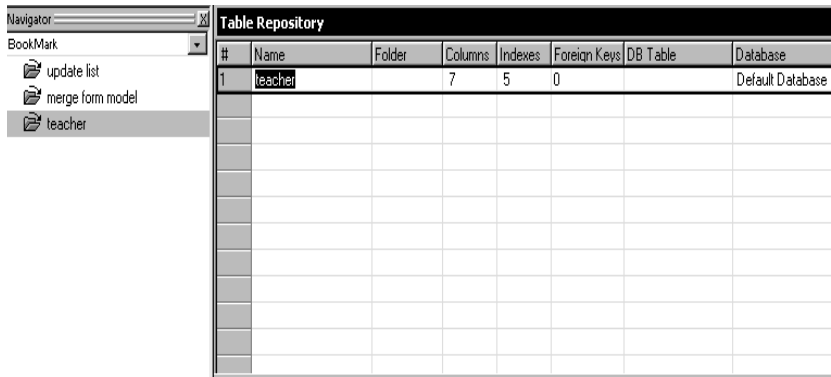


Figure 1-7 Bookmarks for an Application

Bookmarks appear in the Navigator when the **Bookmark** option from the Navigator pane is selected.

Bookmarks have the following characteristics:

- They appear in the order of their creation.
- Selecting an object moves the bookmark to the top of the list.
- You can limit the number of bookmarks created in the application by setting a value in the Maximum Number of Bookmarks environment setting under the Preferences in the Environment dialog.
- Delete a bookmark by pressing **F3**.

Cross-References

The Navigator pane has a X-ref section that lets you display multiple sets of cross-reference results. Each list displays all objects found during the cross-reference search, as displayed in Figure 1-8.

[illegible]

Figure 1-8 Results of a Cross-Reference Search

From the **Options** menu, click **Cross Reference** or press **CTRL+X** to open the Cross Reference dialog. The cross-reference results list the objects that were selected in the Cross Reference dialog.

You can click the result to display the corresponding repository and the selected entry.

Cross references have the following characteristics:

- You can delete a cross reference by pressing **F3**.
- You can store multiple result sets in the X-ref section.

For a full description, see Cross References in Chapter 20, Utilities.

Find and Replace

In Toolkit mode, you can automatically find and replace any repository object with another object of the same type. Highlight the object you wish to replace and select **Find and Replace** from the **Options** menu.

You can change eDeveloper’s default settings for property values that affect the toolkit and the runtime environments both through options on the Settings menu and directly in the files described below. You can change the names of most of these files by setting the corresponding properties in the Environment dialog. You can also use other Settings menu utilities to directly edit their content.

eDeveloper relies on several configuration files for its interface and operational profile. Some of that information is constant, while other areas are user-configurable. The main configuration file is the Magic.ini file. The Magic.ini file also holds pointers to other configuration files.

All of the configurable information of eDeveloper can be updated dynamically from the command line when you start eDeveloper. The relationship between the properties that are physically written in the Magic.ini file and those received via the command line is described in the Command Line section.

In this chapter:

• Settings Menu
• New Applications
• Application Settings
• Environment Settings
• Colors
• Fonts
• Keyboard Mapping
• Servers

• Services
• Visual Connection
• Communications
• DBMS
• Databases
• Logical Names
• Languages
• Printers
• HTML Styles
• Print Attributes
• Magic.ini File
• Command Line Options

The Settings Menu

The Settings menu includes the following repositories:

Repository	For editing properties related to...
Application	The list of applications
Environment	eDeveloper's configurable environment properties
Colors	eDeveloper's interface and application Color repository
Fonts	eDeveloper's Font Definition file
Keyboard Mapping	eDeveloper's reconfigurable keyboard interface
Servers	Available remote eDeveloper host servers' properties (for Client/Server)

Repository	For editing properties related to...
Services	Available applications on a server
Visual Communication	A display of the service connection on a server
Communications	Available communication drivers
DBMS	Available Database gateways
Database	Available physical databases
Logical Names	Logical file name translation
Languages	Available multi-lingual support
Printers	eDeveloper logical printers
HTML Styles	Available HTML styles
Print Attributes	A file that defines the connection between logical print attributes and actual printer control codes
Checker Messages	Lets you customize the level of each message for the checker
Secret Name	Protected access keys and other secure names
User Group	Definition of categories of users for security purposes
User ID	Definition of individual users for security purposes
Logon (dialog)	User ID, Password, and date

Some Settings items are available only when no application is open and some are available only when the user is logged on as Supervisor.

New Applications

You can create new eDeveloper applications by clicking New from the File menu. The New Application dialog appears, as shown in Figure 2-1.

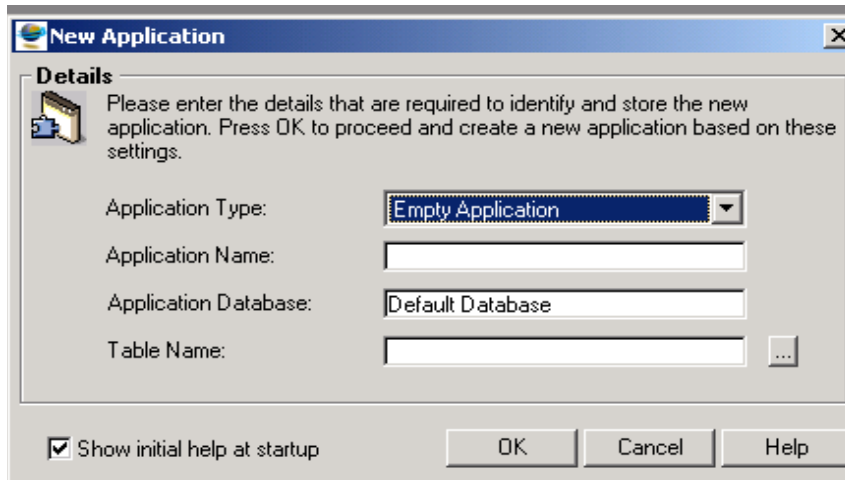


Figure 2-1 New Application

The New Application fields are:

- Application Type - Select the application type you want to create.
 - Empty Application - eDeveloper creates a new and empty application for you to start and define the application structure and logic.
 - XML Component - eDeveloper creates a new application and automatically activates the XML Component Generator, which helps you create an application that easily handles XML documents.
 - JAVA Component - eDeveloper creates a new application and automatically activates the Java Component Generator, which helps you create an application that easily accesses Java objects.
- Application Name - Enter a name for the new application.
- Application Database - Select a database from a list of current databases defined in your eDeveloper environment.

- **Table Name** - An eDeveloper application is stored as a database table. Set the name of the database table where the application information will be stored.
- **Show Initial Help at Startup** - Select to display the Getting Started help page after the application opens.

Click OK to confirm the new application details, and create and open the application.

Any new application created from the New Application dialog is added to the Application repository, which can be opened by clicking Applications from the Settings menu. The details of all applications that have been defined can be viewed and modified from this repository.

Application Settings

There are a number of locations where you can set Application settings for a specific application. These locations are the Application repository, the Application Properties dialog accessed from the Application repository, and the Application Properties dialog accessed when an application is open.

Application Repository

When all of the applications are closed, you can click *Settings/Applications* to access the Application repository. The Application repository is used to maintain the list of applications available for toolkit and runtime. eDeveloper allows access to multiple applications for toolkit and at runtime. Entries in this repository may be existing applications or declarations of new applications for toolkit.

Each entry in this repository also appears in the Application list.

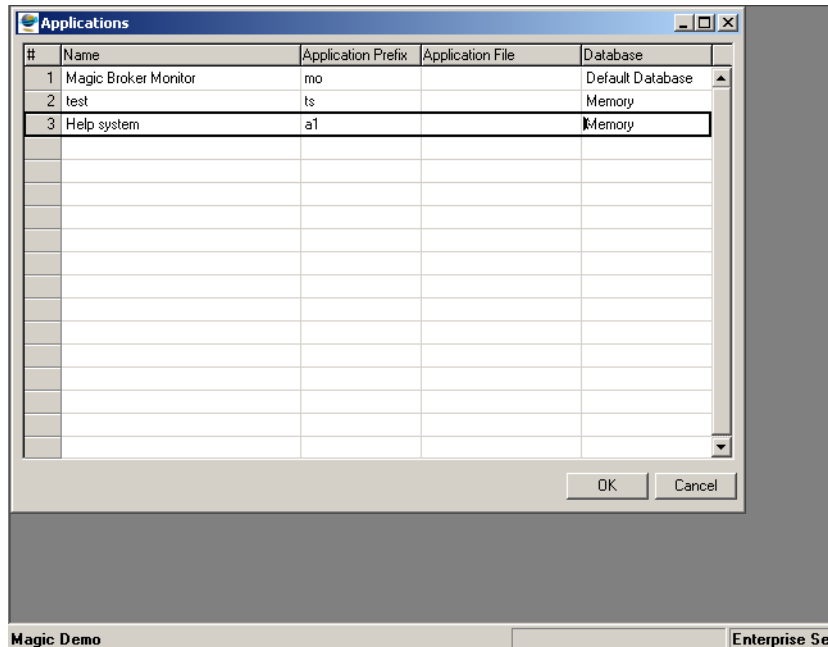


Figure 2-2 The Application Repository

Application declarations are stored in the [MAGIC_SYSTEMS] section of the Magic.ini file.

The Application repository contains the columns described below:

#

This column contains an automatically generated sequential number. You cannot edit this column.

Name

A free-text description of the application. This description will appear in the Application list, and on the message bar when the application is open for toolkit or runtime.

Application names should be unique. eDeveloper allows duplicate names, but the use of duplicate names can lead to serious confusion in development, maintenance, and deployment.

Note: Commas are not allowed in Application names because they are reserved for use as separators in the Magic.ini file.

Prefix

A 2-letter code is used as an Application identifier. The developer defines the 2-letter prefix. All default application-related repository names will be constructed using this same prefix. Some of these files are the Magic Flat File (MFF), Report files (RPR.Mcf), and all data files. For example:

Prefix	eDeveloper-Generated File Name	File Contents
pp	<i>ppCTL.MCF</i>	The eDeveloper Application file holding all application definitions
DM	<i>DMFIL001.DAT</i>	The first data table in the Table repository

The prefix may also contain default location specifications indicating where all the application files will reside (i.e. (server)path). If a location is not specified, application files will be opened in the current directory.

Prefix examples:

Prefix	Location
DM	Current directory
C:\APPL\DM	On a Windows PC, directory APPL on drive C:

If you want to override the default location for the Application or Report files, see the descriptions of the Application file and Report file properties, immediately below. You can override the default name for data tables in the Table repository.

Application (MCF) file

There is an explicit file name and location specification for the Application file. This property will override the default file name derived from the prefix. The Application file contains all of the application's elements, including all its repositories. Each application has its own MCF. If this column is blank, the Application file name will be *ppCTL.MCF*, where *pp* is the prefix specified in the Prefix property. The Application file name may include an optional location specification formatted as follows:

(server)pathname

where:

server - The eDeveloper host database server name enclosed within parentheses. This server is one of the servers in the *Settings/Server repository* described below.

pathname - The directory in which the MCF file will reside.

Zooming from the Application File property will open an Open File dialog.

Database

The access information to the physical database that stores the application's tables. eDeveloper uses the setting of the Default Database property in the Environment dialog as the default database name. To select a different database for the application's tables, select *Edit/Zoom* from the Database column to open the Database list. The Database list displays all the databases of the DBMS type selected for the application that are available to your installation.

If the Database entry shows Unknown, then the proper database gateway module was not loaded. An attempt to open an application without first loading its proper database gateway will fail.

For more information on databases, refer to the Database Settings section.

Note:

- The DBMS repository provides eDeveloper with the needed connection to the database gateway. The database gateway stores all the storage

information required for table definition. The Database repository provides eDeveloper with access information to the physical table during runtime.

- Pervasive SQL is not supported as a database for the MCF application.

Application Properties - Outside the Application

From the Application repository, you can set additional file entry information for each application, by selecting *Edit/Properties*. The Application Properties dialog will appear. The properties specified here are saved into the Magic.ini file. In this dialog, you can set the properties explained below.

Flat MFF Deployment

The eDeveloper Flat File (MFF) Deployment property lets the end user access and run specified applications that have been stored as database-independent binary files. This property is relevant only for runtime purposes because you cannot open a flat eDeveloper deployment file in toolkit mode. If the user selects this check box, the Magic.ini file is modified accordingly and information can be read from the MFF file.

Compressed

Activating the Compressed feature significantly reduces the size of the MCF of your eDeveloper application, which creates a lighter application when stored to your hard disk. An application that is to be compressed must be defined as such at the very beginning of application development, when the MCF is new (you cannot compress an existing MCF file). Compression, however, can lead to degraded application performance, and should be used only when space efficiency is a high priority.

Force MVCS Disable

The Force MVCS disable property tells eDeveloper whether or not to disable the MVCS commands that allow for workgroup development for this specific application.

Selecting this check box limits the eDeveloper application to one developer at a time.

Leaving the check box blank allows multiple developers to work concurrently on the eDeveloper application.

Access Key

The Access Key property specifies a password to be used for every access to the application control file (MCF). The content of the access key is passed to the underlying database's security mechanism to actually handle file security. When you specify an access key for an application control file, update access to that file from external applications or from other eDeveloper systems will require that the program trying to access the table specify the same access key. If the access key is not provided by the requesting program, the underlying database will bar access to the Application repository. The access key is also used as an encryption seed for the Application repository encryption.

Whenever an editing session on the Application repository terminates, eDeveloper scans the Application repository for possible changes to an access key in any of the entries. If such a change is detected, eDeveloper will prompt for confirmation of the addition or modification of the access key. In case such confirmation is given, the Application repository is encrypted immediately, using the new access key value. Note that this feature is dependent on the support of the underlying database.

Because the Application repository is stored in the Magic.ini file, which is an unsecured file, you should use a secret name for the access key. Using a secret name protects the contents of the access key from unauthorized users. For more information on secret names, refer to the Secret Names section in Chapter 13, Authorization System.

Browser Client Task Cache

The Browser Client Task Cache improves the performance of the browser client application by caching logical segments of the XML page on the client side, minimizing the data transmission from the server to client.

For every browser-client task, the engine generates an XML file describing the task's logic. The result page of the task includes a link to the task logic XML file.

As long as the application or revision number is not modified, the name of the XML file of each task remains the same. This way the end-user's browser caches the logic segment of each task as it activates that task for the first time. Consecutive executions of the browser-client application retrieves the logic segments of each task from the browser's local cache.

Cached File Revision

Any modification of a deployed application including its components may result in changes of the logic segments of the tasks. The XML name of each task logic segment should be modified to differentiate it from the already cached files to enable the end-user's browser to load the new logic segments of the modified application.

One of the elements that construct the name of each XML file is the revision of the application. The revision of the application is part of the application environment settings and can be accessed through the Application properties dialog of an entry in the Applications list.

Whenever any part of a deployed application and its components is modified, the revision setting of that application should be modified as well. Failing to modify the revision may cause a mismatch in the logic segments cached on the browser.

Note: There is no need to modify the revision for every modification in development time. In development time, a file is created using a new name for every single execution.

Browser Client Cache Path and Alias

The XML files that keep the logic segments of each task are created and stored in the path set by the Browser Client Cache path environment setting. This setting can be found under the Server tab of the Environment dialog.

A corresponding web alias should be set on the web server and in the eDeveloper environment. The typical installation of eDeveloper automatically sets proper values for these settings. This setting can be found under the Server tab of the Environment dialog.

Application Properties - Within the Application

When an application is open, you can set additional property settings in another Application Properties dialog, by selecting *File/Apl. Properties*.

These properties are saved into the specific eDeveloper application file and override the system-wide eDeveloper environment settings, specified in the Environment dialog.

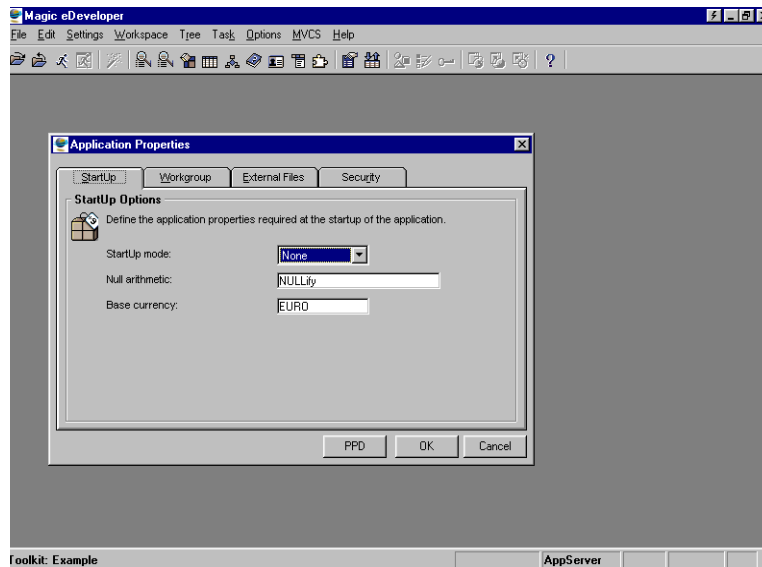


Figure 2-3 The Application Properties Dialog

The Application Properties dialog contains four tabs as described below:

StartUp Tab

StartUp Mode

This property is used to override the Application Startup Mode setting, specified in the Environment dialog, which indicates whether an eDeveloper application will open in runtime, toolkit, or background mode. Use this property to specify a different startup mode for a given application.

The default value is None.

Null Arithmetic

This property setting is used to modify eDeveloper's Null support. Some databases, especially SQL, allow parameters and arguments with null values. This setting indicates how expressions involving one or more null-values are to be computed:

- NULLify - the result will also be a null value.
- Use Default - the result will be computed with the Null property's assigned default values.

The eDeveloper defaults can be overridden at the Model and Column levels.

Base Currency

Zoom from the Base Currency property to access the Currency list. This list, as shown in Figure 2-4, displays all currencies entered in the European Currency table. Euro is the Base Currency default.

External Files Tab

Print Attribute File

This property allows you to alter logical printer attributes defined in the file specified in the Environment dialog, and to save the new printer attributes in a different file. This property ensures that within a multi-application environment, an application-specific printer attribute file is created.

HTML Style File

This property allows you to alter HTML Style tags defined in the file specified, and to save the HTML style tags in a different file. This property ensures that within a multi-application environment, an application-specific HTML Style file is created.

Color Definition File

This property allows you to modify the color settings of the Color Definition file specified in the Environment dialog, and to save the new color settings in a different file. This property ensures that within a multi-application environment, a color definition file is created.

Font Definition File

The Font Definition File property allows you to modify font settings defined in the file specified in the Environment dialog, and to save the font settings in a different file. This property ensures that within a multi-application environment, a font definition file is created. For a full explanation of the Font Definition file, refer to the Font section in the Settings chapter.

Keyboard Mapping File

The Keyboard Mapping File property allows you to specify alternate keyboard mapping in the Application Properties, and to override the Keyboard Mapping file, if any, specified in the Environment dialog. To create an alternate keyboard mapping file, use the Keyboard Mapping utility. In the Keyboard Mapping utility, when basing new mapping configurations on previous ones,

save the new configuration under a new file name in the File Save dialog's Save As property.

Internet Development File Root

This property specifies the location of the Internet Development file.

European Currency Conversion File

Zoom from the European Currency Conversion File property to access an Open File dialog. Then specify the European Currency file.

Security Tab

Application Access Key

The Application Access Key property appears only when the Application Properties dialog is accessed by owners of this key. This property setting holds the name of the application access key. If such a key has been declared, access to the application for development or for runtime purposes is restricted to owners of the key. Other users or developers will not be able to open the application.

Public Rights Access Key

The Public Rights Access Key parameter appears only when the Application Properties dialog is accessed by the owner of this key. This setting holds the name of the public rights access key.

By creating a public rights access key, a developer can improve security for a given application. The supervisor's ownership of this key frustrates any attempt by an unauthorized user to remove system password protection by destroying the security file (user_std.eng), which contains all user passwords, and then to login as the supervisor. A bogus supervisor, without this key, will not be able to find out which rights are associated with which keys in a particular application. After zooming from the Rights column of the User ID repository, a bogus supervisor will not be able to see the Rights Name field, which describes the rights associated with given keys, nor be able to zoom from the Key field into the full Rights list.

Super Right Key

The Super Right Key property appears only when the Application Properties dialog is accessed by the owner of the Super Right Key. This property holds the name of the Super Right Key.

The Super Right Key is created by the supervisor for an application to give rights to all of the application's activities, for both development and runtime purposes. Therefore, a holder of the Super Right Key does not need to hold separate rights to individual activities within the application.

See Application Security Issues in the Authorization System chapter for more detailed information.

Force MVCS Key

The Force MVCS Key property appears only when the Application Properties dialog is accessed by the owner of the Force MVCS Key. This field holds the name of the Force MVCS Key.

The supervisor can create a Force MVCS Key for an application, giving holders the rights of limiting access of a Workgroup transaction to one user.

Remote Flow Monitor Right

You can restrict Remote Flow Monitor access at runtime by specifying flow monitor rights. Zoom from the property to select rights from the Allowed Rights list.

PPD (Programmable Protection Device)

The Programmable Protection Device button is a zoom point to the PPD dialog. The PPD dialog provides the facility to program a password into a special protection module. Using this password in conjunction with a protection device lets you create applications that have an additional measure of protection, beyond what the authorization system provides.

A confirmation dialog appears when the insertion point leaves the PPD Content field on the PPD dialog. Select No to cancel the changes entered in the PPD dialog.

Selecting Yes in the Save Changes dialog causes the updated PPD information to be written into the PPD plug, and the insertion point returns to the Application Properties dialog.

Environment Settings

The Environment dialog contains all of the global configurable eDeveloper properties. These properties reside in the [MAGIC_ENV] section of the Magic.ini file. You can use Environment properties to customize eDeveloper according to the specific needs of the installation. All changes made to properties in the Environment dialog are registered in the Magic.ini file. Some of the properties take effect immediately, while others will be effective from the next eDeveloper session.

Note: eDeveloper Environment properties are not related to Operating System Environment variables.

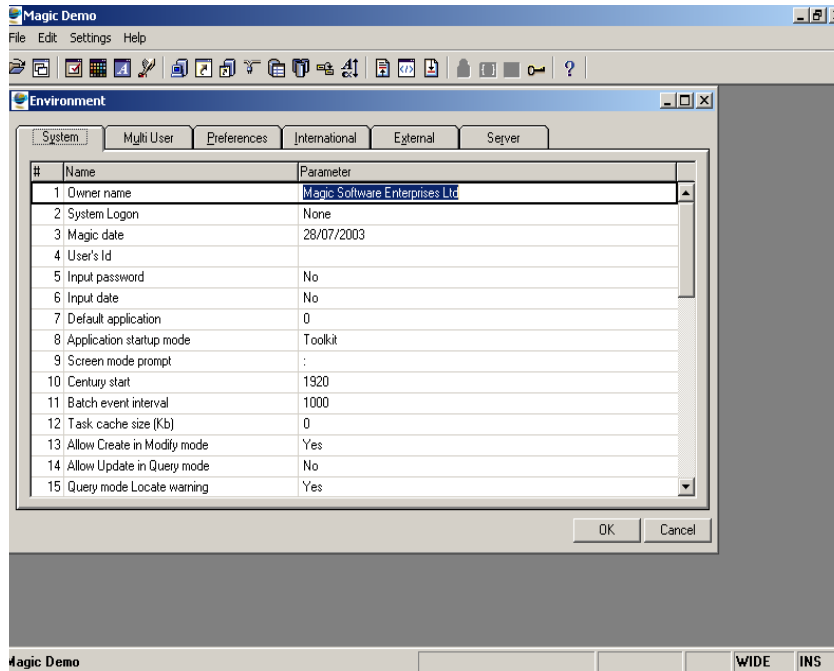


Figure 2-5 Environment Dialog

When activating eDeveloper, it is sometimes desirable to use environment variables as parameter values on the command line. To achieve this, eDeveloper must be activated from a batch file. For example, to pass the Terminal environment property value to eDeveloper using the command line, create a batch file using the following command:

```
mggenw /terminal=%terminal%.
```

Run this batch file instead of running the eDeveloper executable. If eDeveloper is invoked with Command Line properties, the values that appear in the Environment dialog will reflect the Command Line properties for the respective components. Command Line entries take precedence over the Magic.ini values. However, any modifications you make to the Environment dialog during a toolkit session will also automatically update the Magic.ini file, and will override any corresponding, previously entered Command Line values.

Refer to the Command Line section for a complete description of Command Line effects. A full description of the Environment properties dialog follows below. The table at the end of this chapter lists all the Environment properties with their corresponding Magic.ini and Command Line names.

The Environment dialog is organized by six tabs:

- System
- Multi-User
- Preferences
- International
- External
- Server

Note: The setting shown in parentheses is the default setting.

In addition to the properties you can set in the Environment dialog, you can define NULL display strings and a default date value, but only in the 'MAGIC_DEFAULTS' section of the Magic.ini, as described on page 124.

System

The environment settings below appear under the System tab of the Environment dialog.

Owner Name: (Magic Software Enterprises Ltd)

A string of up to 30 characters, intended to contain the Owner Name of the eDeveloper application. This value can be queried by the Owner function.

Change effective: Immediate

Magic.ini and Command Line name: Owner

System Logon

eDeveloper lets you select from the following System Logon options:

- None - eDeveloper does not prompt the user for a logon name and password.
- User Name - eDeveloper prompts the user for a user name and password. eDeveloper checks the user name with the names entered in the Security file (usr_std.eng), and applies the rights assigned to the user. Note that the user name can be up to 20 characters.
- Active Directory - eDeveloper uses the logon name and password entered when logging onto Windows to retrieve the user's rights as defined in the Active Directory. The main benefit to using the Active Directory is that the supervisor no longer has to maintain the user names and rights in the eDeveloper security file.
- LDAP - eDeveloper uses LDAP (Lightweight Directory Access Protocol) to authenticate the user's identity by using an LDAP server. eDeveloper automatically locates the groups, which the user is a member of, in the LDAP server database. These same groups should be defined in the Security file (usr_std.eng), so that eDeveloper can apply the rights of these groups to the user. For more information, see LDAP Address and LDAP Connection environment settings.

The user password in the eDeveloper user file does not have to correspond to the operating system password. The password can be up to 30 characters.

It is best to keep a unique eDeveloper password for each user to prevent users from trying to log on as different users.

Change effective: Next Session

Magic.ini and Command Line name: SystemLogin

Magic Date: (System Date)

eDeveloper provides access to two date settings. One of them is the operating system date, referred to as *System Date*. The other is a user-provided date and is stored by eDeveloper. This date is the *Magic Date*. The Magic Date may be queried by the MDate function. This setting may also be changed in the Logon dialog.

Set and use the Magic Date value whenever you need to use a past or future date for the execution of a program. In these cases the Magic Date will be stored in the Magic.ini file.

Change effective: Immediate

Magic.ini and Command Line name: Date

User's ID: (None)

The User's ID setting holds the ID of the current eDeveloper operator. This ID, in conjunction with the operator's password, is used to verify the operator's access rights and privileges within the application. The User's ID property can be queried in the application by the USER function. The User ID value may be changed in the Logon dialog. This setting does not have a Magic.ini and Command Line name.

Input Password: (No)

Valid values: Yes, No

Yes means eDeveloper will prompt for a User ID and Password at logon time, in the Logon dialog.

No means that User ID and Password prompts will not appear in the Logon dialog at logon time.

Change effective: Next session

Magic.ini and Command Line name: InputPassword

Input Date: (No)

Valid values: Yes, No

Yes means eDeveloper will prompt for a date at logon time, in the Logon dialog. This date is the Magic Date described above.

No means that the date prompt will not appear in the Logon dialog at logon time.

Change effective: Next session

Magic.ini and Command Line name: InputDate

Default Application: (0)

A number identifying an application in the Application list, to be started automatically upon entry to eDeveloper. If a valid application number is found while eDeveloper starts up, the Startup screen will be bypassed and the application will be opened.

Change effective: Next session

Magic.ini and Command Line name: StartApplication

Application Startup Mode: (Toolkit)

Valid values: Toolkit, Runtime, Background

This setting defines the mode eDeveloper will use when opening an application. The possible modes are:

Toolkit - Open the application ready for toolkit work. This setting has no effect in the Runtime version of eDeveloper.

Runtime - Open the application in the Runtime environment in order to use it as an end-user application.

Background - This mode allows the execution of a batch program without any interaction with the user. In Background mode, eDeveloper will not open any windows but will execute the program "silently". For text-based applications only, if an error occurs while the program is executing in Background mode, the error message is written to the Operating System's standard output device. On termination of the program, eDeveloper returns to the operating system.

Change effective: Next Session

Magic.ini and Command Line name: ApplicationStartup

Screen Mode Prompt

The Screen Mode Prompt setting defines the default for variable prompts in screen mode forms. The setting value's syntax is

Suffix

where **suffix** is a string to be appended to the variable's description when eDeveloper creates a screen mode display. For example, the variable Customer Number with Picture N,5 will get the following screen mode prompts based on the settings shown. Space characters are shown as ^. If a blank is required as

the first character of a setting value, it needs to be explicitly declared using a backslash (\), as in the following example:

Screen Mode Prompt	Form Display
:	Customer Number:
\^:	Customer Number^:

Change effective: Immediate

Magic.ini and Command Line name: ScreenModePrompt

Century Start: (1920)

The Century Start setting enables eDeveloper to interpret all dates that are input using a two-character mask for year component in pictures of date fields, such as MM/DD/YY.

The Century Start setting specifies the year that serves as the crossing between centuries when interpreting date settings. Any two-digit value of a year component less than the year represented by the Century Start setting causes the date to be interpreted as a 21st century date. Any value of a year component greater than or equal to the year represented by the Century using a two-character mask for the year component in pictures of date setting causes the date to be interpreted as a 20th century date. For example,

For Century Start = 1960,

01/20/70 is interpreted as 01/20/1970

01/20/50 is interpreted as 01/20/2050

for Century Start = 1980

01/20/70 is interpreted as 01/20/2070

01/20/50 is interpreted as 01/20/2050

01/20/90 is interpreted as 01/20/1990

Change effective: Immediate

Magic.ini and Command Line name: Century

Batch Event Interval: (1000)

The time interval during a batch task execution in which the eDeveloper engine checks the event queue and handles pending events. This setting is given as the number of milliseconds of the interval. When a Record Event Interval has been set for a task, the eDeveloper engine will check for pending events for each record interval and for each time interval as set by the Batch Event Polling Interval setting.

Change effective: Immediate

Magic.ini and Command Line name: BatchPaintTime

Task Cache Size: (0)

The task cache is memory storage used to store tasks read from the database. Using the task cache is recommended to speed up the process for commonly used tasks.

You can specify the cache size (KB) in the Environment dialog. The value 0 renders the task cache inactive.

Change effective: Immediate

Magic.ini and Command Line name: TaskCacheSize

Allow Create in Modify Mode: (Yes)

Valid values: Yes, No

To allow the creation of new records when operating in Modify mode. **Yes** allows the end-user to create new records. **No** blocks the end-user from creating new records.

Change effective: Immediate

Magic.ini and Command Line name: AllowCreateInModify

Allow Update in Query Mode: (No)

Valid values: Yes, No

To allow the update of existing records when operating in Query mode. This refers to an update by the Update operation within the task, and not to an update by the end-user at the keyboard. **No** blocks the updating of existing records. **Yes** allows the updating of existing records while in Query mode.

Change effective: Immediate

Magic.ini and Command Line name: AllowUpdateInQuery

Query Mode Locate Warning: (Yes)

Valid values: Yes, No

In ISAM and SQL databases, when performing a query mode locate on non-key properties in runtime, the message "Locate on non-index columns..." appears. Note that this message appears for ISAM databases when the number of records exceed 5,000.

Yes preserves backward compatibility. **No** means that the message never appears.

Change effective: Immediate

Magic.ini and Command Line name: LocateModeQueryWarning

Allow Access to Application: (Yes)

Valid values: Yes, No

Controls access to the Application repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessApplications

Allow Access to Environment: (Yes)

Valid values: Yes, No

Controls access to the Environment dialog.

Change effective: Immediate

Magic.ini and Command Line name: AccessEnvironment

Allow Access to Colors: (Yes)

Valid values: Yes, No

Controls access to the Color repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessColors

Allow Access to Fonts: (Yes)

Valid values: Yes, No

Controls access to the Font repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessFonts

Allow Access to Keyboard Mapping: (Yes)

Valid values: Yes, No

Controls access to the Keyboard Mapping repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessKeyboardMapping

Allow Access to Servers: (Yes)

Valid values: Yes, No

Controls access to the Server repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessServers

Allow Access to Services: (Yes)

Valid values: Yes, No

Controls access to the Services repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessServices

Allow Access to Visual Connection: (Yes)

Controls access to the Visual Connection display.

Change effective: Immediate

Magic.ini and Command Line name: AccessVisualConnection

Allow Access to Communications: (Yes)

Valid values: Yes, No

Controls access to the Communication Driver repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessCommunications

Allow Access to DBMS: (Yes)

Valid values: Yes, No

Controls access to the DBMS repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessDBMS

Allow Access to Databases: (Yes)

Valid values: Yes, No

Controls access to the Database repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessDatabases

Allow Access to Logical Names: (Yes)

Valid values: Yes, No

Controls access to the Logical Name repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessLogicalNames

Allow Access to Languages: (Yes)

Controls access to the Language repository.

Valid values: Yes, No

Magic.ini and Command Line name: AccessLanguages

Allow Access to Printers: (Yes)

Valid values: Yes, No

Controls access to the Printer repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessPrinters

Allow Access to HTML Styles: (Yes)

Valid values: Yes, No

Controls access to the HTML Style repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessHTMLStyles

Allow Access to Print Attribute: (Yes)

Valid values: Yes, No

Controls access to the Print Attribute repository.

Change effective: Immediate

Magic.ini and Command Line name: AccessPrintAttributes

Allow Access to Logon: (Yes)

Valid values: Yes, No

Controls access to the Logon dialog.

Change effective: Immediate

Magic.ini and Command Line name: AccessLogon

Allow Access to Toolkit: (Yes)

Valid values: Yes, No

This setting controls the availability of the eDeveloper Toolkit. If **Yes** is specified for this setting, access to the eDeveloper Toolkit for toolkit purposes is allowed.

No will prevent access to the eDeveloper Toolkit.

eDeveloper will start an application in Runtime mode whenever this setting is set to No, overriding the value of the Application Startup Mode setting.

Change effective: Immediate

Magic.ini and Command Line name: AccessToolkit

Allow Testing Environment: (No)

Valid Values: Yes, No

A **Yes** value tells eDeveloper to interface with the testing environment to report all information on controls.

Change effective: Next Session

Magic.ini and Command Line name: AllowTesting

Allow Access to Checker Messages: (Yes)

Valid Values: Yes, No

Controls access to the Checker Messages table.

Change effective: Immediate

Magic.ini and Command Line name: AccessCheckerMessages

Temporary Tables Path

eDeveloper uses temporary files to store intermediate values during its operation and then automatically clears them. The size of the temporary files may, in rare cases, be up to several hundred kilobytes, causing I/O operations on a standard disk to be slower than you want. Use this setting to specify an alternative path, such as a RAM drive, for better performance. If no value is specified, temporary files are written to the current directory.

Note: In operating systems without memory constraints, eDeveloper keeps the temporary information in memory instead of writing it to temporary files.

Change effective: Next session

Magic.ini and Command Line name: TempPath

Maximum File Handles: (0)

Setting Maximum File Handles to a value greater than 0, changes the number of file handles allowed by the operating system.

Change effective: Next session

Magic.ini and Command Line name: FileHandles

License

This setting controls the behavior of the eDeveloper engine according to the properties found on the license server or license file. If eDeveloper does not find the appropriate feature name in the license file, it will issue an error message and abort the session.

The default is set by the installation program.

Change effective: Next session

Magic.ini and Command Line name: LicenseName

License file

This setting specifies the location of the license server, through a Host/Port combination, and also provides an alternate license file that can be found on an accessible file system. In the event that eDeveloper cannot locate the primary License file, it will use the alternate License file specified. If eDeveloper does not find a license file, it will issue an error message and abort the session.

The default is set by the installation program.

Change effective: Next session

Magic.ini and Command Line name: LicenseFile

Load Monitor

When set to Yes, the Flow monitor will be loaded on opening an eDeveloper application. This setting enables you to open the Flow monitor upon executing an eDeveloper application from a background enterprise server.

Change effective: Next session

Magic.ini and Command Line name: LoadMonitor

Flow Monitor Output File

When the Flow monitor is loaded through a background server, the entire log of the application activities may be outputted to a given file. Use this setting to set the path and name of the Monitor Output file.

Change effective: Next session

Magic.ini and Command Line name: Monitor2File

Remote Flow Monitor

When the Remote Flow Monitor is set to Yes, in runtime the engine can be viewed by a remote flow monitor. The engine sends messages to the monitor.

Change effective: Next Session

Magic.ini and Command Line name: RemoteFlowMonitor

Remote Flow Monitor Port

This setting specifies the port number used by the engine to process incoming connection requests from a remote flow monitor.

Change effective: Next Session

Magic.ini and Command Line name: RemoteFlowPortNumber

Multi-User

The environment settings below appear under the Multi-User tab of the Environment dialog.

Terminal: (0)

This setting is relevant in a multi-user environment, where it is used to assign a unique numeric identifier to each end-user terminal. This setting may be queried from within a program, using the Term function. For more information on the Term function, refer to Chapter 8, Expression Rules. The Terminal value can be used by programs wherever there is a need to create some unique resource for the user, such as a disk file. The Terminal number may then be used for generating a resource name unique to the user.

The default value of zero for the Terminal setting instructs eDeveloper to resolve unique resource allocation automatically.

Change effective: Immediate

Magic.ini and Command Line name: Terminal

Multi-User Access: (Yes)

Valid values: Yes, No

The Multi-user access setting enables the SQL gateway to perform locks in the underlying database (in addition to eDeveloper locks if requested). It is advisable to set the Multi-user Access setting to Yes, especially when eDeveloper locks are not used or when applications other than eDeveloper are accessing the database.

The Multi-user access setting is also used to enable Team Development.

Yes means that eDeveloper will implement concurrency controls on all database table access, allowing table-sharing while maintaining database integrity.

No instructs eDeveloper to open all tables for exclusive use (overriding task table open mode), assuming that this is the only session accessing the table, hence database locks are not issued.

For more information on the multi-user environment, refer to Chapter 24, Workgroup Development.

Change effective: Immediate

Magic.ini and Command Line name: MultiUser

ISAM Transactions: (No)

Valid values: Yes, No

eDeveloper always uses the services of the underlying database for transaction processing. When ISAM files are utilized, however, transaction use is optional. Whenever a transaction is defined in the program, eDeveloper checks the database of each table participating in the program. If the table originates from an ISAM database, then based on this setting, eDeveloper decides whether or not to apply transaction processing. The recommended value setting is Yes.

Change effective: Immediate

Magic.ini and Command Line name: ISAMTransaction

Deadlock Prevention: (No)

Valid values: Yes, No

eDeveloper provides a built-in mechanism for deadlock prevention, which is implemented for the ISAM-type databases that do not feature a deadlock detection or prevention feature. Deadlock situations occur when two users are each waiting for a resource held by the other. For example:

Station A	Station B
Locks table x	Locks table y
Tries to read table y	Tries to read table x

Station A holds table x in a transaction and tries to read table y held by station B. Station B is doing the same in reverse order.

Yes in the Deadlock Prevention setting will prevent such deadlocks.

No in the Deadlock Prevention setting will not take any special action to prevent deadlocks.

Using Deadlock Prevention is recommended for databases that do not have built-in detection or prevention facilities. Refer to the relevant eDeveloper Database Gateway documentation for specific information on the database used by your system. Note that deadlock prevention can cause performance degradation.

Change effective: Immediate

Magic.ini and Command Line name: DeadlockPrevent

Server Communication Interval: (0)

The interval, in seconds, during which eDeveloper will perform a communication check with all of its eDeveloper servers for tables being accessed by the Client/Server.

This check is required. The eDeveloper server will remove the process serving the client after a predefined period of inactivity.

This setting should be coordinated with the timeout values of the eDeveloper Server so that communication will occur before the server times out.

A value of 0 means that no communication checks will be carried out.

Magic.ini and Command Line name: ServerTimeout

Lock File: (mglock.dat)

This setting specifies the name of the file eDeveloper uses to implement the lock mechanism. Do not use path names for the Lock File setting, because they are supplied automatically. For more information about the lock file, refer to Chapter 24, Workgroup Development.

Change effective: Next session

Magic.ini and Command Line name: LockFile

ISAM - Force Locking Within Transaction: (Yes)

Valid values: Yes, No

The flag is relevant for toolkit behavior only and for ISAM files.

When the flag is set to Yes the following behavior occurs:

In the import phase, if the lock issued in the task is not within a transaction, the Transaction Begin property is changed to enforce the locking within an open transaction.

The default setting of the transaction in a task with an ISAM file will be changed as described in the table below:

Force Locking Within Transaction	Batch Task	Online Task
Yes	Task Prefix	On record lock
No	None	Before record update

Note: This table is relevant only for physical transactions. When the task transaction mode is changed from deferred to physical, the Transactions Begin property value changes according to the table.

When the ISAM - Force Locking Within Transactions property is set to Yes, the table below describes how the Transaction Begin and Locking Strategy property values coincide.

	Prefix	Suffix	Update	None	On Lock
Immediate	X				X
On Modify	X				X
After Modify	X				X
Before Update	X	X	X		X
None	X	X	X	X	X

Change effective: Immediate

Magic.ini and Command Line name: LockWithinTran

Resource Lock File: (mgres.loc)

This Environment property specifies the path of the lock file to be used by the resources locking utility lock file. eDeveloper cannot use any of the database lock files because there is one for each database definition.

Change effective: Next session

Magic.ini and Command Line name: ResourceLockFilePath

Preferences

The environment settings below appear under the Preferences tab of the Environment dialog.

Default Database

The name of the database to be used by eDeveloper as the system default. This default is used for application, report, and database tables, and can be overridden at these levels. This name must be selected from among the databases defined in the Database repository for this installation. For more

information on the Database repository, refer to the Database Properties dialog section in this chapter.

Change effective: Immediate

Magic.ini and Command Line name: DefaultDatabase

Database for Sort/Temporary

The name of the database to be used by eDeveloper for creating sort tables or other temporary database tables needed as temporary storage, such as the Direct SQL result table. For example, a fast ISAM database that uses memory for storage may be used to provide improved performance for sorts, when the original tables are stored in a traditional disk-based database. This name must be selected from the databases defined in the Database repository for this installation. To see the Database list, select *Edit/Zoom* (F5). DB2 and ODBC cannot be a sort database.

Note that if a location for a sort table's database has been specified in the Location setting of the Database repository, it will always be used. The path specified in the Database for Sort/Temporary setting will be used only if the Location setting in the Database repository is blank.

Change effective: Immediate

Magic.ini and Command Line name: TempDatabase

Range/Locate Box Popup Seconds: (10)

The duration, in seconds, to elapse before the initial display and then the subsequent update of the Range or Locate Popup window. Range or Locate popup windows appear whenever a sequential search is performed on a database table, and they display the number of records already searched. Specify 0 for this setting if you want to prevent the window from appearing at all.

Change effective: Immediate

Magic.ini and Command Line name: RangePopTime

Sort /Temp Box Popup Seconds: (10)

The duration, in seconds, to elapse before the initial display and then the subsequent update of the Sort/Temp Box popup window. The Sort/Temp Box popup window appears whenever a Sort operation is performed on a database table. The Sort/Temp Box popup window displays the number of records already sorted. Specify 0 for this setting if you want to prevent the window from appearing.

Change effective: Immediate

Magic.ini and Command Line name: TempPopTime

Keyboard Idle Seconds: (1)

The duration, in seconds, between idle signals when there is no keyboard activity. During interactive sessions, eDeveloper awaits user input. If no such input is available, an idle signal is raised to allow other processes to take place. The Keyboard Idle Seconds value is the inactivity time before an idle signal is raised.

This setting affects the intervals of the Idle function returned value.

Change effective: Immediate

Magic.ini and Command Line name: IdleTime

Pulldown Menu Close Timeout: (0)

The Pulldown Menu Close Timeout setting determines whether the pulldown menus, when opened, will remain open until the user executes a user action, a selection or subsequent close, or will automatically close after a predefined timeout. The setting specifies the duration, in seconds, that pulldown menus will remain displayed. A value of 0 means the pulldown menu will wait indefinitely for the next user action.

If a non-zero value is set, then pulldown menus will close automatically after the specified number of seconds, if there is no user activity.

Pulldown menus, when open, halt the execution of other processes in the system, such as events. Give the Pulldown Menu Close Timeout setting a non-zero value when you do not want this effect.

Change effective: Immediate

Magic.ini and Command Line name: MenuCloseTimeout

Confirm When Auto-Exiting: (No)

Valid values: Yes, No

When you modify any of eDeveloper's dialog settings or repository columns and you Exit the dialog or the repository, eDeveloper will display a Save Changes? dialog. In some situations, the Exit action is implied by some other choice you make. For example, while editing an Operation repository in a task, you may select *Workspace/Tables* (Shift+F2) to switch to the Table repository. In this case, eDeveloper receives an internal Exit request from the Task Operation repository. If you have already modified the Operation repository, the Save Changes? dialog will be displayed before control is passed to the Table repository. Such an internal Exit action is called an *Automatic Exit*.

No in this field means that eDeveloper will not display the Save Changes? dialog when in an Auto Exit mode due to a user request. In such a case, the changes are automatically accepted without interruption.

Yes in this field means that eDeveloper will display the Save Changes? dialog whenever an automatic exit occurs, to request explicit user confirmation for every modified dialog or repository along the path between the current *context*, repository, or dialog being modified, and the requested context. For example, suppose the current context is a Task Event repository that was invoked as an object while the Operation repository was being edited, and both the Task Event and Operation repositories have been modified. If at this point you want to edit the Table repository, by selecting *Workspace/Tables*, eDeveloper will first pop up the Save Changes dialog for the Event repository and then the Save Changes dialog for the Operation repository. eDeveloper will move to the Table repository only after you have responded to both of these dialogs.

Change effective: Immediate

Magic.ini and Command Line name: ConfirmAutoExit

Task Flow Modification: (Free)

Valid values: Free, Safe

This setting affects the behavior of eDeveloper during the editing of the Task Operation repository.

Free means a user can modify any operation line freely. In this mode, if an operation code is unintentionally overwritten, the setting values of the operation are blanked. Free mode allows faster editing of the Operation repository.

Safe mode of operation will not allow any modification of an operation code after it has been accepted, although any of the operation's properties can be modified. When in Safe mode, the only way to change an operation itself is to delete it and then to re-enter it.

Change effective: Immediate

Magic.ini and Command Line name: FlowModify

Display Copyright Messages: (Yes)

Valid values: Yes, No

The Display Copyright Messages setting controls the display of the eDeveloper copyright messages, and can be used for customizing the deployed runtime application.

Yes in this field will display all the eDeveloper copyright messages, as published by Magic Software Enterprises.

No has the following effect, depending on whether eDeveloper is running in toolkit mode or runtime mode:

- In the toolkit mode, all logos still appear.
- In the runtime mode, all notices of eDeveloper are removed. These include: status line, menu bar, registration screen, loading message.

Note: The message in the Help/About dialog remains the eDeveloper message unless the Deployment Custom Copyright, described next, is used to replace it.

Change effective: Next Session

Magic.ini and Command Line name: CopyrightMessages

Deployment Custom Copyright: (none)

The Deployment Custom Copyright setting contains a text message to be displayed to the user of the runtime mode of eDeveloper when running applications. The text value of this setting will be displayed in the following places:

- The eDeveloper registration dialog, which appears immediately when eDeveloper loads, replacing the eDeveloper copyright message.
- The Help/About dialog, replacing the eDeveloper copyright message.

Note: the copyright message may span over several lines. Use the ‘\’ sign to indicate a new line within the message text.

Change effective: Next session

Magic.ini and Command Line name: RTUserCopyright

Resident Magic.ini: (No)

Valid values: Yes, No

Yes means that the Magic.ini file contents will be resident in memory at all times. The amount of memory needed to keep the Magic.ini file resident is roughly equal to its operating system size. When resident, read access to Magic.ini variables is faster. Updates to the Magic.ini file will be written directly to disk, even if the file is resident, after the user exits from eDeveloper.

No means that every query or update of the Magic.ini file involves reading and writing it to disk.

Change effective: Next session

Magic.ini and Command Line name: ResidentINI

Display Toolbar: (Yes)

Valid values: Yes, No

This setting controls the appearance of the Toolbar in runtime. If Yes is specified for this setting, the Toolbar will be visible.

No in this setting will prevent appearance of the Toolbar in runtime.

Change effective: Next session

Magic.ini and Command Line name: RtToolBarGUI

Load Resident Tables: (No)

Valid values: Yes, No

This setting is a global setting that enables or disables the loading of resident tables. If this setting is defined as No, the loading of resident tables is disabled. If the setting is defined as Yes, the loading of resident tables is enabled, but it is still necessary to specify for each table if it is to be loaded as a resident table.

Change effective: Next session

Magic.ini and Command Line name: LoadResidentTables

Display Full Messages: (Yes)

Valid values: Yes, No

The underlying DBMS may return errors, such as constraints violations, during the application execution. eDeveloper keeps a buffer containing the last error received from the DBMS.

Setting Display Full Messages to Yes causes eDeveloper to display the error message in Runtime, and to clear the buffer.

If you do not want the end-user to see the error messages, change Display Full Messages to No. You can then use DbERR or ErrDbmsMessage functions to

put the error message into a variable as a string, and to manipulate the error message in the program.

i

Setting Display Full Messages to Yes causes the DbERR function to return an empty string, because the error messages buffer is cleared when messages are displayed. The recommended setting value depends on the application.

Change effective: Immediate

Magic.ini and Command Line name: DisplayFullMsgs

Center Screen in Online: (No)

Valid values: Yes, No

The Center Screen in Online setting enhances performance in online tasks with a table control that accesses database tables.

- This setting affects tasks where the Main table is browsed using a two-way index or no index.
- Only online tasks that have a Table control on the form are influenced by this setting.

When performing the operations described below, eDeveloper in earlier versions would redraw the screen by positioning the current record in the center of the screen, reading records before and after this record to fill the rest of the screen. This resulted in the opening of two cursors; one to read forward and one to read backward from the current record position. The Center Screen in Online setting will instruct eDeveloper to position the current record in these situations at the top of the screen, eliminating the need to open a cursor backward from the now current record position, as these records will no longer be displayed in the refreshed screen.

This setting will affect the behavior in the following situations:

- Changing the task mode from Modify, Query, Locate, Index, or Sort to Modify or Query.

- After a Locate Next operation.
- After a successful Query mode Locate.
- Behavior Exceptions:
 - Changing from Range mode with or without a new range parks on the first record of the range.
 - Changing from Sort mode with a new sort parks on the first record of the range.
 - Changing from Create mode parks on the first record of the range.

The changed behavior is driven by the eDeveloper engine itself, and should be the same for different gateways.

Change effective: Immediate

Magic.ini and Command Line name: CenterScreenInOnline

Reposition After Modify: (No)

Valid values: Yes, No

Only online tasks that have a table control on the form are influenced by this setting.

When you enter **Yes**:

When changing the value of a setting that is one of the segments of the main index of the task, or when inserting a single new record in Modify mode only, the following actions occur on exiting the record:

- eDeveloper writes the record back to the database.
- eDeveloper rereads the records of the current screen from top to bottom. This causes the changed record to be redisplayed in its new position, or to disappear from the screen if it was outside the screen.
- eDeveloper parks on a new record according to the action, such as Down Arrow, PgDn, etc., that was used to exit the record.

When you enter **No**:

The second operation, rereading the screen, does not occur. This causes the record to remain in its old position, even though it is now out of sequence. The record will maintain its position until scrolled outside the screen.

The Refresh Task Window setting overrides the value of this setting. If the setting is set to Yes, the screen will be reread from the database every time a record is updated, whether or not the main index values change.

Note: This setting can cause the same physical record to appear twice on the same screen due to lack of refresh.

Change effective: Immediate

Magic.ini and Command Line name: RepositionAfterModify

Indent Character: (0)

This setting defines the indent spacing for the indentation in HTML pages and RTF controls.

Change effective: Immediate

Magic.ini and Command Line name: IndentCharacters

Default Color

This setting determines the colors of new entries you create in the Color repository and the colors that will be used during deployment when a color defined in an application is not found.

When you create a new entry in the Color repository, eDeveloper assigns the values you have specified in the Default Color setting. If no Default Color has been specified, eDeveloper assigns the following colors:

- Foreground color: System color - Window Text
- Background color: System color - Window Background

When deploying an application, if a color defined in the application does not exist, eDeveloper uses the colors specified in the Default Color setting. If no Default Color has been specified, eDeveloper uses the above colors.

Change effective: Immediate

Magic.ini and Command Line name: Default Color

Default Font: (0)

When you create a new entry in the Font repository, eDeveloper uses the value you have specified in the Default Font setting. If no value has been specified, eDeveloper assigns the MS Sans Serif 8 point font.

When deploying an application, if a font defined in the application does not exist, eDeveloper uses the font specified in the Default Font setting. If no value has been specified, eDeveloper uses the Windows default font.

Change effective: Immediate

Magic.ini and Command Line name: DefaultFont

Tooltip Timeout: (5)

The Tooltip Timeout setting determines the number of seconds that a tooltip is displayed, either on expressions while you are developing an application or on controls during deployment. The maximum value is 31 seconds. For values over 31 seconds, the behavior of Windows is unpredictable.

Change effective: Immediate

Magic.ini and Command Line name: TooltipTimeout

Maximum Number of Bookmarks: (10)

This setting defines the number of kept bookmarks. All bookmarks for each application are stored in the Windows registry according to the application name. The bookmarks are available when the toolkit is closed and then reopened.

Change effective: Next session

Magic.ini and Command Line name: Bookmarksnumber

Maximum Number of X-refs: (5)

This setting defines the maximum number of kept cross references. All cross references for each application are stored in the Windows registry according to the application name. The cross references are available when the toolkit is closed and then reopened.

Change effective: Next session

Magic.ini and Command Line Name: MaxCrfResults

Retry Operation Time Interval: (600 seconds)

This setting specifies the time, in seconds, in which eDeveloper will retry a defined operation.

The default retry time is 600 seconds (10 minutes).

The value 0 in this setting represents None, that is no retry.

Change effective: Immediate

Magic.ini and Command Line name: RetryOperationTime

IO Device Open Timing

Valid values: Immediate, On Demand

The timing of opening an IO device can be controlled in two ways: immediately as the task that defines it is opened, or on Demand, when the first output or input operation to the IO device occurs or when an IO-device-related function, such as EOF, EOP, Line, or Page is used. The default option is Immediate.

Immediate – When set to Immediate, any IO device will be opened when the task that defines the IO is opened.

On Demand – When set to On Demand, an IO device will be opened at the first execution of an output or an input operation that is set for the IO or for the evaluation of an IO-device-related function, such as EOF, EOP, Line, or Page.

Floating Palettes Always On Top: (Yes)

When set to No, eDeveloper's floating palettes (the Navigator, Property sheets, and the Comments box) will close when you press **Esc** or **ENTER**. The cursor returns to the eDeveloper workspace.

When several palettes are located on the same floating window. **Esc** or **ENTER** from one of the palettes closes the entire palette window.

When set to Yes, eDeveloper's floating palettes remain open even when you press **Esc** or **ENTER**. The cursor returns to the eDeveloper workspace.

This option does not affect docked palettes.

Change effective: Immediate

MAGIC.INI and Command Line name: PalettesAlwaysOnTop

Dockable Palettes: (Yes)

When set to Yes, you can drag-and-drop a floating palette to a border of the eDeveloper workspace to dock the palette.

When set to No, a floating palette cannot be docked to the border of the eDeveloper workspace.

This option does not affect already docked palettes. Once a docked palette becomes undocked, however, and Dockable Palettes is set to No, the palette cannot be docked again until Dockable Palettes is set to Yes.

Change effective: Immediate

MAGIC.INI and Command Line name: DockablePalettes

Single Expand Palettes: (No)

When set to No, you can have all sections of the property sheet expanded at the same time.

When set to Yes, only one section in the property sheet can be expanded. Expanding a collapsed section will collapse the currently expanded section.

Change effective: Immediate

MAGIC.INI and Command Line name: SingleExpandPalettes

Property Sheet Automatic Handling: (Close)

This setting determines whether eDeveloper automatically opens or closes the property sheet that relates to a specific eDeveloper repository.

The options are:

- None - eDeveloper does not automatically open or close the property sheet.
- Open - eDeveloper automatically opens the property sheet when a relevant eDeveloper repository is accessed. However, eDeveloper does not automatically close the property sheet.
- Close - eDeveloper closes the property sheet when eDeveloper returns to a location where the property sheet is no longer relevant. The property sheet remains closed even when eDeveloper returns to a repository that is relevant to the property sheet.
- Full - eDeveloper opens the property sheet whenever it becomes relevant and closes the property sheet when it is no longer relevant.

When the property sheet is opened automatically, the focus remains on the workspace and does not return to the property sheet.

Change Effective: Immediate

MAGIC.INI and Command Line Name: AutomaticPropertySheet

Image Cache Size: (0)

This setting lets you control the maximum memory size for caching the displayed images of an application.

The Image Cache Size is displayed in kilobytes. Zero kilobytes means that there is no limit to the Image Cache Size.

When the image cache is about to exceed the defined limit, the least used images are removed from the cache to make room for new images.

Change Effective: Next session

MAGIC.INI and Command Line name: ImageCacheSize

Check Image Change Time: (No)

This setting determines the method of how an image is cached.

When Check Image Change Time is set to Yes, the image cache method is able to detect image file modifications by creating a time stamp for each image file. Before a cached image is displayed, eDeveloper checks the time stamp. If the time stamp differs from the currently cached image, the image is reloaded.

If Check Image Change Time is set to No, the time to display the cached image is much quicker because eDeveloper does not perform any additional disk I/O operations.

Change effective: Next session

MAGIC.INI and Command Line Name: ImageCacheCheckTime

Toolkit Checker Minimal Level

Choose the minimal level by which the syntax checker will check your application.

- Error - The checker will display only the error messages.
- Warning - The checker will display only the error and warning messages.
- Recommendation - The checker will display the error and warning messages, and recommendations.

Change Effective: Immediate

Magic INI and Command Line Name: CheckerLevel

Group Checker Messages By

This setting determines how checker messages are grouped in the Checker Results window. The checker message types are:

- Object - Grouped by eDeveloper object. For example: Models, Tables, and Programs. Checker messages are sorted by the order in which they are found by the Syntax Checker.

- Type - Grouped by checker message type: Error, Warning, or Recommendation. Checker messages are sorted by the order in which they are found by the Syntax Checker.
- Object and Type - Grouped by eDeveloper object and then by checker message type.

Change Effective: Immediate

Magic INI and Command Line Name: CheckerGroups

Jump Automatically to First Item in Checker List

This setting determines if eDeveloper automatically highlights the first checker result message entry and parks in the field where the error occurred.

Change Effective: Immediate

Magic.ini and Command Line Name: CheckerJumpAuto

International

The environment settings below appear under the Prefences tab of the Environment dialog.

Date Mode

Valid values: American, European, Scandinavian, Buddhist

This setting specifies the default mode of the date used throughout the system, according to the following scheme, using the example February 1, 1993:

- American date format is MM/DD/YY, displays as 02/01/93
- European date format is DD/MM/YY, displays as 01/02/93
- Scandinavian date format is YY/MM/DD, displays as 93/02/01
- Buddhist date format is DD/MM/YY, displays as 01/02/36, because the year is increased by 543.

Change effective: Immediate

Magic.ini and Command Line name: DateMode

Thousands Separator (,)

This setting defines the character eDeveloper will use at runtime as the delimiter between the thousands in a displayed numeric variable. The default character is a comma. When in toolkit mode, eDeveloper expects a comma as the thousands separator in picture definitions, regardless of the definition of this setting.

Change effect: Immediate

Magic.ini and Command Line name: ThousandSeparator

Decimal Separator (.)

This setting defines the character that eDeveloper will use at runtime as the delimiter between the whole and decimal parts in a displayed numeric value containing decimals. The default character is a period. When in toolkit mode, eDeveloper expects a period as the decimal separator in picture definitions, regardless of the definition of this setting.

Change effective: Immediate

Magic.ini and Command Line name: DecimalSeparator

Date Separator (/)

This setting defines the character that eDeveloper will use at runtime as the divider between month, day, and year, of displayed date settings. The position of the date components is defined by the Date Mode setting, explained above. When in toolkit mode, eDeveloper expects a slash (/) as the date separator in picture definitions, regardless of the setting's value.

Change effective: Immediate

Magic.ini and Command Line name: DateSeparator

Time Separator (:)

This setting defines the character that eDeveloper will use at runtime as the delimiter between the hours, minutes, and seconds parts of a time value in

displayed values. The default character is a colon. When in toolkit mode, eDeveloper expects a colon as the time separator in picture definitions regardless of the setting's definition.

Change Effective: Immediate

Magic.ini and Command Line Name: TimeSeparator

External

External files for an application can now be saved and retrieved from a server by entering the name of the external file in the required External Files environment settings.

Logo File: (None)

This file is optional and specifies the location and filename where the system logo data, displayed in the opening screen, is saved. The file should be a bitmap file.

Change effective: Next session

Magic.ini and Command Line name: LogoFile

Const File: (mgconstw.eng)

The Const file is a required file.

The Const File setting specifies the location and filename where all eDeveloper's constant data is stored. The Const file includes all the user interface and language information for eDeveloper.

Change effective: Next session

Magic.ini and Command Line name: ConstFile

Help File: (mghelpw.hlp)

This file is optional. The setting specifies the location and filename where all the eDeveloper system (not Application) help text is stored.

Change effective: Next session

Magic.ini and Command Line name: HelpFile

Color Definition File: (clr_std.eng)

The Color Definition file is required.

This setting specifies the location and filename of the file where eDeveloper's color definitions are saved. Zooming from this entry will open a Color Definition repository, showing the definitions stored in the named Color Definition file.

Refer to the description of the Color Definition file for more information.

Change effective: Next session

Magic.ini and Command Line name: ColorDefinitionFile

Font Definition File: (fnt_std.eng)

This setting specifies the location and filename of the file where eDeveloper's font definitions are saved. Zooming from this entry will open a Font Definition repository, showing the definitions stored in the named Font Definition file.

For more information, refer to the description of the Font Definition repository.

Change effective: Immediate

Magic.ini and Command Line name: FontDefinitionFile

Keyboard Mapping File: (act_std.eng)

The Keyboard Mapping file is required.

The Keyboard Mapping File setting provides the location and filename of the file where eDeveloper's keyboard assignment definitions are saved.

Select *Edit/Zoom* To view the Keyboard Mapping repository.

Refer to the description of the Keyboard Mapping file for more information.

Change effective: Immediate

Magic.ini and Command Line name: KeyboardMappingFile

Documentation Template File: (doc_std.eng)

The Documentation Template file is required.

This setting specifies the location and filename of the file where eDeveloper can find the templates for the application documentation utility. eDeveloper provides extensive self-documentation of all its repositories. The format and content of the output is controlled via the template file. Two templates are supplied with eDeveloper. The standard template (doc_std.eng) provides an abbreviated output with essential information about the documented object. The extended template (doc_ext.eng) provides full information about the documented object.

Change effective: Immediate

Magic.ini and Command Line name: DocumentTemplateFile

HTML styles file: (html_stl.eng)

This setting specifies the location and filename of the file containing the HTML style templates for the HTML documentation utility.

Change effective: Immediate

Magic.ini and Command Line name: HTMLStyles

Print Attributes File: (prn_std.eng)

The Print Attributes file is optional. This setting provides the location and filename of the file containing the definitions of the logical print attributes used within eDeveloper applications. A printer attribute is a logical object that may be used on eDeveloper output forms and where used, translates to a sequence of printer control codes during output in runtime. The Print Attributes file defines the connection between the logical attributes and the actual escape codes defined in external printer command files. For more information, refer to the sections on Print Attributes and Printers in this chapter.

Select *Edit/Zoom* to reach the Print Attribute repository, where you can define logical print attributes.

Change effective: Immediate

Magic.ini and Command Line name: PrintAttr

Security File: (usr_std.eng)

The Security file is required. If not specified, or if it does not exist, eDeveloper will create a default empty file.

This setting specifies the location and file name of the file holding all of the secure system information, including:

- User IDs, passwords, and their rights
- User Groups and their rights
- Secret Names

The security file is encrypted to prevent unauthorized entry into the system. If the property is set to ANSI, then the file is considered ANSI and no conversion is done. If the property is set to OEM, then the file is considered as OEM, and the data is converted from screen to screen.

Note: The security file must reside in a shared directory when in a multi-user environment because it contains global information for all users.

Change effective: Next session

Magic.ini and Command Line name: UsersFile

Startup Security File

This setting specifies the name and operating system location of an eDeveloper security file to be loaded upon the initialization of eDeveloper. After initialization is complete, the file is replaced in memory by the file specified in the Security File setting. The reason for the seeming duplication is in Client/Server considerations. It may be necessary for the Security file in a Client/Server environment to reside on the server where it can be common to all the users.

The Security file (as opposed to the Startup Security file) contains the Secret Name repository. The Secret Name repository may include the definitions of the User ID and Password logical names specified in the Server Properties. The User ID and Password are required to logon to the server. The Security file,

however, is not accessible before logon. The Startup Security file is designed therefore to provide the information required for the logon.

If eDeveloper is not running in Client/Server mode, or if the User ID and Password are not specified in the Server Properties dialog, leave this field empty.

Change effective: Next session

Magic.ini and Command Line name: StartupUsersFile

The OEM2ANSI Translation File

This file maintains the location and file name of the external file that holds the translation table for eDeveloper internal character symbols and the character table of the platform. The OEM2ANSI Translation file lets the user translate from OEM to ANSI and from ANSI to OEM, or for whatever character set translation the user requires. If this file is not set, then the default function for conversion is from OEM to ANSI. The default filename is OEM2ANSI.

The OEM2ANSI Translation setting appears in the Column properties sheet and in the I/O repository. When the user clicks Yes, any external update to the object is translated from OEM to ANSI and any export of this object is translated from ANSI to OEM. When the user clicks No, no translation occurs.

Change effective: Next session

Magic.ini and Command Line name: OEM2ANSIFile

ANSI to Unicode

Use this setting to define the name of the encoding table to be used by the HTML page of a browser task. When this setting has a defined value, each HTML page created by a browser task instructs the page to use the defined character set by embedding a META tag:

<META HTTP-EQUIV="Content-Type" content="text/html; charset=XXX">

where **XXX** is the string taken from the Environment setting.

This information is also used to instruct the browser client module to use the defined encoding table for translating the entered data that has been passed to the eDeveloper enterprise server engine.

If your HTML page already has a META tag, this tag remains unchanged. However, the browser client module will still be instructed to use the encoding table defined by the Environment setting.

Note: The encoding table defined in this setting is case sensitive.

Change effective: Immediate

Magic.ini and Command Line name: Ansi2Unicode

Alternate Collating Seq File

This setting specifies the location and filename of the file that holds an Alternate Collating Sequence file. A Collating Sequence file holds information required for sorting values, such as an alphabetical sequence. An Alternate Collating Sequence file may be used to alter the default table used by an underlying database manager. Therefore its use is dependent on the underlying database support of such a switch. Refer to your database gateway documentation for more information.

You can also define separate Alternate Collating Sequence (ACS) files for specific database management systems (DBMSs) or individual databases, if they support the feature. A database-specific ACS file can be declared in the Alternate Collating Sequence setting of the DBMS and Database dialogs of the DBMS repository and the Database repository. Any database for which no database-specific Alternate Collating Sequence file has been defined will use the ACS file defined in the Magic.ini file.

Note that if an ACS file is defined for a database but the ACS file is not present, the database will use the ACS file defined in the Magic.ini file. eDeveloper does not support two different ACS files for one DBMS.

Change effective: Next session

Magic.ini and Command Line name: CollatingFile

Starting Language

In a Multi-lingual Support (MLS) environment, this setting should specify the language to be used at startup. An MLS environment does not support multi-line edits and rich text controls.

Valid values: Empty, or a language selected from the Language repository.

Change effective: Immediate

Magic.ini and Command Line name: StartingLanguage

Checker Messages Table File

This setting lets you specify the file path for the Chk_std.dat file. The Chk_std.dat file provides information displayed in the Check Messages table.

Change effective: Immediate

Magic.ini and Command Line name: CheckerMessageTable

European Currency Conversion File

This setting contains the location of the European Currency Conversion file, if there is one. Zoom from the European Currency Conversion File setting to the European Currency Conversion Table. This table is an external ASCII file that holds currency values based on the euro currency.

Valid values: Empty or a valid path name

Change effective: Immediate

Magic.ini and Command Line Name: EuropeanCurrencyConversionFile

Drop Data Supported User Formats

To handle drop data of user-defined formats, you must first set the user-defined format names. This setting instructs the eDeveloper engine to retrieve drag-and-drop operation data in user-defined formats, separated by a comma. For example, if your user-defined formats are ABC and EFG, you should enter ABC,EFG.

Change effective: Immediate

Magic.ini and Command Line Name: DropUserFormats

Command Processor: (command.com)

This setting, using a full OS directory path, specifies the program used to execute user exits - the command processor. The default setting is given according to the operating system on which eDeveloper was installed.

Change effective: Immediate

Magic.ini and Command Line name: CommandProcessor

HTTP Proxy (address:port)

This setting specifies the location of your HTTP proxy. The HTTPPost function can connect through a proxy server.

Change Effective: Immediate

MAGIC.INI and Command Line Name: HTTPProxyAddress

HTTP Timeout

This setting determines the time, measured in tenths of a second, which the HTTPGet and HTTPPost functions will wait for a response. If the value is 0, the HTTP Timeout will be two minutes. The timeout cannot exceed two minutes.

Change Effective: Immediate

Magic.ini and Command Line Name: HTTPTimeout

Print Data HTML Template

This setting lets you specify an HTML template that is used when generating an HTML file from the Print Data Wizard.

Change Effective: Immediate

Magic.ini and Command Line Name: PrintDataHtmlTemplate

Print Data XML Template

This setting lets you specify an XML template that is used when generating an XML file from the Print Data Wizard. The XML template should be an XSL file.

Change Effective: Immediate

Magic.ini and Command Line Name: PrintDataXmlTemplate

WSDL Files Path

(Program Files/Common Files/Magic/WSDL file.wsdl)

This setting is used to enter a location where the WSDL file will be saved. The WSDL file describes the Web Services provided. The WSDL file is only created when an eDeveloper application provides Web services. eDeveloper will use *Program/Files/Common Files/Magic* as a default value if no other location is entered.

A WSDL file is created through the Component Builder, as described in Chapter 14, Components.

Change effective: Immediate

Magic.ini and Command Line name: WSDLFilesPath

Mail Connection Timeout (default: 0)

This setting defines the timeout duration in seconds for connecting to a mail server from the MailConnect function. If eDeveloper cannot connect to the mail server within the defined timeout duration, the MailConnect function will fail.

Change effective: Immediate

Magic.ini and Command Line name: MailConnectionTimeout

Mail Operation Timeout (default: 0)

This setting defines the timeout duration in seconds for any mail operation using the various mail functions described in Chapter 8, Expression Rules, Expressions. If eDeveloper does not complete the mail function within the defined timeout duration, the function will fail.

Change effective: Immediate

Magic.ini and Command Line name: MailOperationTimeout

SNMP Database Connections Utilization Threshold (default: 0)

This setting lets you enter a percentage number that determines the number of exceeded open connections permitted for a DBMS, as specified in the DBMS Maximum Connection property, before eDeveloper sends a trap message.

Change Effective: Immediate

Magic.ini and Command Line Name:
DatabaseConnectionsUtilizationThreshold

LDAP Address:Port

This setting lets you enter the LDAP directory address and port number. For example, 127.0.0.1:389

Change Effective: Immediate

Magic.ini and Command Line Name: LdapAddress

LDAP Connection String

When a user binds to an LDAP server (System Logon = LDAP), a Distinguished Name (DN) and password is sent. The LDAP Connecting String is used to specify the user's DN, which is a unique entry identifier in the LDAP server database, for example: cn=John, ou=users, dc=mycompany, dc=com. You can use the \$USER\$ string as an alias for the user name entered in the Logon screen or the user name value returned by the Logon function, for example: cn=\$USER\$, ou=users, dc=mycompany, dc=com. The \$USER\$ string is automatically replaced by the user name. The password entered in the Logon screen or the password value returned by the Logon function will be used for authentication as well.

Note: You can choose to define two secret names, LDAP_USER and LDAP_PASS, for the user name and password. The \$USER\$ alias in the LDAP Connection String will be substituted with the value of the LDAP_USER secret

name and the LDAP_PASS secret name will be used for the password when authenticating the user's identification on the LDAP server without using the Logon screen or the Logon function.

Change Effective: Immediate

Magic.ini and Command Line Name: LdapConnectionString

LDAP Domain Contexts

Use this setting to specify the search base that would be used to locate the groups where the LDAP user is a member of. for example:
ou=groups,dc=mydomain. For accessing the Microsoft Active Directory, use one of these naming contexts, for example, dc=mycompany,dc=com.

Change Effective: Immediate

Magic.ini and Command Line Name: LdapDomainContext

LDAP Timeout

This setting is used to specify the number of seconds eDeveloper waits when trying to access the LDAP server. If eDeveloper does not receive an answer from the LDAP server before the timeout interval expires, an error message is displayed. The default value is 120 seconds.

Change Effective: Immediate

Magic.ini and Command Line Name: LdapTimeout

SSL CA Certificate Files

Use this environment setting to determine the intermediate Certificate Authority (CA), such as Certificates Service of Microsoft (CertSrv), used for all HTTP interactions. Certificate authorities are separated by the semi-colon symbol (;).

The **SSL CA Certificate doesn't exist: [Name of the SSL CA Certificate]** error message appears, during the first access to the client certificate, when the delimited SSL CA Certificate doesn't exist.

Change Effective: Next Session

Magic.ini and Command Line Name: SSLCACertificateFile

SSL Client Certificate Files

Lets you browse for a PKCS12 certificate (*.pfx). The selected client certificate will also be assigned to every Call Web Service and HTTP Call.

The following client certificate errors can appear on the status bar for an open application:

- Certificate does not exist.
- Certificate is not in PKCS12 format.
- Password is incorrect.

The ClientCertificateDiscard function discards the certificate during runtime. When switching from runtime to toolkit, the application will again register the certificate, making it part of every Web Service and HTTPS call in runtime.

Change Effective: Next Session

Magic.ini and Command Line Name: SSLClientCertificateFile

SSL Client Certificate Password

You should enter a password for a private key in a client certificate.

Change Effective: Next Session

Magic.ini and Command Line Name: SSLClientCertificatePassword

Server

The environment settings below appear under the Server tab of the Environment dialog.

Activate Enterprise Server: (No)

Valid Values: Yes, No

This setting controls whether eDeveloper will try to connect to a request broker as an enterprise server. If set to **Yes**, eDeveloper will register itself with the broker and accept requests for program execution.

Change effective: Next session

Magic.ini and Command Line name: ActivateRequestsServer

Messaging Server (Default Broker)

This setting specifies the location of the Request Broker to connect to when eDeveloper is used as an enterprise server. When specifying only a port number, the broker is assumed to reside on the same host as the eDeveloper engine.

An enhanced Client/Server interface called the Generic Messaging Layer provides greater flexibility for the calling application to connect to a variety of heterogeneous networks. The Generic Messaging Layer allows the Request Client to issue the same call through either the eDeveloper Broker, The MQSeries Manager, or other defined messaging servers.

Change effective: Next session

Magic.ini and Command Line name: MessagingServer

Enterprise Server Can Change Application: (Yes)

Valid Values: Yes, No

If set to **Yes**, the enterprise server will be able to service requests for program execution of multiple applications. When a request for an application that is not open is received, the current open application will be closed, and the requested application will open.

If set to **No**, a request for an application that is not the active application will fail.

Change effective: Next session

Magic.ini and Command Line name: RequestsServerCanReplaceCtl

HTTP Requester

This field contains the URL and the name of the eDeveloper Internet requester module. This URL hyperlinks to an eDeveloper Program to specify the name and location of the eDeveloper Internet Requester module.

The default is set by the installation program.

Change Effective: Immediate

Magic.ini and Command Line name: InternetDispatcherPath

Web Document Alias

This setting is used to define the alias to use for creating temporary files on the Internet server. This alias will be added to all of the URL's temporary files so that the Web server can access them in its own address space.

The default is set by the installation program.

Change Effective: Immediate

Magic.ini and Command Line name: WebDocumentAlias

Web Document Path

This path is used to select images, Java, and ActiveXs, in a location that relates to the runtime Web server root. It is also used for creating an HTML file for the View in Browser option in the HTML Form Command palette.

The default is set by the installation program.

Change effective: Immediate

Magic.ini and Command Line name: WebDocumentPath

Requester Timeout: (0)

This setting specifies the time within which the broker returns a signal to the requester. When the time between signals exceeds the maximum time set, an error message appears. The default value of 0 implies an infinite timeout.

Change effective: Next session

Magic.ini and Command Line name: RequesterTimeout

Maximum Number of Concurrent Requests

This setting specifies the maximum number of threads that the enterprise server will be allowed to create. The number of threads is limited only by the licence and machine capacity.

Change effective: Immediate

Magic.ini and Command Line name: MaxConcurrentRequests

Load Balancing Priority

This setting defines the priority of the eDeveloper engine as an enterprise server. You can select a numeric value from 1 to 5, where 5 is the highest priority and 1 is the lowest priority. A request is directed to idle engines assigned with a higher priority.

Change effective: Next Session

MAGIC.INI and Command Line name: LoadBalancingPriority

Web Authoring Tool

The Web Authoring Tool setting specifies the path of an external Web Authoring Tool that is used to edit HTML Merge forms.

Change effective: Immediate

Magic.ini and Command Line name: AuthoringToolPath

Context Inactivity Timeout: (600)

This setting determines the time, measured in tenths of a second, that a user who is executing a browser-based program will stay connected to the eDeveloper application while actually being inactive. A value of 0 means that there is no limitation on being connected while inactive.

Change effective: Immediate

Magic.ini and Command Line name: ContextInactivityTimeout

Post Context Unload Timeout: (1200)

This setting determines the timeout of a context when the task page is switched to another page using the browser (such as Back, Forward, or a new URL). If the timeout has not expired, returning to the previous page reloads the last positioned context. If the timeout value has expired, the context for the previous task page is terminated.

The timeout value increments are defined in tenths of seconds. The default value is 1200 tenths of seconds. If you set the value as 0, no timeout duration is set and the context is cleared immediately.

Change effective: Immediate

Magic.ini and Command Line name: ContextUnloadTimeout

Persistent Browser Client Module

Activating a browser client application while keeping the browser client module persistent on the end user machine improves the time required to load the browser client module.

Set this property to Yes to let the end user keep the browser client module persistent on the local hard disk.

Set this property to No for a new browser client to be downloaded for every initialization of a browser application.

Change effective: Immediate

MAGIC.INI and Command Line name: UseSignedBrowserClient

Browser Client Sub-Version

Use this property to modify the expected names of the browser client module file (e.g. MGBC920_01.cab & MGBC920_01.js)

The string defined in this setting will be added as the suffix of the expected browser client module file names.

For example: if set to 'xyz' then the expected file names in version 9.20 SP1 will be MGBC920_01xyz.cab & MGBC920_01xyz.js.

This option should be used when you wish to use different names for your browser client module files.

Change effective: Immediate

MAGIC.INI and Command Line name: BrowserClientSubVersion

Browser Client Technology

You can configure the enterprise server to execute browser client applications using one of the following client-side technologies:

- Microsoft .NET
- Microsoft JVM

You can configure the server to support either one or both technologies by choosing one of the available options:

- Java: Browser Client applications will be executed using Microsoft JVM only.
- .NET: Browser Client applications will be executed using the Microsoft .NET framework only.
- Java and .NET: Browser Client applications will be executed using either the Microsoft JVM or the Microsoft .NET framework. The application will first attempt to execute the application using the Microsoft JVM.
- .NET and JAVA: Browser Client applications will be executed using either the Microsoft JVM or the Microsoft .NET framework. The application will first attempt to execute the application using the Microsoft .NET framework.

If a client does not have the expected framework technology, the client will be unable to run the application and the Missing browser client technology error page will be displayed.

Change Effective: Next Session

Magic INI and Command Line Name: BrowserClientTechnology

Missing Browser Client Technology Error URL

In this setting, you can define the error page URL to be opened when a client does not have the framework technology specified by the Browser Client Technology environment setting.

Change Effective: Immediate

Magic INI and Command Line Name: BrowserClientTechnologyErr

Browser Client Network Error Recovery Timeout

it is recommended to set a recovery timeout when running a browser client application on a slow or disruptive network. By setting this timeout, the client tries to recover unprocessed requests within a designated timeout. The recovery mechanism is disabled when set to 0 seconds (default).

Change Effective: Next Context

Magic.ini and Command Line Name: BrowserClientRecovery

Browser Client Cached Path

The Browser Client Task Cache improves the performance of the browser client application by caching XML's logic segments of the page on the client-side, minimizing the data transmission from the server to client.

This setting lets you specify the physical path in which the engine creates the cached files.

Change effective: Immediate

MAGIC.INI and Command Line Name: CTLCacheFilePath

Browser Client Cached Alias

The browser task's result page requires a URL reference to the browser client cached file. You can specify the browser client cached alias from this environment setting.

Change effective: Immediate

MAGIC.INI and Command Line Name: CTLCacheFilesAlias

Foreground Generator Context Management

From this environment setting, you can set the foreground engine to handle requests **As background engine** or **Single common context**.

When you select **Single common context**, the engine handles all requests within the same single context. This mode of operation resembles a foreground runtime engine.

When you select **As background engine** the following engine behavior occurs:

When switching to Runtime mode, the Main Program's Task Prefix operations are not executed.

The Menu system displays limited built-in options for closing the application and switching back to Toolkit mode. The user-defined pulldown menu is not displayed.

Main Program events are not activated at idle time.

For every explicit request:

- the Main Program's Task Prefix operations are executed.
- a new context is created.
- When the request is completed or when exiting from the top-level browser task, the Main Program Task Suffix operations are executed and the context is cleared.

Opening a new context for every explicit request means that:

- All Main Program variables are reset.
- All memory tables are reset.
- All global values are reset.
- All environment settings are reset.

Change effective: Next session

MAGIC.INI and Command Line name: ForegroundContextManagement

eDeveloper Defaults

You can define NULL display strings and a default date value for the entire application in the 'MAGIC_DEFAULTS' section of the Magic.ini file. These settings can only be defined by editing the Magic.ini text file. They are not available in eDeveloper's Environment dialog box.

These settings will be regarded as the system default values for:

- The default value of a date field definition.
- The NULL display string for all other fields' attributes.

If these settings do not exist in the Magic.ini, eDeveloper uses its own pre-defined defaults.

The following are the entries in the 'MAGIC_DEFAULTS' section:

```
[MAGIC_DEFAULTS]
```

```
DefaultDate = [date value]
```

```
NullAlphaDisplay = [string value]
```

```
NullNumericDisplay = [string value]
```

```
NullLogicalDisplay = [string value]
```

```
NullDateDisplay = [string value]
```

```
NullTimeDisplay = [string value]
```

```
NullMemoDisplay = [string value]
```

```
NullBLOBDisplay = [string value]
```

Note: The DefaultDate value format should correspond to the format of the Magic Date setting of the *Environment/System* tab.

Advanced Toolkit Settings

The Magic.ini options below let the developer create user-defined utility applications that enhance the capabilities of eDeveloper's development environment.

JAVA Settings

The developer can load and activate Java classes without setting the operating system's environment variables. Java class settings can be designated for the eDeveloper environment.

The MAGIC_JAVA Section

The [MAGIC_JAVA] section is displayed in the Magic.ini file. The following Java settings can be entered:

- **JAVA_HOME** - This setting points to the Java Virtual Machine stored in the client's operating system. If JAVA_HOME is set to Yes, eDeveloper overrides the JAVA_HOME's operating system variables for the variables designated in the eDeveloper environment.
- **CLASSPATH** - This setting determines the path for locating additional classes stored for the Java Virtual Machine. If CLASSPATH = Yes, an eDeveloper prefix is added to an operating system's CLASSPATH setting.

Color Settings

The Color repository defines the identifying foreground and background colors for 14 entries that represent preset, user-defined, reserved color assignments, and operation repository colors.

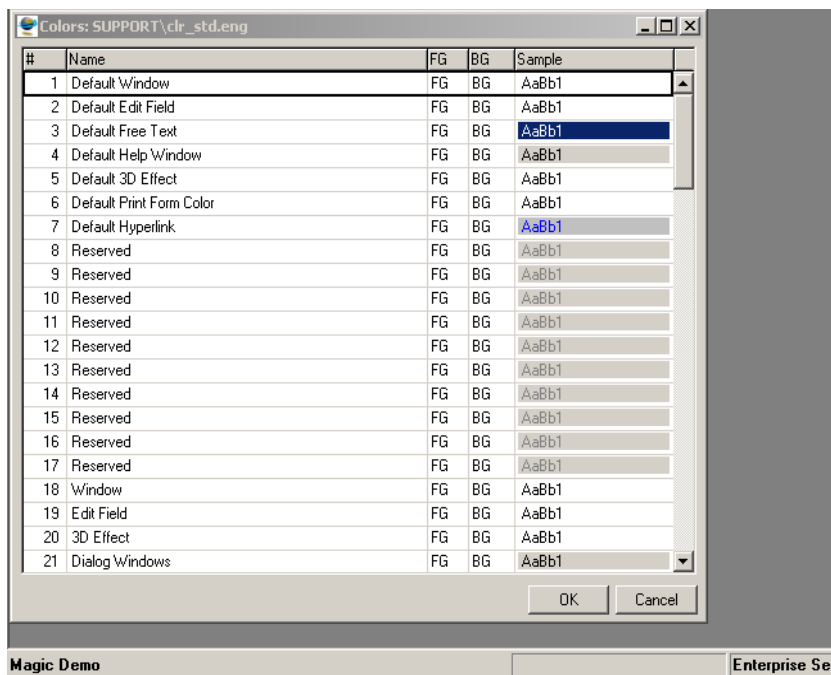


Figure 2-6 The Color Repository

The Color Repository Settings

#

This column contains an automatically-generated sequential number used by eDeveloper as a mapping identifier. You cannot edit this column.

Name

The Name column of the Color repository contains a textual description of the color's function. Use it to give meaningful names to colors for ease of selection and maintenance.

FG and BG

The Color repository, shown in Figure 2-6, assigns a foreground (FG) color and a background (BG) color value for each of the display entities, one per row.

You can enter a new color value after the last row. The Color repository stores an unlimited number of color values.

The Color Assignment Palette

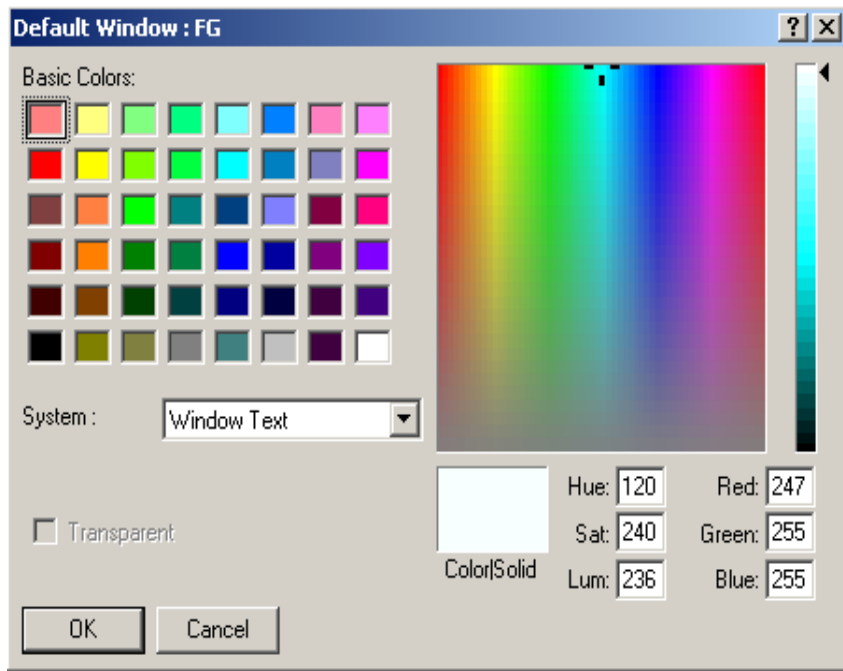


Figure 2-7 The Color Assignment Palette

To change the color of a foreground or background component, place the cursor on the FG or BG you want to change. Then zoom F5 to the Color Assignment palette. An example of a Color Assignment palette is shown in Figure 2-7. A color can be selected from the Basic Colors shown by clicking a color. The selected color will display a heavy border.

A color can also be selected or modified by changing the numeric values for Red, Green and Blue, and for Hue, Saturation, and Luminescence (Hue, Sat, Lum). The name value from the current row of the Color repository will appear on the Title Bar of the Color Assignment Palette window.

A System Color combo box appears on the Color Assignment Palette window. A system screen's logical background or foreground color definitions can be specified using this combo box.

You can define a color background as transparent by selecting the Transparent check box, which is enabled only when the System Color combo box is empty.

Clicking OK will accept the new color assignment and will close the Color Assignment Palette window.

Clicking CANCEL will undo all changes to the color assignments made in the current entry to the Color Assignment palette, and will close the Color Assignment Palette window.

Be careful not to choose the same values for both FG and BG of a display entity, because the resulting display will not be readable.

Sample

The Sample setting provides a visual representation of the selected foreground and background colors.

Saving Changes to the Color Repository

From the Color repository, click OK to accept the changes and to end the color editing session. The color settings you edit are saved in a special color file. The file eDeveloper uses is set in the Color Definition File setting in the Environment dialog. The default name for this file is CLR_STD.ENG. You can create and use various color files. When you end an editing session in the

Color repository, eDeveloper prompts you to save the changes with the Save File dialog, as shown in Figure 2-8.

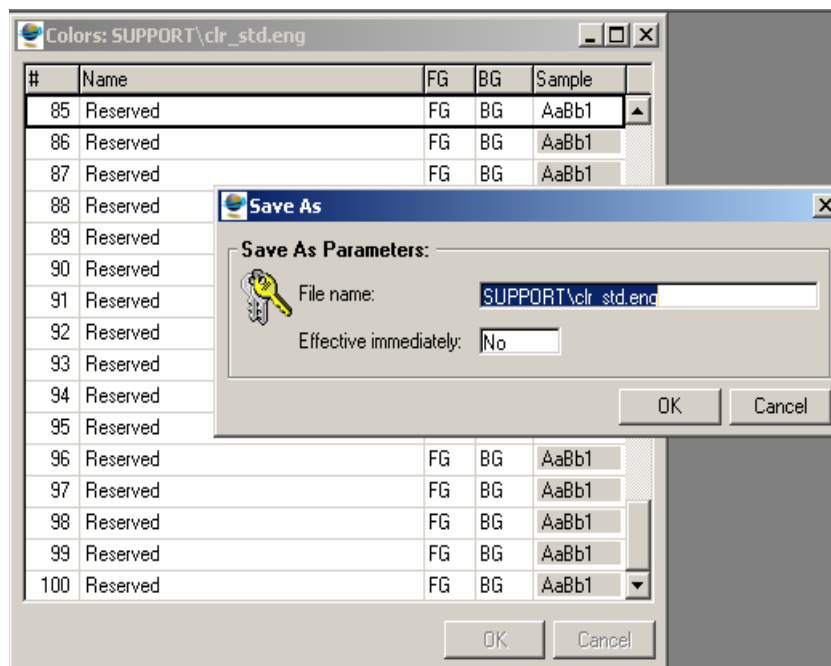


Figure 2-8 Saving Edited Colors

It is possible to save these changes to a file different from the one currently used by eDeveloper, by specifying a different name in the Save As field. The changes made to the color repository will take effect the next time you load eDeveloper from the operating system, unless Yes is specified in the Effective Immediately prompt of the Save File dialog.

If Effective Immediately is requested, eDeveloper will load the new color repository, and every new window painted on the display will use the new settings. However, the current display will not be not changed until it is refreshed in a proper context.

Within a particular application, it is possible to replace the default color file of the Environment with an application-specific file.

Note that if you change system colors and color schemes in your operating system, these changes are not reflected immediately in eDeveloper. These changes will take effect in the next activation of eDeveloper.

Font Settings

The Font repository, shown below, associates specific fonts to each kind of output available. The bulk of the entries in the Font repository are for user-defined font assignments. The Font repository can store an unlimited number of font values.

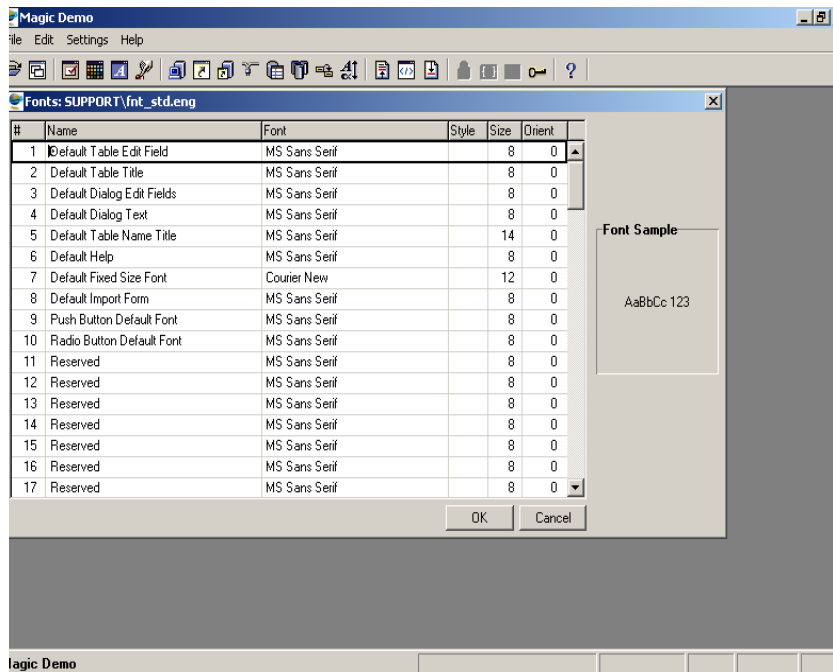


Figure 2-9 The Font Repository

The Font repository is stored in the Font Definition file specified in the Environment dialog, in the Font Definition File setting.

The Font Repository Settings

#

A line number assigned by eDeveloper. You cannot edit this column.

Name

The description of the kind of output. You can change this description by simply overwriting the name in the repository.

Font

The name of the font, such as MS Sans Serif or Helvetica. You can change the font by double-clicking on it. This will open a font assignment window as shown in the figure below. You can select a type face, a size, a font style such as Bold or Italic, and an effect such as Strikeout or Underline in this window.

Style

If the font style is either Bold or Italic, or both, and if Strikeout or Underscore effects are selected, these conditions are noted in the Style column.

Size

The typeface size is shown in the Size column.

Orientation

This column shows the angle at which the font displays.

Using fonts with display orientation is limited to the Tab control and the Text control only. Using such a font in another control may lead to unpredictable results. When applying a font to a Text control it is not possible to edit the text directly from the Form editor. The text must be edited from the Text Control's Control Properties dialog.

The Font Style Window

The font style from the current row of the Font repository appears in the Font Style window.

Font Assignment Window

The name value from the current row of the Font repository will appear in the caption of the Font Assignment window.

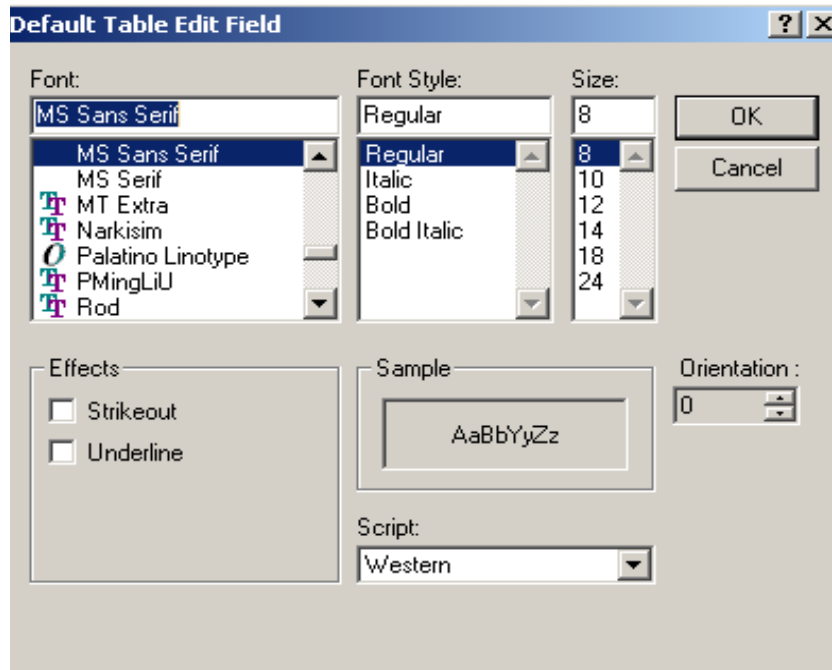


Figure 2-10 The Font Assignment Window

Click OK to accept the new font assignment, close the Font Assignment window, and return to the Font repository.

Click CANCEL to undo all changes to the font assignment made in the current entry, and close the Font Assignment window.

Saving Changes to the Font Repository

From the Font repository, click OK to accept the changes and to end the Fonts editing session. The font settings you edit are saved in a Font Definition file. The file eDeveloper uses is set in the Font Definition file setting in the

Environment dialog. You can create and use various font files. On conclusion of an editing session in the Font repository, eDeveloper prompts you to save the changes with the Save File dialog, as shown in Figure 2-11.

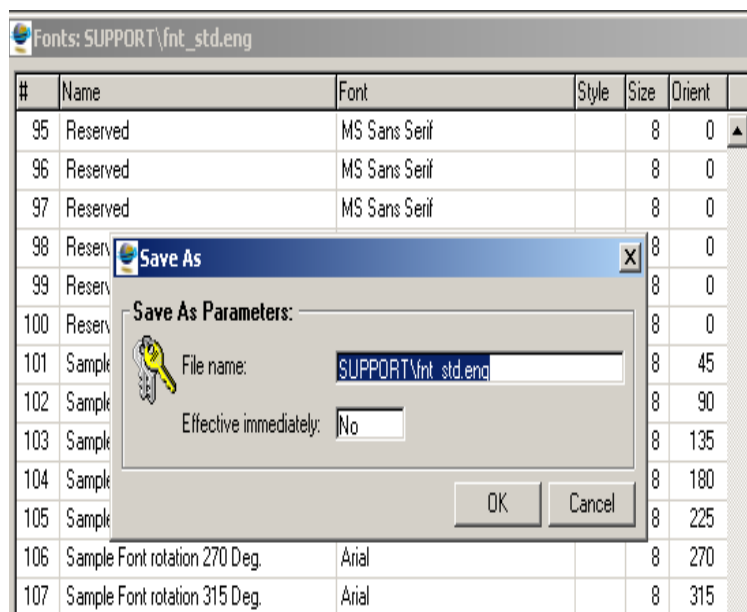


Figure 2-11 Saving Changes to the Font Definition File

Keyboard Mapping Settings

eDeveloper Actions

eDeveloper's internal commands are referred to as *eDeveloper Actions* or *Actions*. An eDeveloper Action is the lowest level command that eDeveloper can understand. Every action has a well-defined function, which means all user interaction with eDeveloper is translated into Actions that invoke some internal process. For example, the Exit Action will cause the current process to terminate, while the About Action will open the About dialog. Because many end-users interact with eDeveloper using the computer's keyboard, keystrokes must be able to trigger internal eDeveloper Actions to achieve results. The

Keyboard Mapping repository contains the assignments of keystrokes to eDeveloper Actions.

eDeveloper has a large set of internal actions, each of which performs a unique function. Some of these functions are global, and may be performed from anywhere in eDeveloper, while others are context-specific and can only be invoked in context. Because certain functions are context-specific, it is possible to assign the same keyboard sequence to more than one action, provided that the actions are not available in the same context.

State Qualifications to eDeveloper Actions

You can refine the keystroke-action relationship further by adding state qualifications to the assignment of a keystroke to an Action. Each assignment can have up to four state qualifications. The logical condition between the 4 keyboard state conditions is an AND condition. In other words, all the keyboard state conditions have to occur for the key to invoke the assigned action. These state qualifications are related to the two different editing contexts - the Form level, Record level, or the Control level. The indicators that display whether an action is for a form, record, or a control are:

- The *Tbl* prefix to a keyboard state condition name in the Keyboard States list refers to an action for the form or record.
- The *Edt* prefix to a keyboard state condition name in the Keyboard States list refers to an action for a control.

Two examples of different keyboard state conditions are:

- The **Tbl:Not Screen Top** keyboard state condition means that the key activates the action only if the insertion point is not placed at the top of the screen or table.
- The **Edt:Not Form Top** keyboard state condition means that the key activates the action only if the insertion point is not parked on the first value of a Choice control or the first line of a Multi-Edit control.

eDeveloper Action Example

While editing a repository, CTRL+PGDN is assigned to the End of Repository Action, which tells eDeveloper to move to the end of the last row of repository data. CTRL+PGDN is also assigned to the End of Screen Action, which tells eDeveloper to move to the last row of the repository displayed on the screen. Both actions are active in the same context, Repository Editing, and have the same keystroke assignment CTRL+PGDN. The ambiguity is resolved by qualifying the assignment of CTRL+PGDN using eDeveloper States. The End Repository Action is qualified by the Tbl:Screen End state, which means it will only be invoked when CTRL+PGDN is pressed and the insertion point is positioned at the last visible row of the repository. The End Screen Action is qualified by the Table Not Screen End State, which means it will only be invoked when CTRL+PGDN is pressed and the insertion point is not positioned at the last visible row of the repository. Because the two states are mutually exclusive, eDeveloper has no problem determining which action should be invoked. The result of these settings is that the first pressing of CTRL+PGDN will move the insertion point to the last visible row of the repository. The Table Screen End State is then turned on, and the second pressing of CTRL+PGDN will move the insertion point to the last row of repository data.

The Keyboard Mapping Repository

The Keyboard Mapping repository is made up of a main screen and the Keyboard States window. The screen title at the top of the screen contains the file name of the current Keyboard Mapping. Each entry in the repository represents one assignment of a keyboard value to an eDeveloper Action. While scrolling the repository lines, the display of the keyboard states changes to reflect the states conditioning the assignment of the highlighted line.

In this repository it is possible to change, delete or add action-keystroke assignments. It is not possible to delete the last keystroke assignment of an

action. When adding a new line to the repository, the action is copied from the line preceding it, without the key assignment.

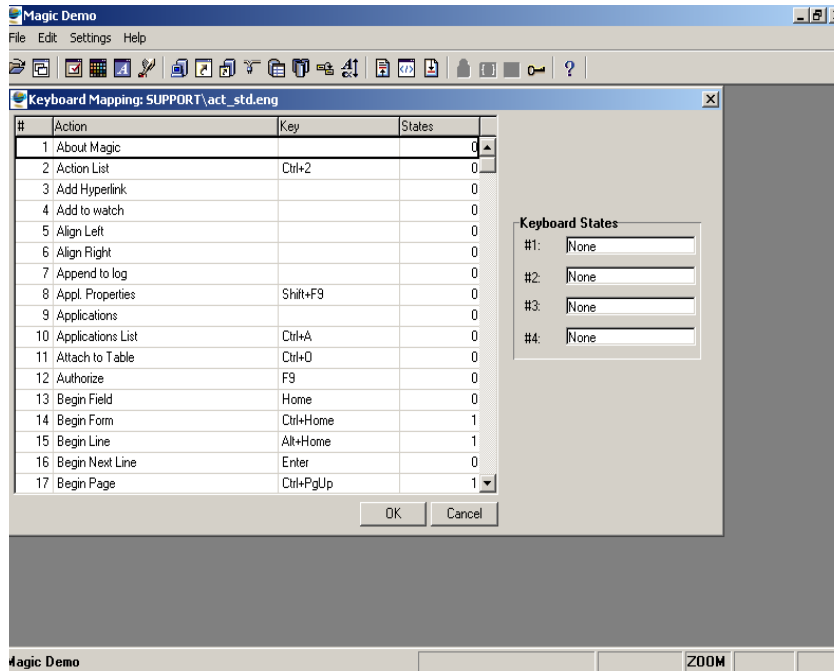


Figure 2-12 Keyboard Mapping Repository

The Keyboard Mapping Repository Settings

#

This column contains an automatically-generated sequential number used by eDeveloper as a Mapping identifier. You cannot edit this column.

Action

This column contains the eDeveloper Action name. You cannot edit this column. Action names are fixed and are provided by eDeveloper. The same Action may appear on more than one row, meaning the Action has multiple key assignments. Pressing any of the keys assigned to the Action will invoke it.

Key

The key column displays the key assigned to the action of the current row. From this column it is also possible to define the key. Select *Edit/Zoom* F5 to access the Key Definition dialog. In the Key Definition dialog, just press the key you want assigned to the action and it will register on the screen. Click OK to accept the definition, or ESC to cancel the operation. The space bar will clear the current value in the Key Definition dialog. If accepted, the key will be copied to the Key column and assigned to the key.

NOTE: If the word Internal is displayed in the Key column, then no operations are allowed on this Action. Internal Actions may not be redefined.

States

The States column displays the number of states conditioning the assignments. All states have to be satisfied to invoke the Action when the Key is pressed. From the States column select *Edit/Zoom* or press F5 to zoom into

the Keyboard States window, as shown below. In this window, define the conditions for the assignment of the Key to the Action.

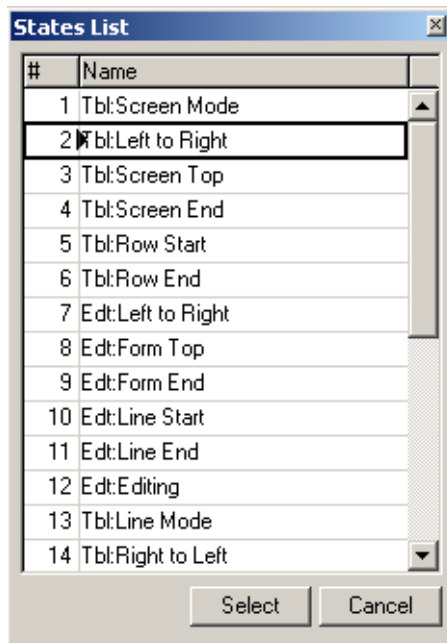


Figure 2-13 The Keyboard States List

State 1 to State 4

While the insertion point is positioned at one of the state settings, select *Edit/Zoom F5* for a list of states. It is possible to add up to four states to condition the Key assignment to the Action. All the states have to be on at once for the Key to invoke an action.

The States list constitute 24 states that can help you condition the key assignment to the action. The **Tbl** prefix means that the state refers to the Form or Record level. The **Edt** prefix means that the state refers to editing inside a control.

The Keyboard Mapping settings you edit are kept in a special file. The eDeveloper file is named in the Keyboard Mapping File setting in the Environment dialog. By default this file is ACT_STD.ENG. You can create and use different mapping files. On conclusion of an editing session on the

Keyboard Mapping repository, eDeveloper prompts you to save the changes with the Save File dialog. You can save these changes to a file different from the one currently used by eDeveloper, by specifying a different name in the Save As dialog. The changes made to the Keyboard Mapping repository will only take effect the next time you load eDeveloper from the operating system, unless Yes is specified in the Effective Immediately prompt of the Save File dialog.

If Effective Immediately is requested, eDeveloper will load the new Keyboard Mapping repository and change the Key assignments accordingly.

Servers Settings

eDeveloper Servers are the basis for the Client/Server capabilities built into eDeveloper. The eDeveloper server is a File Management and Server Type program, capable of performing Database and File I/O for remote eDeveloper applications. Any eDeveloper installation can access many such servers, residing on various hardware platforms and operating systems, via different communication protocols. From the Server repository you can define the remote eDeveloper servers accessible for database and file I/O services to the current installation. The Server repository also defines the means by which the local eDeveloper software communicates with its host server.

The Server repository, Figure 2-14, defines the remote eDeveloper servers accessible for database and file I/O services to the current installation. The Server repository also defines the means by which the local eDeveloper software communicates with the host server.



For more information, refer to Chapter 19, Distributed Application Architecture.

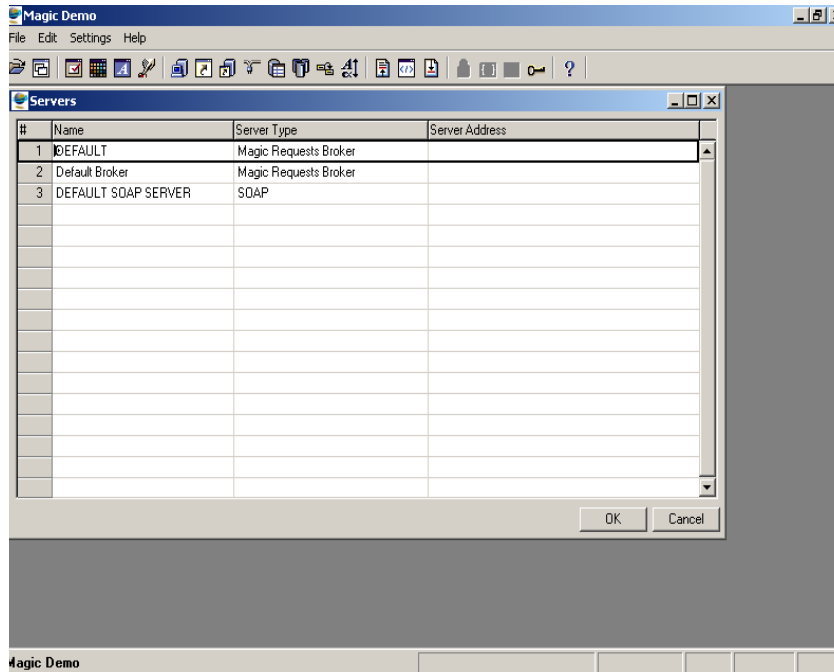


Figure 2-14 The Server Repository

The Server Repository Settings

Name

The Name field defines the server name for the local eDeveloper. The Server name, when used within parentheses in any file name, will cause automatic redirection of any input or output request for the file to the Server.

Server Type

Server Type identifies the Server types available to transfer and receive data from the host Server. You can update this setting only by selecting Edit/Zoom (F5) from a list of available servers: Magic Request Broker, and MQSeries. The Server Types in the list are only those loaded before eDeveloper itself was loaded. The local eDeveloper and the Server must use the same

communication protocol. If a previously defined server has a type that is not recognized, the server type setting is "Unknown". Trying to open a server without first loading its proper communication driver will fail.

The available server types are:

Magic Request Broker: The Magic Request Broker (MRB) implements a Requester-Broker connection. The MRB appears in the Server Type list only if the MRB is defined in the appropriate Magic.ini section, and communication support is installed.

MQSeries Server: The MQSeries server implements an MQSeries Messaging server. The MQSeries appears in the Server Type list only if the MQSeries Messaging gateway is defined in the [MAGIC_MESSAGING_GATEWAYS] section of the MGREQ.INI file, loaded into memory, and the appropriate communication support is installed.

SOAP Server: The SOAP server lets you send a SOAP type Web service request to a Web Service Provider.

Server Address

In the Server Address column, specify the Server's address on the network connecting it to the local eDeveloper installation. The Address is made up of two parts, separated by a / symbol.

The Server repository must contain at least one entry. This entry is named DEFAULT and cannot be deleted. Other rows can be added or deleted, and all of the settings except # can be edited. The DEFAULT server is the "local" server of the local installation. No communication driver or address is necessary.

Server repository information is contained in the [MAGIC_SERVERS] section of the Magic.ini file.

Properties

The optional Properties dialog for the current line in the Server repository is accessed by selecting Properties from the Server's context menu.

The Properties dialog contains the following settings:

- User Name
- Password
- Timeout
- Alternate Server

The Timeout setting defines the duration of time, in seconds, of no response from the server before the driver reports a failure to eDeveloper.

You may optionally enter, in the Alternate Server setting, the name of an alternative server that eDeveloper can use if the selected server is unavailable.

Communication Manager

The Communication Manager setting identifies the Communication protocol used to transfer and receive data from the host Server. You can update this setting only by selecting *Edit/Zoom* F5 for a list of available Communication drivers. The Communication drivers in the list are only those loaded before eDeveloper itself was loaded. The local eDeveloper and the Server must use the same communication protocol.

If the Communication Manager entry shows Unknown, then the proper communication driver module was not loaded. Trying to open a Server without first loading its proper communication driver will fail.

Services Settings

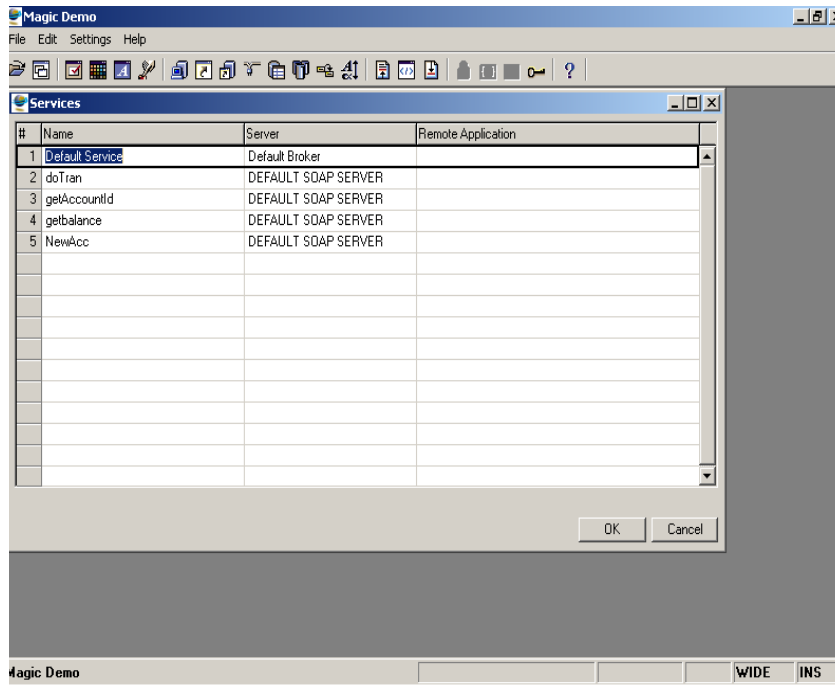


Figure 2-15 The Service Repository

The Service Repository Settings

#

This column contains an automatically-generated sequential number used by eDeveloper as a mapping identifier. You cannot edit this column.

Name

The unique name of the service displayed.

Server

The Server entry from the Servers list. Zoom from this field to select the appropriate server.

Remote Application

The name of the eDeveloper application. Zoom from this field to select an application from the Application list.

When you select a SOAP server for a Web Service request, enter the SOAP request identifier from the Web Service Provider's Web Service Description Language (WSDL).

The Services Properties Dialog

Access the Services Properties dialog by selecting Properties from the Services context menu.

The Service Properties dialog contains the following settings:

User Name

The user name that will be used to perform the eDeveloper logon, when a request is sent for a particular eDeveloper application.

Password

The password associated with the user name when a request is sent for a particular eDeveloper application.

Filter

In the Magic.ini file, the filter values are displayed in the [MAGIC_SERVICES] section, as shown below:

Default Service = Default Broker, <application>, <username>,
<password>, <filter>

The filter value is sent to the broker with each request.

Refer to MRB Filters on page 1167 for a full explanation about filters.

Visual Connection Settings

The Visual Connection display is a virtual representation of the connection between a particular eDeveloper Service and the list of servers available. You

can also modify server connections by changing the lines displayed in this visual connection.

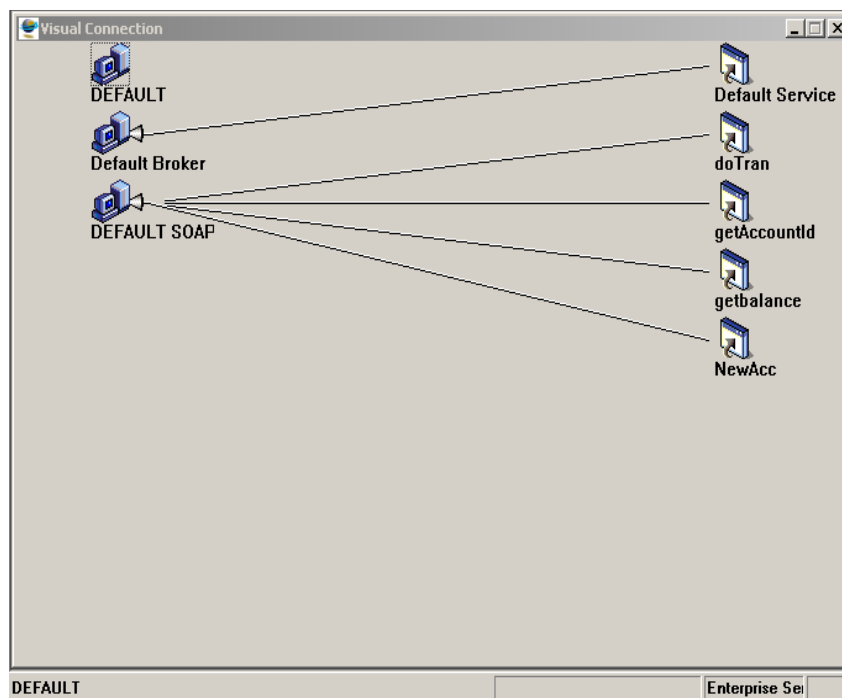


Figure 2-16 The Visual Connection Display

Communication Settings

The screenshot shows a window titled "Magic Demo" with a menu bar (File, Edit, Settings, Help) and a toolbar. A "Communications" dialog box is open, displaying a table of network protocols. The table has five columns: "#", "Name", "Timeout", "Parameters", and "ID". The data is as follows:

#	Name	Timeout	Parameters	ID
1	NONE	0	NO Parameters needed	1
2	DECnet	0	DECnet Parameters	3
3	IPX	20	IPX Parameters	4
4	NetBios	20	NetBios Parameters	5
5	TCP/IP	30	1500-2000	2

The dialog box has "OK" and "Cancel" buttons at the bottom right.

Figure 2-17 Communication Repository

The Communication repository is generated automatically by eDeveloper at its initialization stage. The Communication repository lists all the communication drivers loaded prior to eDeveloper. These drivers will also appear in the Communication list selection window. The communication drivers are used to connect the local eDeveloper to host eDeveloper Servers.

The Communication Repository Settings

#

This column contains an automatically generated sequential number used by eDeveloper as a Communications Driver identifier. You cannot edit this column.

Name

The Communications Driver name. This setting is provided by eDeveloper according to the communication drivers supported by eDeveloper.

Timeout

The Timeout setting defines the duration of time, in seconds, of no response from the receiving end before the driver reports a communications failure to eDeveloper.

Parameters

The Parameters column contains customization information that is transferred to the Communications driver at runtime.

Communication repository information is contained in the [MAGIC_COMMS] section of the Magic.ini file.

ID

The ID field should not be modified. The ID setting lists the internal code used by eDeveloper for identifying the communication protocol.

DBMS Settings

The DBMS repository lists all the DBMSs that are supported by eDeveloper or will be supported by eDeveloper in the future. Information included in the DBMS repository as opposed to the Database repository, described in the following section, is generic to the DBMS type and not specific to the database based on that DBMS. An eDeveloper database gateway product must be loaded prior to loading eDeveloper in order to provide the physical access to any database of any of the specific DBMS types. Database gateways provide an interface for eDeveloper with its underlying Database Management Systems. For example, in order to access Microsoft SQL Server or Oracle

databases, eDeveloper needs the eDeveloper database gateways for Microsoft and Oracle to be loaded before eDeveloper is loaded.

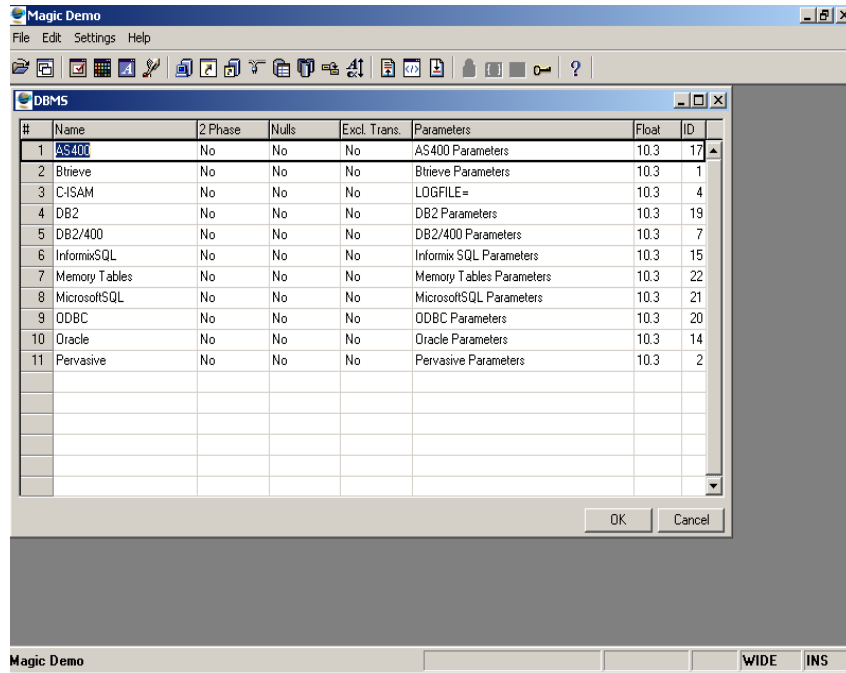


Figure 2-18 The DBMS Repository

eDeveloper supports connectivity to the following databases:

- AS/400 ISAM and SQL interface
- Pervasive.SQL 2000 ISAM engine
- Cache
- DB2
- Informix
- Microsoft SQL Server
- Oracle

- eDeveloper also supports connectivity to various databases by using ODBC (Open Database Connectivity)

The eDeveloper installation provides the following bundled databases:

- MSDE – The Microsoft® Database Engine.

Pervasive.SQL™ 2000:

1. The Pervasive.SQL 2000 Workgroup engine (relational and transactional) is bundled with eDeveloper's development environment. This product is intended solely for development purposes and is restricted to usage under this environment.
2. The Pervasive.SQL 2000 server engine (relational and transactional) is bundled with eDeveloper's deployment environment. This engine has a license that expires after three months.

To obtain a permanent license either:

- call TOLL FREE 0080012123434 (For BE, DK, NL, SE, UK, DE, CH, FR, IT)
- call +32/70/233761
- or send an email to cic@pervasive.com

The DBMS Repository Settings

#

This column contains an automatically-generated sequential number used by eDeveloper as a DBMS identifier. You cannot edit this column.

Name

The Name column contains the name of the DBMS. The name can be modified.

2 Phase

Not applicable. This option is no longer supported.

Nulls

If the underlying DBMS supports NULL values and the eDeveloper applications are to attach to tables with NULL values, set this field to Yes. Use the No setting if the DBMS does not support NULL values, or if the DBMS supports NULL values, but that support is not required in eDeveloper.

This setting controls the default behavior of the eDeveloper toolkit when creating new columns or field models. When this property is set to Yes, all columns or field models are created with Allow Null=Yes. If the property is set to No, then all settings or models are created with Allow Null=No. If this setting is set to Yes, the gateway creates nullable columns when issued in the CREATE TABLE statement. It is advisable to set the Nulls setting to No and change the setting in the Column properties sheet when a nullable column is required.

Excl Trans

Not applicable. This option is no longer supported.

Parameters

The Parameters column contains customization information that is transferred to the RDBMS at runtime. Refer to Chapter 25, SQL Considerations for information on customizing the database gateway. Also see the section below on Variable MCF record length.

Float (Default Float Picture)

When the Table Definition is uploaded from the DBMS's repository (when using the Get Definition utility), most of the settings can be matched into eDeveloper attributes with proper pictures. However, in the case of floating point types of settings, it is not possible to tell what the picture of the resulting eDeveloper setting or type should be. The Default Float Picture setting provides a default picture template that will be used for all floating point type setting definitions uploaded into eDeveloper Model and Table repositories. A sample value is 10.3.

ID

The ID setting should not be modified. The ID setting lists the internal code used by eDeveloper for identifying the DBMS.

DBMS repository information is contained in the [MAGIC_DBMS] section of the Magic.ini file.

DBMS Properties

The DBMS Properties dialog box provides access to additional settings required by some database entries and not required by others. The settings in this dialog do not change frequently

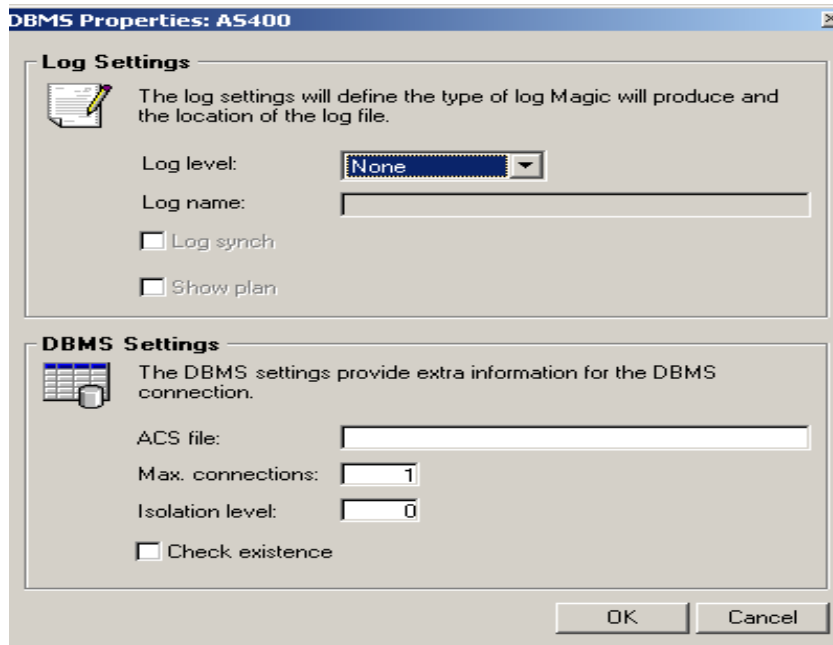


Figure 2-19 DBMS Properties

Log Level

The Log Level is the level of the log to be generated by the Magic Database Gateway for your RDBMS. The possible values are:

- None - No log file will be generated
- Customer - Log only the SQL commands generated
- Support - Additional information for the developer
- Developer - A full log to be generated for use by the MSE Technical Support department.

Note: The Log Level property affects performance. Therefore, it is recommended to use the Log Level property only during debugging.

It is advisable to verify that the Log Level property is disabled (= None) in your customer's environment.

Log Name

The Log Name is the name of the log file, including its path. If a path is not specified, the log file is created in the eDeveloper directory.

Log Synch

The Log synch property controls the synchronization between the Log file and the application execution.

'Selected' - When this check box is selected, the log is synchronized with the application execution.

'Not selected' - When this check box is not selected, there is a delay between the application execution and the log.

If eDeveloper is terminated improperly you may not see the last application execution in the log.

Note: When the Log Synch check box is not selected, the Log is saved into a buffer and written to the Log file every several records.

Note: This property is mainly useful when debugging your application. This is because you want to see all of the operations, especially the last ones, when eDeveloper has terminated improperly.

Show Plan

Not applicable. This option is no longer supported.

Alternate Collating Sequence

Zoom from the Alternate Collating Sequence (ACS) setting to an Open File dialog. Select the ACS file. The file name appears in the ACS properties setting.



For more information, refer to Chapter 25, SQL Considerations.

Maximum Connections

Some RDBMSs such as MS-SQL use different numbers of connections in their internal implementations.

Setting the Max. connections property controls the maximum number of connections the gateway can use.

The default value is 3 connections.

The Max. connections property applies to each line in the Database repository.

The greater the number of connections the gateway uses, the better the client concurrency

However, at the same time, RDBMS resource requirements increase while server performance decreases.

Specify 0 to use the default number of connections.

Specify a number greater than zero to limit the number of connections to the databases.

In ISAM and SQL/400 DBMS, you must specify the number of allowed connections to AS/400. This number depends on the number of AS/400 licenses.

Isolation Level

Isolation levels determine the type of phenomena that can occur during the execution of concurrent transactions.

eDeveloper sets this property only for the following databases: MSSQL, Informix and DB2.

Three phenomena define SQL Isolation Levels for a transaction:

Dirty Reads returns different results within a single transaction when an SQL operation an uncommitted or modified record created by another transaction. Dirty Reads increase concurrency, but reduces consistency.

Non-Repeatable Reads returns different results within a single transaction when an SQL operation reads the same row in a table twice. Non-Repeatable Reads can occur when another transaction modifies and commits a change to the row between transaction reads. Non-repeatable reads increase consistency, but reduces concurrency.

Phantoms returns different results within a single transaction when an SQL operation retrieves a range of data values twice. Phantoms can occur if another transaction inserted a new record and committed the insertion between executions of the range retrieval. Each Isolation level differs in the phenomena it allows:

Phenomenal/ Isolation Level	Dirty Reads	Read Committed	Repeatable Read	Can be Serialized
Dirty Read Allowed	Yes	No	No	No
Non- Repeatable Read Allowed	Yes	Yes	No	No
Phantoms Allowed	Yes	Yes	Yes	No

Read Committed: Specifies that shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the

transaction, resulting in nonrepeatable reads or phantom data. This option is the SQL Server default.

Can Be Serialized: Places a range lock on the data set, preventing other users from updating or inserting rows into the data set until the transaction is complete. This is the most restrictive of the four isolation levels. Because concurrency is lower, use this option only when necessary. This option has the same effect as setting `HOLDLOCK` on all tables in all `SELECT` statements in a transaction

For more information about Isolation Levels, see Chapter 23, Multi-User Considerations.

Check Existence

This property determines if eDeveloper will check the existence of every table it tries to access in runtime, and if the table does not exist, eDeveloper will create it.

Clearing this check box will enhance performance and will prevent eDeveloper from creating tables in the database. If a specific table does not exist, an error will be issued by the underlying RDBMS.

This option is recommended for the Runtime environment.

Selecting this check box will decrease performance, since eDeveloper will have to send extra `SELECT` statements to the DMBS each time the table is accessed. If the table does not exist, eDeveloper will attempt to create the table in the database.

This option is recommended for the Toolkit environment

Note: This property will be copied to any new database entry that belongs to the specific DBMS in the Database repository.



For more information, refer to Chapter 25, SQL Considerations.

Variable MCF Record Length

All components of an eDeveloper application reside in an eDeveloper Application file. Each component is stored in a logical record in the file. A component can be a task, help screens, global types, and so on.

The physical structure of the eDeveloper application file consists of two parts. The first is the index, in order to access a certain component, and the second is the actual data on the size of the component. The size of the data portion is predetermined by default, with different values for each DBMS.

Why Change the MCF Record Length?

Invoking an eDeveloper task involves reading several logical records. Therefore we should attempt to reduce the elapsed time for reading data and to load the task faster. Because MCF records are a fixed length, if our tasks are small and do not include many occurrences of the components of a task, we might be reading more data than we actually need to construct the eDeveloper component. On the other hand, if we have large tasks with many occurrences of eDeveloper components, we might be doing too many I/O operations to load the task.

Defining the MCF Record Length

To override the default record length values, a new qualifier, in the setting section in the DBMS repository can be used. The usage is MCFRECLen=x, where x is a number. The default will only apply to new MCF files that are opened, and will not affect existing MCF files.

The valid sizes for each DBMS are shown in the next table. If an invalid size is given, an error message will be displayed and the eDeveloper application file will be created in the default length.

A change is effective immediately.

Valid Sizes for MCFRECLen (Data Portion)			
DBMS	Default	Minimum	Maximum
Btrieve	1007	330	4076

Valid Sizes for MCFRECLen (Data Portion)				
DBMS	Default	Minimum	Maximum	
C-ISAM	1010	330	4082	
DB2	214	214	214	(Fixed Size)
Microsoft SQL	269	269	269	(Fixed Size)
Oracle	269	269	269	(Fixed Size)
Microsoft	269	269	269	(Fixed Size)

Note: The total MCF record length consists of the MCF record length (data portion) plus 14 bytes (index).

Effects of Changing the MCF Record Length

After creating the MCF file, no additional definitions are needed, and eDeveloper will be able to open all MCF files, regardless of their record lengths.

Changing the record length can also affect the size of the MCF. Because the MCF record is a fixed length record, there is unused space in the file.

If the record length is enlarged the physical MCF file will grow, and there will be more unused space.

Database Settings

The Database repository registers details about all the physical databases that can be accessed by this installation of eDeveloper, as shown below. eDeveloper is able to connect to multiple Database Management Systems (DBMSs). Some of these DBMSs provide support for the distribution of data on different physical locations on one machine, or even across several different machines. Some of the DBMSs that eDeveloper supports maintain several separate

If a set of tables in the database are accessed by one user and another set of tables are accessed by a second user, then two identical definitions can be defined by using a different user in each one.

i

Once the connection to a database is created, it is only disconnected explicitly (using the eDeveloper DbDiscnt function or by the DBA) or by exiting eDeveloper (exiting the application will not close the connection). Therefore, the changes made in the Database properties will take effect only in the next connection to the database. It is recommended to exit eDeveloper and launch it again whenever changing a database property that affects the eDeveloper Gateway behavior.

The Database Repository Settings

#

This column contains an automatically-generated sequential number. You cannot edit this column.

Name

The Name setting is a short description of the physical database. The Name value is used as a label string that identifies the database anywhere in eDeveloper where such a reference is required. For example, in the Table or Application repository. By using a label instead of an index into the database repository, eDeveloper makes the application more portable, because it is unlikely that different developers will use the same label. The Name value is case-sensitive.

The default databases available are:

Default - The default database used when creating a new application or a new table within an application. This entry usually points to Pervasive DBMS, but you can changed the entry to point to any supported DBMS system.

Memory - This database points to a memory DB that is implement by eDeveloper. This gateway is available as long as the engine is up. When eDeveloper is closed, the memory database stops and all its data is cleared.

DBMS

The DBMS setting identifies the name and type of the underlying database. This column can be modified only by selecting a value from the DBMS list (zoom to the list).

When creating a new line, some of the properties are inherited from the DBMS repository (properties that appear in the DBMS repository).

The DBMS setting is required.

Database Name

The Database Name setting identifies the database that you want to connect to in the RDBMS.

RDBMSs can support and handle more than one database.

Some of the DBMSs that eDeveloper supports maintain several separate sub-databases in the same RDBMS.

The Database Name is the name of the sub-database as defined by the RDBMS.

In ODBC, the Database Name of the defined Data Source in the ODBC administrator.

The Database Name setting is optional.

Magic Server

The Magic Server setting specifies the eDeveloper Server to be used to get to the data. eDeveloper supports two methods for accessing remote data: the eDeveloper Client/Server model or the DBMS's Client/Server model.

eDeveloper's Client/Server Model: Using this method, an eDeveloper Client/Server setup should be implemented. eDeveloper will access the remote data via the eDeveloper server on the host machine. Only an eDeveloper Definition Database gateway is required for such access.

DBMS's Client/Server Model: If the DBMS provides Client/Server services, eDeveloper can be used to take advantage of them. In this case, the database appears to eDeveloper as though it were a local database, and it is accessed through a full eDeveloper Database gateway. When the database is on the same computer, or when the DBMS Client/Server architecture is used, do not specify the eDeveloper server parameter.

The Magic Server setting is used only if the data resides on a remote machine, and if the access to that machine is gained via an eDeveloper Client/Server model.

The Magic Server setting is optional.

Location

The Location setting is optional and relevant for ISAM databases only.

The Location setting provides eDeveloper with information about the physical location of the data file.

Files that are created by eDeveloper will be placed in the location specified in this property.

A Logical Name may be used for this setting.

The Location setting is optional.

The Database Properties Dialog

The Database Properties dialog provides access to additional settings required by some databases entries but not required by others. The settings in this dialog do not change frequently.

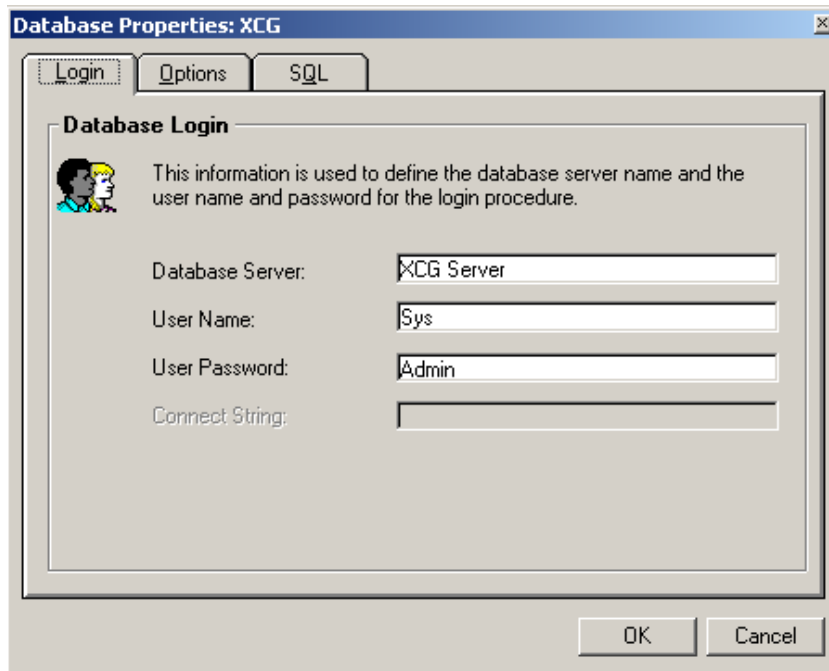


Figure 2-21 Database Properties Dialog

The Database Properties dialog tabs are:

- Login
- Options
- SQL

Login

The Login database properties are described below.

Database Server

The Database Server setting specifies the name or the IP address of the RDBMS server.

When you use DB2 and ODBC databases, there is no need to specify the Database Server property, because the DB2 and ODBC databases have a DNS (specified in the Database Name column in the Database repository).

When you using Oracle, you specify an Alias.

The Magic Server setting in the Database repository and this setting should not both be used for the same database entry.

The Database Server setting is optional.

User Name

Certain databases require a User Name and Password. This is true for databases accessed by an eDeveloper server or by the DBMS's own Database Server. When you use the eDeveloper Client/Server Model for accessing remote host machines (usually mini-computers or workstations with a strict security system built into their operating systems) a User Name and Password are mandatory. If you leave the User Name setting blank in the Database Properties dialog, eDeveloper will open a dialog requiring a User Name and User Password on the first attempt to attach to a table accessed using this Database entry.

Use this setting to specify a User Name recognized by the host security system, where the table accessed, by this database entry, resides. A Secret Name can be used for this setting, because the User Name setting is kept in the unprotected Magic.ini file.

The User Name setting is optional.

User Password

Use this setting to specify a User Password recognized by the host security system where the table accessed by this database entry resides. A secret name can be used for this setting, because the User Name setting is kept in the unprotected Magic.ini file. The User Password setting is optional.

Connect String

The Connect String is only relevant for Oracle. Writing a string overrides the database server definition, user name, and password.

Options

The Options database properties are described below.

Change Tables in Toolkit

This setting determines whether an eDeveloper application in Toolkit mode can alter the table structure of an underlying database table.

The default for this setting depends on the type of database selected in the Default Database setting of the Environment dialog. If the Default Database is an SQL-type database, then the default for the Change Tables in Toolkit setting will be a 'not selected' check box. The default may be set to a 'selected' check box when building the table structure, *before* data is entered, and before the application goes into production.

Note: Once data has been entered, and if the default is 'selected', eDeveloper will drop the original table from the database, and re-create the table with the new modifications. eDeveloper advises against making the default 'selected' once the application is in use. The 'selected' default in SQL-type databases, unlike in ISAM-type databases, affects the underlying rules for the table structure.

If the Default Database is an ISAM-type database, then the default for the Change Tables in Toolkit setting will be 'selected'.

Check Definition (required)

eDeveloper maintains its own Table repository, while providing access to data stored in various DBMSs that maintain their own data dictionaries (all of the DBMSs manage their own data dictionaries). The data structure may be modified both by eDeveloper (such modification is reflected automatically in the DBMS's data dictionary), and by external DBMS utilities (unknown to eDeveloper). As a result, eDeveloper's Table repository might not reflect the current structure of the data that eDeveloper programs process. This may lead

to abnormal behavior in eDeveloper programs. Not selecting the Check Definition setting instructs eDeveloper to disable the file structure checking when the file opens. Selecting the Check Definition setting, instructs eDeveloper to check the data structure of the physical file against eDeveloper's Table repository definitions every time a data table is opened. If the physical file structure and the eDeveloper Table repository definitions do not match, an error message will be displayed by eDeveloper and processing will be aborted. This feature is available for those DBMSs that can provide table structure information. The extent of such information is also dependent on the DBMS type.

Note: The method used by previous versions of eDeveloper for this feature, specifying 'CHKDEF=YES' in the Database Information setting, is still supported. However it is recommended that you use this new setting instead.

Check Index (required)

This property provides a useful method of maintaining data integrity. When Check Index is enabled, eDeveloper tests the data table for the existence of a unique index value as the end user inputs data into a record. If a duplicate value is entered for an index that is defined as unique, eDeveloper issues a Duplicate Index Error message and prevents further end-user processing until the input is changed. This behavior requires no programming on the part of the developer. However, for some DBMS systems, this check takes a toll in performance and it is not recommended. In such a case, the developer may decide to disable this check, and then the results of duplicate index value entry will depend on the DBMS behavior.

Server Sort

This setting is available with AS 400 databases.

This setting indicates to eDeveloper whether or not to perform the task's Sort operation using the virtual indexes defined in eDeveloper to open a Query Field. If the Server Sort setting is selected, eDeveloper will not perform its own sort but will add a virtual index to the current table and use it as the current cursor.

eDeveloper Locking (None)

This property is mainly relevant for ISAM databases.

eDeveloper manages its own locking mechanism at the row and table levels without transactions by using the MGLOCK file. These locks are referred to as eDeveloper locks and can be enabled by setting the eDeveloper Locking property to Record Lock or Table Lock. Because SQL RDBMS locking is always invoked, there is usually no need for an additional locking mechanism when accessing an SQL database in a physical or deferred transaction mode task.

Setting the eDeveloper Locking property to None implies that only the underlying database's concurrence controls will be in effect.

Lock Path

When using a Database server to access data, eDeveloper is not aware of the physical location of this data: the data may reside on some remote host machine, but it still appears to eDeveloper as though it were local. In this case, the internal eDeveloper locking mechanism cannot operate, because this mechanism requires all users accessing the same data to share the same locking file.

Previously, table locking was determined only by the program's Access and Share mode settings in the Table repository. In Online programs with multi-user access, the eDeveloper program, depending on the Access and Share mode settings, caused the table to be locked for the duration of the task. This situation can now be controlled in SQL databases, to allow the DBMS to handle table locking and thereby reduce locking duration.

The default value for this setting is Yes. When Table Lock = No, eDeveloper issues no table level locks, and the table's Share mode is determined by the type of transaction issued by the DBMS. It is useful to set Table Lock = No when you want protected Write transactions to be issued by the underlying DBMS.

If you set the eDeveloper Record Lock setting to Yes, eDeveloper's internal locking mechanism will be used in addition to the Database's mechanism. In this case, eDeveloper must have access to a disk directory that is shared by all

the users of this database entry. Such a directory can be specified using this Lock Path setting.

Setting the Database Lock Path in the Database Properties dialog in *Settings/Databases* will override all other definitions.

If a database has a lock path defined in its properties, all locks regarding this database will be performed on the lock file found in that path. Lock files will not be opened in the directories in which the database files are opened. If you define a database lock path, be sure the path is shared by all the users who access the tables. Otherwise, you risk data corruption.

A Logical Name may be used for the Lock File Path setting.

The Lock Path setting is optional.

Common Data Dictionary

On some of the older platforms that eDeveloper supports, there are central database information repositories. These common data dictionaries allow different application programs to access the same data, and always remain synchronized regarding the data structure.

Use the Common Data Dictionary setting to specify the name of a Common Data Dictionary that will be referred to by eDeveloper.

In modern DBMSs this setting is not frequently used.

A Logical Name may be used for the Common Data Dictionary setting.

The Common Data Dictionary setting is optional.

SQL

The SQL database properties are described below.

Database Information

The Database Information setting lets you supply database-dependent information for eDeveloper to pass to the underlying RDBMS. For more information, refer to Chapter 25, SQL Considerations.

Hint

Some RDBMSs such as Oracle and MSSQL, allow hinting the optimizer for processing a query. In this field, the programmer can enter a string that will be concatenated to the SELECT statement.



For more information, refer to Chapter 25, SQL Considerations.

Check Existence

This setting specifies if eDeveloper is to check the existence of every SQL table it attempts to access in deployment (runtime) mode, and to create that table if it does not exist.

Selecting this check box enables eDeveloper to create tables in an SQL database. This also directs eDeveloper to check for the existence of every table it attempts to access. Note that checking for the existence of each table may cause a performance degradation.

Not selecting this check box prevents eDeveloper from creating tables in the database. If a specific table does not exist, an error is issued from the underlying RDBMS.

Array Size

The eDeveloper gateways to SQL support array processing. When retrieving rows from the database, the gateway does not retrieve one row at a time, but retrieves a group of rows, which reduces network traffic.

Although eDeveloper uses its own array size of rows, these numbers can be revised. When scanning a large table, increasing the array size can enhance performance. It is recommended, however, to use the eDeveloper default. The default should be changed only in special cases.

The array size is copied to the table property only when you create a new table.

XA Transactions

Not applicable. This option is no longer supported.

Alternate Collating Sequence

Zoom from the Alternate Collating Sequence (ACS) setting to an Open File dialog. Select the ACS file. The file name appears in the ACS properties setting. Note that if no ACS file is specified in the ACS setting, eDeveloper will look for an ACS string in Database Information. If no string is found, then eDeveloper will access the ACS file defined in the Alternate Collating Sequence File setting of the Environment dialog.

The ACS file that appears in this setting reflects the ACS file set for the DBMS. When a new DBMS is selected for a specific database, the ACS file changes.

Once the connection to a database is created, it is only disconnected explicitly or by exiting eDeveloper (only exiting the application leaves the connection open). Therefore, the changes made in the database properties will take effect only in

the next connection to the database. It is recommended to exit and re-launch eDeveloper whenever change database properties that affect eDeveloper gateway behavior.



For more information, refer to Chapter 25, SQL Considerations.

Logical Names Settings

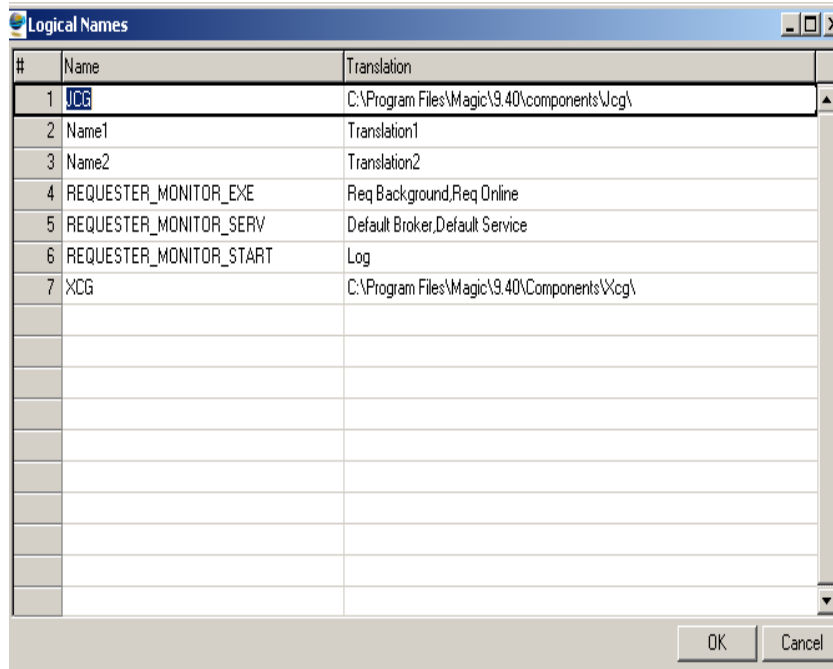


Figure 2-22 The Logical Name Repository

eDeveloper's Logical Names feature is a valuable facility for writing portable applications. The Logical Names facility allows development

of applications without any explicit relation to physical storage media or operating system naming conventions. eDeveloper achieves such portability by translating application Logical Names at runtime, according to the Logical Name repository used as the translation table of the installation. A Logical Name can be used whenever a file name or path is to be used.

The Logical Name Repository Settings

#

This column contains an automatically generated sequential number used by eDeveloper as a Logical Name identifier. You cannot edit this column.

Name

The Name to be used in the application. This name will be replaced with the value entered in the Translation column at runtime. The Name is constant for all installations of the same application.

Translation

The Translation column contains the actual string that should replace the Logical Name during file access.

Logical Names Usage

At runtime, whenever eDeveloper compiles a file name in order to perform some I/O process, the Logical Names algorithm is executed. File names are scanned, from left to right, and any logical name is substituted with the Translation from the Logical Name repository. Multiple Logical Names may be used in one file name (for example, one for disk drive, a second for directory, and a third for the file name).

Logical Names Syntax (in use from Version 5 and higher)

The syntax for Logical Names is:

%logicalname%

Where:

% means Start Logical Name.

logicalname is the Logical Name.

% means End Logical Name.

Rules for using Logical Names:

1. The '%' symbol is not allowed in any file name for purposes other than delimiting logical names.
2. During translation, the logical name and its delimiters are replaced by their Translation according to the Logical Name repository. If the Logical Name is not found in the repository, it is cleared from the file name.
3. If the closing delimiter is missing, all the text from the first delimiter to the end of text is treated as the Logical Name.

Language Settings

The Language repository contains details of the translation files used by eDeveloper's Multi-lingual support feature.

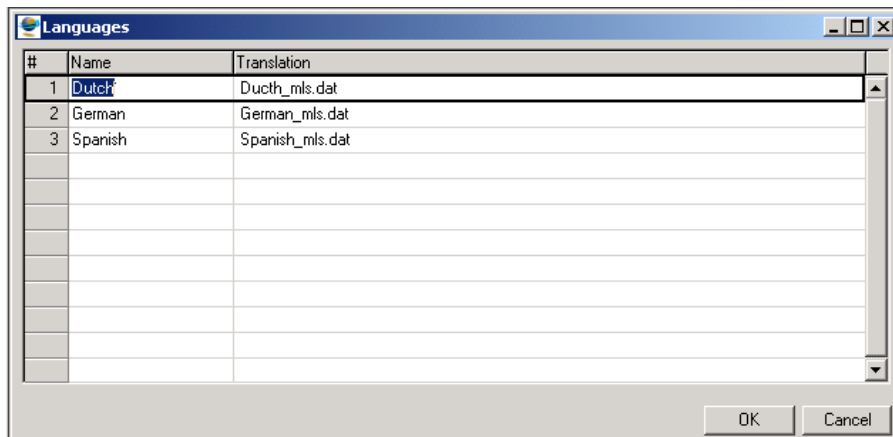


Figure 2-23 The Language Repository

Multi-lingual Support (MLS)

eDeveloper's multi-lingual support allows developers to develop an eDeveloper application in one language and then deploy it in other languages.

The following restrictions currently apply to this feature:

- MLS is available only in the Windows version of eDeveloper. eDeveloper versions with a text-based user interface do not support this feature.
- MLS works only for left-to-right languages.
- MLS does not support multi-line edits and rich text controls.

The Language Repository Properties

#

This column contains an automatically generated sequential number used by eDeveloper as a language identifier. You cannot edit this column.

Language

The Language Name column contains the name of the target language for the translation. This is a case-sensitive column for up to 128 text characters.

Translation File

This column contains the full path and name of a language translation file to be used when a language is selected. Zoom from this column to open a Windows Open File dialog.

A Starting Language setting appears in eDeveloper's Environment repository.

Creating a Language Translation File

The command line utility MLS_BLD does not check for the existence of a destination file, and will always overwrite an existing file with the same name.

Using MLS at Runtime

You can specify a language for deployment by making an entry in the [MAGIC_LANGUAGE] section of the Magic.ini file.

If you specify a deployment language, the contents of the following texts will be translated automatically at runtime using the Language Translation file:

- Controls text property
- Menu entry text
- Prompt help text
- Verify expression operation text
- Window titles text
- Field names text
- Index names text
- Index segment names text
- I/O output file names text
- Choice controls (radio button, combo box, list, tab control) selection value text

The Font property of each control can be defined as an expression that is evaluated at runtime to select the appropriate language translation file. The actual data value used to set the active selection will remain as defined in the application, regardless of the displayed translation string.

During toolkit all operations that check the size of text in controls (Fit size, APG on variable), create the control according to the original size of the text and not according to the translated size of the text.

When using translation text for choice controls, make sure the number of options in the translated text is exactly the same as the number of options in the original text. Failure to do so may cause unpredictable results.

For related functions, see GetLang and SetLang in the Functions in Chapter 8, Expression Rules.

The Printer Repository Properties

#

This column contains an automatically-generated sequential number. You cannot edit this column.

Name

The Logical Printer's name. This is the name that will appear in the Printer list selection window. The printer name is the data string registered in the eDeveloper program as the printer identifier. The printer name specified here is searched for at runtime when the eDeveloper program tries to resolve the logical printer name to get the actual printer data.

Queue

The Queue property specifies the physical printer that will receive all the output directed to the current logical printer. You can specify either a local or a remote printer. The Queue property is a string value. eDeveloper parses this string and extracts valid information. If no valid printer information is found the logical printer will default to the operating system's default printer (such as LPT1 for the Windows operating system).

Commands File

The Commands File property specifies the location and operating system name of the file that contains the printer control codes that will be used to resolve the logical eDeveloper print attributes. The Commands file contains both labels and printer control codes. The eDeveloper print attributes are built using these labels. When an eDeveloper form is printed, the logical print attributes are translated according to the information found in the printer's Commands file. Commands files of different printers may contain the same labels. The same program may output different results when the Commands file is different.

Translation File

The Translation File property specifies the location and operating system name of a file containing a conversion table for converting eDeveloper's internal character codes to physical printer codes. The translation file may have 256

entries, one per ASCII character code, which is the internal eDeveloper character set. Each entry specifies the string substitute of the character that will be sent to the printer. The printer strings may contain single or multiple character codes. You may also include control codes in these strings.

Lines

The Lines property specifies the page size eDeveloper uses while printing to the printer. This page size does not have to be identical to the physical printer's page size. Usually the Lines Property specifies a value less than the number of lines per page of the physical printer. The Lines property value is used for eDeveloper's internal accounting while eDeveloper keeps track of printer output. After sending a number of lines equal to the value of Lines, eDeveloper will send a form feed signal to the printer. The Lines property is used as the default for the current printer and may be overridden by the Rows property in the particular I/O file used for an Output Form operation. Refer to the I/O Files section of the Output Form discussion in Chapter 7, Operations.

Printer repository information is contained in the [MAGIC_PRINTERS] section of the Magic.ini file.

HTML Style Settings

The HTML Style repository defines the HTML Style tags available to associate with HTML forms generated in the HTML Form Editor.

The HTML Style Properties

Name

A unique name representing an HTML style.

HTML Tag

An HTML tag or tags making up the HTML style, which is merged with an HTML tag generated by eDeveloper.

An example would be:

BORDER=0 to eliminate the border when displaying hyperlink objects.

Print Attribute Settings

The Print Attribute repository is explained in the Chapter 10, Output Forms.

Secret Name Settings

Secret names are intended for use where there is a need to hide Authorization system implementation information from unauthorized users. For example, Secret Names should be used for Application file access keys, User password fields, Servers/DB properties and Data file access keys.

The Secret Name repository is basically identical to the Logical Name repository, with the following exceptions:

- Access to the Secret Name repository is allowed to the Supervisor only.
- Secret names are stored in the security file that is specified as an Environment setting. This security file is encrypted. The secret name for a server password is stored in the server settings.

A Secret Name can be used wherever a Logical Name can be used, using the same syntax. Refer to the *Settings/Logical Names* section of this chapter for an explanation of the association of Logical Names to Secret Names. eDeveloper will attempt to resolve a logical name through the Secret Name repository first, before looking for it in the Logical Name repository.

Connecting to an LDAP Server

eDeveloper uses the \$USER\$ and \$PASS\$ secret names in the LDAP Connection String environment setting to bind to the Lightweight Directory Access Protocol (LDAP) server. The \$USER\$ and \$PASS\$ tags are replaced with the user's name and password.

If the user enters eDeveloper without providing a user name and password, the LDAP_USER and LDAP_PASS secret names are used to bind to the LDAP server. If the LDAP_USER and LDAP_PASS are not defined, the LDAP server binds to eDeveloper as an anonymous user. Since the user has logged onto the operation system domain, user authentication is not an issue.

User Groups Settings

The User Groups repository is where the supervisor declares groups and defines their rights.

User ID Settings

The User ID repository lists the User's ID, name, password, rights, and groups in which the user is associated. Access the User ID Properties dialog to include additional user information. For more information about User IDs, refer to Chapter 13, Authorization System.

Logon Settings

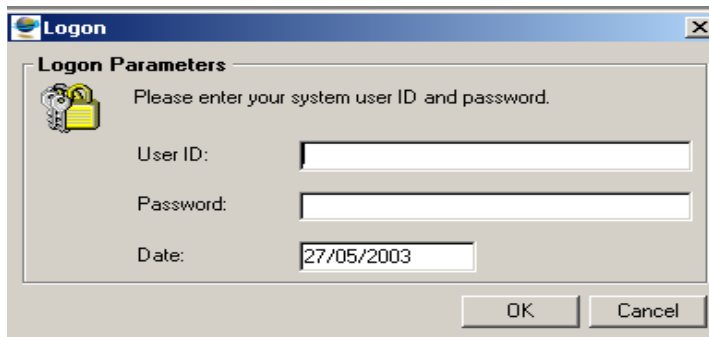


Figure 2-25 The Logon Dialog

The Logon dialog allows the entry of a User ID and Password of the eDeveloper user. These values are checked against the User ID repository to determine

the user's rights. If the User ID and Password do not match an entry in the User ID repository, an error message is displayed and the dialog stays open. If the User ID and Password combination are found in the User ID repository, they are used to compile the rights of the user for later use within an application. User ID and Password have both toolkit and runtime functions. The Logon also provides access to the Magic Date property. The user may change the Magic Date property for the specific session. The Logon dialog is controlled by the following Environment options:

- Input Password
- Input Date
- Allow Access to Logon

These options are described in the Environment dialog section.

The Logon Properties

User ID

The name of the user. Any alphanumeric input is allowed. The User ID must also be in the User ID repository. The Logon dialog will display the User ID prompt if the Input Password property is set to Yes in the Environment dialog. The user identification cannot be more than 30 characters.

Password

The Password to match the User ID. Any alphanumeric input is allowed. The User ID-Password pair must be defined in the User ID repository. The Logon dialog will display the User ID-Password prompt if the Input Password property is set to Yes in the Environment dialog. The user password cannot be more than 30 characters.

The User ID-Password check is case-insensitive. If the User ID-Password prompt is displayed, the Logon dialog will remain open until both items are entered correctly.

Date

An alternative date to the System Date. This date may be queried by the MDate function. Refer to Chapter 8, Expression Rules. Also see Magic Date in the Environment dialog.

The Logon dialog will display the Date prompt if the Input Date property is set to Yes in the Environment dialog.

System Logon Setting

The System Logon setting is located at Settings/Environment/System. This setting determines the level of the eDeveloper environment integration with the operating system. The valid values for this setting are:

- **None** – The Logon dialog may be opened either automatically, according to the Input Password setting, or explicitly, by activating the Logon menu option without using any default values as the user name.
- **User Name** – The Logon dialog may be opened either automatically, according to the Input Password setting, or explicitly, by activating the Logon menu option, and the default value for the Logon name will be the current user who is logged on to the operating system.

Note: The user password in the eDeveloper user file does not have to correspond to the operating system password.

The Magic.ini File

The Magic.ini file contains all of Magic's variable configuration information. The Magic.ini file must be present in the Magic working directory for Magic to start up. Magic provides various facilities for editing the Magic.ini properties, in the form of the *Settings* menu entries. Refer to the preceding sections in this chapter for information about these facilities. The Magic.ini is a free format text file that can be edited using any external text editor. eDeveloper also provides the INIPut and INIGet functions for access to the Magic.ini file from

within an application. For more information on the INIPut, INIGet functions, refer to Chapter 8, Expression Rules.

The Magic.ini file is divided into sections. Each section contains a group of related properties. Some Magic.ini file sections are used by eDeveloper and some contain user information. The Magic sections of the Magic.ini file are:

- Environment - [MAGIC_ENV] - Global environment properties.
- Systems - [MAGIC_SYSTEMS] - Applications for the current installation.
- Servers - [MAGIC_SERVERS] - The list of all the remote Servers available for the current installation.
- Services - [MAGIC_SERVICES] - The list of eDeveloper Applications available per server.
- Communications - [MAGIC_COMMS] - The list of eDeveloper communications drivers.
- DBMS - [MAGIC_DBMS] - The list of all the DBMSs supported by eDeveloper.
- Databases - [MAGIC_DATABASES] - The list of all the Databases supported by Magic.
- Logical Names - [MAGIC_LOGICAL_NAMES] - The Logical Name repository.
- Printers - [MAGIC_PRINTERS] - Printer information.
- Gateways - [MAGIC_GATEWAYS] - The list of eDeveloper database gateways.
- Languages - [MAGIC_LANGUAGE] - The list of supported languages.

The User sections of the Magic.ini file can contain any information that you see fit to store in it. Typical uses could be:

- Storage of intermediate values produced by the application, such as last positions in different files and programs, or last customer processed, and so on, without the overhead of using database tables.
- Storage of global variables; values that may be shared by different

programs within the application.

- Specific end-user applications, such as General Ledger - [GENERAL_LEDGER].
- The Magic.ini file contains both installation-specific information, and user sections with user-specific information. Therefore, in multi-user installations of eDeveloper, each user should have a separate copy of a Magic.ini file in a separate directory.
- There are no concurrency controls when accessing the Magic.ini file. If in a Multi-user installation the Magic.ini file is shared by several users, it should be used for read-only purposes. An attempt to write to a shared Magic.ini file by more than one user at the same time will result in one user overwriting the updates of the other.

The Magic.ini File Format

The Magic.ini is a free-format text file. Each setting value starts on a separate line. Setting values may span more than one line. Settings are built from a fixed identifier (name) and a variable part (value). Empty lines and remark lines may be included. The format for the file is:

```
[SectionName]
SettingName = SettingValue
.
.
.
SettingName = SettingValue
[SectionName]
.
.
.
;This is a remark line
```

Where:

[SectionName]

Sections are preceded by a section header line.

Section headers are placed within square brackets. Section headers should carry unique meaningful names. Blanks are not allowed in a section header name. The section header is used when accessing properties of the section. Use long names in order to eliminate possible conflicts between sections written by different applications. A section ends at the next section header or at the end of the file.

SettingName

All the properties found after a section header belong to that section. Every setting value occupies one or more lines. The SettingName is the fixed part of the Magic.ini setting. It is used to identify and access the variable part of the setting. Setting names must be unique within a section but not across sections. Blanks are not allowed in a setting name.

The setting separator. The equal sign is the separator between the property name and the property value. Every character in front of the separator (excluding leading and trailing blanks) is part of the property identifier. Every character after the separator (excluding leading and trailing blanks) is part of the setting's value.

SettingName=SettingValue

Any value is allowed for the setting. Setting values are determined by eDeveloper for the Magic sections and by the developer for the application sections. To span the parameter value over more than one line, append the + sign to the end of the line. eDeveloper will treat the next line as a continuation of the previous one. To explicitly specify the + symbol as part of a property, use the \+ combination.

; Remark lines

The ; symbol at the beginning of a line will cause eDeveloper to ignore whatever follows to the end of line.

Saving Server Information to the Magic.ini File

Server information is saved to the Magic.ini file in the following format:

<server name>=<communication type number>, <server address>,
<username>, <password>, <timeout>, <alternate server name>, <server
type number>

Notes:

- The server type number is a unique number assigned to a server type. The translation between a server type number and a server type is hard-coded in eDeveloper.
- The eDeveloper server type number is 1. The server type number is 1 for the Magic Request Broker. For a data server, the server type number is 0.

Command Line Options

Using Command Line options, you can set a session-specific configuration of the eDeveloper installation. eDeveloper accepts all the configurable properties from the command line using the same identifiers as used by the Magic.ini file. Values received from the command line override those written in the Magic.ini file.

Specifying Command Line Options

The syntax for eDeveloper command line properties is:

MG**xxx** /Property1=Value1 /Property2=Value2 ...

Where:

xxx - eDeveloper executable file extension:

- GENW for Windows toolkit
- RNTW for Windows for runtime

/ - The / symbol specifies the beginning of a new property.

Property - The property identifier, as specified in the Magic.ini file. The Property name may contain a file section name. If a section name is specified, then the property is localized to that section. If no section name is specified, the [MAGIC_ENV] section is assumed by default.

= - The separator between property identifier and its value. No spaces are allowed between the property identifier, separator, and value.

Value - The value to be assigned to the property. Specify any of the valid values. Refer to the sections for information about properties and valid values.

The Operating System command line may be too short to hold all the necessary options. DOS, for example, allows only up to 128 characters in the Command Line. eDeveloper, therefore will accept a file that contains all the Command Line options.

When you want to include a comma (,), a backslash (\), a plus sign (+), or an equal sign (=) as a literal part of your command line text, it should be prefixed by a \ character. An example that includes commas is shown in the next section. Alternatively, you can use a pair of single quote marks (') preceding and following the value to instruct eDeveloper to accept the value as is. This way, you can include a slash as a literal character. For example:

```
/CommandProcessor = '%comspec%'
```

You can include an asterisk (*) in the command line property value to instruct eDeveloper to take the entire value until the end, ignoring terminators and quotes. This way you can build a clipboard, using the following expression:

```
INIPut ('clipboard=*' & VarCurr (VarInp (1)))
```

This function will work on any alpha value, regardless of its contents. If you want to use this clipboard concept for attributes other than alpha (such as numeric, date, ...), you should use the VarAttr function and translate the VarCurr function to string (using Str(), DStr()...) before concatenating. An example of this can be seen in The eDeveloper Demo application.

The syntax for an eDeveloper Command Line file is:

MGxxx @filename

Where:

xxx - eDeveloper executable file extension:

- GENW for Windows development,
- RNTW for Windows runtime

@ - The @ symbol instructs eDeveloper to treat the remainder of the actual command line as a Command Line file.

filename - The location and operating system name of a Command Line options file. The contents of the file should follow the rules set above for command line settings. Different options may be on different lines in the file.

Command Line Options Examples

Following are some command line examples for Windows systems.

1. MGGENW /StartApplication=1 /StartProgram=6

This command starts the Development module of eDeveloper and automatically opens the first application in the Application list, executing the Task Prefix level of the Main Program and the 6th program of that application.

eDeveloper assumes that the settings belong to the [MAGIC_ENV] section, because no section is specified in the identifier part of the property.

2. MGGENW /[MAGIC_LOGICAL_NAMES]Drive=C:

Will set the Logical Name Drive to have the Translation C:. If the Logical Name Drive is not found in the [MAGIC_Logical_NAMES] section of the Magic.ini file, a new temporary Logical name of these attributes will be created for the duration of the eDeveloper session.

3. MGGENW @mycmdl

Will instruct eDeveloper to look for its Command Line Properties in a file named mycmdl located in the current directory.

4. /StartApplication=1

/[MAGIC_LOGICAL_NAMES]Drive=C:

This file combines the command line settings shown in the first two examples above.

5. MGENW

```
/[MAGIC_PRINTERS]Printer1=LaserPrinter\,lpt1:\,lp.atr\, lp.eng\,66
```

Will override the settings of the first printer in the Printer repository with the setting specified in the command line.

Application Launch via the OS Command Line

You can launch eDeveloper with an application physical filename or with an application prefix as a command line property.

To use this feature, on the OS command line add the property /MCF= followed by the application prefix or by the full path name of the application file (MCF).

If the property value passed to eDeveloper contains only two characters, eDeveloper searches the prefix column first for an exact match. Note that the search is case insensitive.

When the property value is longer than two characters, eDeveloper searches the MCF name column in the Application repository first.

If an application matching the property is not found, one is created using the default database.

After the application is either found or created, it is run as any eDeveloper application.

Command Line Options and Magic.ini Values

Command Line options may be used to override any of the eDeveloper configurable properties. The Command Line values are essentially temporary and are only

in effect for the duration of the session. Sessions subsequent to a session that was started with Command Line options will revert back to the Magic.ini values.

Command Line values are kept in memory and are not written to the Magic.ini file, unless one of the Settings menu options is used to edit the eDeveloper configuration values. When editing Magic.ini values and Command Line values are also in effect, the resulting repository will reflect the combination of the two. The actual repositories will contain the Magic.ini file values for all the properties that were not specified in the Command Line. The Command Line values will complete the rest of the property values. Note that in this case the resulting repositories will not reflect all the actual Magic.ini values.

If any of the property values in one of the actual repositories is modified, the section containing the property values will be written to the Magic.ini file with the Command Line values. The original Magic.ini value will be overwritten. If no modification is performed while in one of the repositories, the changes will not be written into the Magic.ini file.

Environment Properties and Command Line Values

Environment Property Name	Command Line Name	Values
<i>System</i>		
Owner Name	Owner	string
System Logon	SystemLogin	string
Magic Date	Date	system date
User's ID	None	string
Input Password	InputPassword	Y, N
Input Date	InputDate	Y, N
Default Application	StartApplication	numeric value
Application Startup Mode	ApplicationStartup	T, R, B
Screen Mode Prompt	ScreenModePrompt	string
Century Start	Century	numeric value
Batch Event Interval	BatchPolling	numeric value
Task Cache Size	TaskCacheSize	numeric value
Allow Create in Modify Mode	AllowCreateInModify	Y, N
Allow Update in Query Mode	AllowUpdateInQuery	Y, N
Query Mode Locate Warning	LocateModeQueryWarning	Y, N
Allow Access to Applications	AccessApplications	Y, N
Allow Access to Environment	AccessEnvironment	Y, N
Allow Access to Colors	AccessColors	Y, N
Allow Access to Fonts	AccessFonts	Y, N
Allow Access to KBD Mapping	AccessKeyboardMapping	Y, N
Allow Access to Servers	AccessServers	Y, N
Allow Access to Services	AccessServices	Y, N
Allow Access to Visual Connections	AccessVisualConnection	Y, N

Environment Property Name	Command Line Name	Values
Allow Access to Communications	AccessCommunications	Y, N
Allow Access to DBMS	AccessDBMS	Y, N
Allow Access to Databases	AccessDatabases	Y, N
Allow Access to HTML Styles	AccessHTMLStyles	Y, N
Allow Access to Languages	AccessLanguages	Y, N
Allow Access to Logical Names	AccessLogicalNames	Y, N
Allow Access to Logon	AccessLogon	Y, N
Allow Access to Printers	AccessPrinters	Y, N
Allow Access to Print Attributes	AccessPrintAttributes	Y, N
Allow Access to Toolkit	AccessToolkit	Y, N
Allow Access to Checker Messages	AccessCheckerMessages	Y, N
Allow Testing Environment	AllowTesting	Y, N
Temporary Tables Path	TempPath	string
Maximum File Handles	FileHandles	numeric value
License	LicenseName	none
License File	LicenseFile	none
Load Monitor	Load Monitor	Y,N
Monitor Output File	Monitor2File	none
Remote Flow Monitor Port	RemoteFlowPortNumber	numeric value
Remote Flow Monitor	RemoteFlowMonitor	Y,N
<i>Multi-User</i>		
Terminal	Terminal	numeric value
Multi User Access	MultiUser	Y, N
ISAM Transactions	ISAMTransanctions	Y, N

Environment Property Name	Command Line Name	Values
Deadlock Prevention	DeadlockPrevent	Y,N
Server Communication Interval	ServerTimeout	numeric value
Lock File	LockFile	string
ISAM Force Locking Within Transaction	LockWithinTra	Y/N
Resource Lock File	ResourceLockFilePath	string
<i>Preferences</i>		
Default Database	DefaultDatabase	string
Database for Sort/Temporary	TempDatabase	string
Range/Locate Box Popup Seconds	RangePopTime	numeric value
Sort/Temp Box Popup Seconds	TempPopTime	numeric value
Keyboard Idle Seconds	IdleTime	numeric value
Pulldown Menu Close Timeout	MenuCloseTimeout	numeric value
Confirm When Auto-Exiting	ConfirmAutoExit	Y, N
Task Flow Modification	FlowModify	F, S
Display Copyright Messages	CopyrightMessages	Y, N
Deployment Custom Copyright	RTUserCopyright	string
Resident MAGIC.INI	ResidentINI	Y, N
Display Toolbar	RtToolBarGUI	Y, N
Resident Load on Program Init	ResidentLoadOnInit	Y, N
Load Resident Tables	LoadResidentTables	Y, N
Display Full Messages	DisplayFullMsgs	Y, N

Environment Property Name	Command Line Name	Values
Center Screen in Online	CenterScreenInOnline	Y, N
Reposition After Modify	RepositionAfterModify	Y, N
Indent Characters	IndentCharacters	numeric value
Default Color	DefaultColor	numeric value
Default Font	DefaultFont	numeric value
Tooltip Timeout	TooltipTimeout	numeric value
Maximum Number of Bookmarks	Bookmarksnumber	numeric value
Maximum Number of X-refs	MaxCrfResults	numeric value
Retry Operation Time Interval	RetryOperationTime	numeric value
IO Device Open Time	IOTiming	numeric value
Floating Palettes Always On Top	PalettesAlwaysOnTop	Y,N
Dockable Palettes	DockablePalettes	Y,N
Single Expand Palettes	SingleExpandPalettes	Y,N
Property Sheet Automatic Handling	AutomaticPropertyHandling	Open, Close
Image Cache Size	ImageCacheSize	numeric value
Check Image Change Time	ImageCacheCheckTime	numeric value
Toolkit Checker Minimal Level	CheckerLevel	Error, Warning, Recommendation
Group Checker Messages by	CheckerGroups	Object, Type, Object and Type
Jump automatically to first item in checker list	CheckerJumpAuto	Y,N

Environment Property Name	Command Line Name	Values
<i>International</i>		
Date Mode	DateMode	A, E, S, B
Thousands Separator	ThousandSeparator	one character
Decimal Separator	DecimalSeparator	one character
Date Separator	DateSeparator	one character
Time Separator	TimeSeparator	one character
<i>External Files</i>		
Logo File	LogoFile	string
Const File	ConstFile	string
Help File	HelpFile	string
Color Definition File	ColorDefinitionFile	string
Font Definition File	FontDefinitionFile	string
Keyboard Mapping File	KeyboardMappingFile	string
Documentation Template File	DocumentTemplateFile	string
HTML Styles File	HTMLStyles	string
Print Attributes File	PrintAttr	string
Security File	UsersPath	string
Startup Security File	StartupUsersFile	string
OEM2ANSI Tranlation File	OEM2ANSIFile	string
Browser Task ANSI to Unicode Translation	Unicode2Ansi	string
Alternate Collating Seq File	CollatingFile	string
Starting Language	StartingLanguage	none
Checker Messages table file	CheckMessageTable	string
European Currency Conversion File	EuropeanCurrencyConversionFile	string
Drop Data Supported User Formats	DropUserFormats	string

Environment Property Name	Command Line Name	Values
Command Processor	CommandProcessor	string
HTTP Proxy - Address Port	HTTPProxyAddress	string
HTTP Timeout	HTTPTimeout	numeric value
Print Data HTML Template	PrintDataHtmlTemplate	string
Print Data XML Template	PrintDataXmlTemplate	string
WSDL Files Path	WsdlFilePath	path string
Mail Connection Timeout	MailConnectionTimeout	numeric value
Mail Operation Timeout	MailOperationTimeout	numeric value
SNMP database connections utilization threshold	DatabaseConnectionsUtilizationThreshold	numeric value
LDAP Address	LdapAddress	string
LDAP Connection String	LdapConnectionString	string
LDAP Domain Context	LdapDomainContext	string
LDAP Timeout	LdapTimeout	numeric value
SSL CA Certificate Files	SSLCACertificateFile	string
SSL Client Certificate File	SSLClientCertificateFile	string
SSL Client Certificate Password	SSLClientCertificatePassword	string
<i>Server</i>		
Activate as Enterprise Server	ActivateRequestsServer	Y, N
Messaging Server	MessagingServer	Y, N
Requester Timeout	RequesterTimeout	numeric value
HTTP Requester	InternetDispatcherPath	URL address
Web Document Path	WebDocumentPath	path location
Web Document Alias	WebDocumentAlias	string
Maximum Number of Concurrent Requests	MaxConcurrentRequests	numeric value

Environment Property Name	Command Line Name	Values
Enterprise Server Can Change Application	RequestsServerCanRepla ceCtl	Y, N
Load Balancing Priority	LoadingBalancingPriority	numeric value
Web Authoring Tool	AuthoringToolPath	path string
Context Inactivity Timeout	ContextInactivityTimeout	numeric value
Post Context Unload Timeout	ContextUnloadTimeout	numeric value
Persistent Browser Client Module	UseSignedBrowserClient	Y, N
Browser Client Sub-Version	BrowserClientSubVersion	string
Browser Client Technology	BrowserClientTechnology	Java, .Net, Java or .Net, .Net or Java
Missing Browser Client Technology Error URL	BrowserClientTechnologE rr	path string
Browser Client Cached Path	CTLCacheFilesPath	string
Browser Client Cached Alias	CTLCacheFilesAlias	string
Foreground Generator Context String	ForegroundContextManag ement	As background engine Single common context

A model is a set of properties that can be inherited by an object. When an object is associated with a model, the value of each property that has not been defined inherits the value of the model. When a property value of a property is defined for an individual object, the inheritance for that value is considered “broken.” The defined value overrides the model’s property value. When the properties of a model class are updated, the values are reflected for each associated object for all property values except for those that have been defined individually.

The use of model class definitions is optional, but using them will benefit you throughout the development and maintenance of your applications. Some advantages to using the Model repository are time savings for application development, ease of maintenance, and matching field values in different tables.

In this chapter:

• Model Repository
• Data Items
• Pictures
• Field Class Properties
• Controls
• Forms
• Help Screens
• Working with Models

Model Repository

The Model repository displays the user-defined models for each object class. The Type repository of earlier versions of eDeveloper has been replaced by the Model repository.

You can select more than one model of the same object class. The Property dialog will display property values that are shared for each selected model.

Columns

- Name - The name of the user-defined model
- Class - Model-assigned objects.
- Attribute - Property assigned to an object.
- Folder -Organizes objects in the repository.
- Public Name -Defines a unique name for this model that is used by another eDeveloper application through an eDeveloper component.

Classes

Fields

Data items of previous versions of eDeveloper have been replaced with field classes. Select the field model from the Model Class list, and the field attribute from the Attribute list. The values of the system-based field model are used when there are no changes to the system-based properties.

Controls

Controls can be assigned to the model classes listed below. Not all controls are supported by each model class. The controls supported are displayed in the Attribute column.

- Browser - For tasks executed on a browser

- HTML - For tasks implemented in an HTML environment
- GUI Display - For online interactive tasks
- GUI Output - For online reports
- Text-based - For batch reports

Forms

A Form model can be assigned to the following model classes:

- Browser - For Browser forms
- HTML - For HTML forms
- GUI Display - For online forms
- GUI Output - For online reports
- Text-based - For printed reports
- Frame Set - For HTML Frame Set forms
- Merge - For HTML Merge forms

Help Screens

A help screen model can be assigned to one of the following classes:

- Internal - Helps defined within the eDeveloper application
- Windows - Helps defined outside of the eDeveloper application

Properties

For each model class, a default set of properties have been defined. A default property value is displayed in italics. You can break the inheritance of a property to an object model by clicking the Break Inheritance button that

appears to the left, as displayed in Figure 3-1. A disinherited property value does not appear in italics.

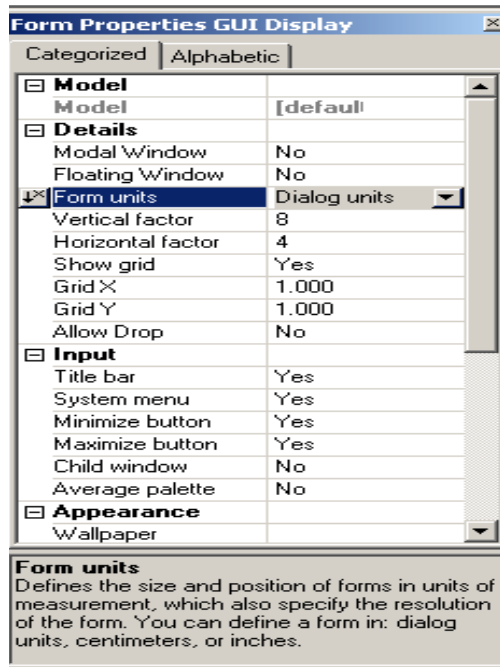


Figure 3-1 Breaking an Object's Link to a Default Property

To assign a model with the system model settings of a specific model class, click the System Model button. When a system model is assigned, you cannot change the system's property value defaults. You also cannot disinherit any property from the model.

The Set Inheritance, Break Inheritance, Go To, and System Model buttons are displayed below.



Set
Inheritance
button



Break
Inheritance
button



Defines an
expression for
the entry



Ellipsis
button



System Model
button

When you click on the value field, eDeveloper prompts you to inherit any broken properties. When you click **Yes**, the broken properties become inherited, which is indicated by the property name appearing in italic.

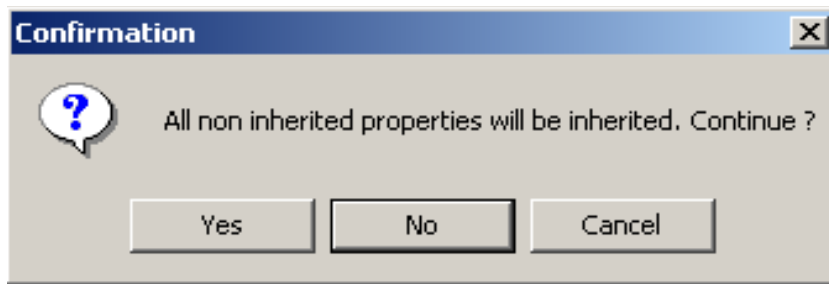


Figure 3-2 Model Inherit Properties Prompt

Data Items

In previous eDeveloper versions, data items were defined in the Type repository. In eDeveloper Version 9, the Type repository has been eliminated. Data fields are now defined in the Model repository.

eDeveloper data items are:

- **Real Columns** - components of data table rows that hold actual data values. You define the real columns for an application in the Table repository. When several different columns have the same characteristics, you can define a field model for them in the Model repository. For more information about defining columns and field models, refer to Chapter 4, Models.
- **Virtual Variables** - local variables used by an eDeveloper task for computation and temporary storage. Define the virtual variables you need when you define the task. You can also use the field models you predefined in the Model repository.
- **Parameter Variables** - local variables designated to receive values from a called task or program. For more information, refer to Chapter 8, Operations.

Data Item Qualifiers

Data items are defined in terms of attributes (required) and pictures (required), as well as other optional properties such as ranges and storage field models.

Attributes

The attribute of a Field model, which you enter in the Attribute property of either the Model repository or the Column list, specifies the nature of the information held. Attributes available in eDeveloper are shown in the table below. Click the Attribute box to select the appropriate attribute.

Attribute	Represents
Alpha	a string of alphanumeric characters. The maximum length for virtual variables is either 32K, the amount of available memory, or maximum column and row lengths of the underlying database. For Btrieve™ this may be 4096 bytes, assuming Btrieve was loaded with the /P:4096 page size property.
Numeric	an integer or decimal number. eDeveloper supports up to 18 digits, with the condition that the number of whole digits and decimal digits are each rounded up to the nearest even number.
Logical	a field usually stored internally as a single byte with value either 0 or 1. Use logical attributes when you are storing pairs of values, such as True/False, Black/White, Yes/No. Logical attributes are usually accessed more quickly than their equivalent Numeric attributes. 0 represents False and 1 True.
Date	an attribute that eDeveloper stores internally as Numeric although it can also be specified as a String attribute. The numeric date attribute is a counter of days since 01/01/01 or since 01/01/1901, depending on its storage field model. The fact that a Date attribute is stored as a numeric value lets eDeveloper perform calculations to create new date values. A Date attribute is translated to its visible value only when it is displayed.

Attribute	Represents
Time	an attribute that eDeveloper stores internally as a counter of seconds. You can use a Time attribute to represent either a duration of time or an absolute time value. Just as with Date attributes, Time attributes can be subtracted from one another, and values can be added to or subtracted from them. A Time attribute is translated to its visible value only when it is displayed.
Memo	a variable length alpha attribute. eDeveloper attempts to store this attribute in a varying length format, utilizing the nearest equivalent attribute in the underlying database. In Btrieve, this attribute is stored after the fixed part of the record. If the underlying database doesn't support this attribute, the attribute will be stored as Alpha. Once a Memo attribute is read into eDeveloper, it behaves as an Alpha attribute, including the consumption of internal memory based on its maximum length. For this reason, be careful when you allocate the maximum size of a Memo attribute, and do not use a Memo attribute when an Alpha attribute will suffice. A Memo attribute cannot be part of an index.
BLOB	a Binary Large Object. An attribute that contains binary information not created in eDeveloper, and of unknown size. eDeveloper stores this information as is, without understanding the contents. A common use for BLOB attributes would be to store OLE objects or image bitmaps. Functions cannot act on BLOB attributes with the exception of the NULL() function that will return 'True' to signify an empty BLOB.

Attribute	Represents
OLE	A BLOB field that is used to create an instance of an OLE COM object. For OLE field properties, see page 227.
ActiveX	A BLOB field that is used to create an instance of an ActiveX COM object. For ActiveX field properties, see page 227.
Vector	<p>The eDeveloper Vector is basically an array that lets you store and retrieve data from a specified cell index. The Vector attribute is based on the BLOB attribute with an additional cell model property.</p> <p>The Vector cell must be specified from an eDeveloper field model, as defined in the Models repository. The model can be any field data attribute: Alpha, Numeric, Logical Data, Time, Memo, BLOB, OLE, ActiveX, or Vector.</p> <p>Vector indexing starts from one. The Vector attribute can only be selected from Virtual and Parameter fields. You cannot directly store vectors in a table.</p> <p>Recursive vector definitions are not supported. You cannot put a Vector variable on a GUI or Browser form. It is not recommended to store large amounts of data in a vector because the array is stored in the computer's memory.</p> <p>You can access and modify the vector cells by using the vector functions described in Chapter 8, Expression Rules.</p>

Storage Field Models

A *Storage Field Model* refers to the machine representation of a data item. eDeveloper automatically associates a default optimized storage field model to each attribute. If your application data files will not be shared with another database, you can ignore storage field models. If you will be accessing files previously created by a non-eDeveloper application, you may have to solve compatibility problems by specifying particular storage field models for data columns. To do this, you specify the required storage field Pictures

Each data item requires a picture qualifier that is strictly related to its attribute. Picture qualifiers serve three purposes:

- Define the actual size and storage format of the data items
- Customize the default visual representation of a data item on screen forms or reports (refer to Chapter 7, Programs).
- Define expressions that specify how to convert a data item from its current attribute to another, for example from numeric to string. For more information, refer to Chapter 9, Expression Rules.

Pictures

A *picture* is a string of characters that tells eDeveloper how to define the format of an attribute. You can specify the picture format in the Picture dialog, which is accessed from the Picture property in the Details section of the Field Properties sheet. eDeveloper automatically defines the most commonly used pictures for data items that have known attributes (for example, dates).

You may be required to specify a picture in three different situations:

1. To define the attribute's size and default form in the Model or Table repositories. This way you control the attribute's data input and

output representation.

Attribute Name	Attribute Type	Picture	Meaning
Title	Alpha	15	A 15-character alphanumeric attribute
ID Number	Numeric	### or 3	3-digit integer numeric attribute
Due Date	Date	MM/DD/YY	Date attribute format for input and output
Percent	Numeric	###.## or 3.2	A number attribute with 3 whole digits and up to 2 decimals.

2. To override the default format of an attribute to display it on a screen or report form (refer to Chapter 7, Programs).
3. To specify data conversion rules for eDeveloper data conversion functions. For more information, refer to the Expressions chapter. For example, the Str function translates a numeric attribute to a string representation. Therefore `Str (712.93, '###.####')` returns the string 712.9300.

The picture string is composed of three basic types of characters:

1. Symbolic characters interpreted as functional directives.
2. Symbolic characters interpreted as positional directives.
3. Characters used with their proper value as mask characters.

Each eDeveloper attribute has its own set of picture directives.

A description of each of the functional and positional directives pertaining to each attribute follows.

Functional Directives

A functional directive is a picture character that is interpreted the same way regardless of its location in the picture. For example, in the following three pictures: N####, ####N, ##N##

- Each defines a 4-digit numeric integer attribute that can hold positive and negative values.
- N is the functional directive that specifies that the attribute may also contain negative values. The exact location of N inside the picture string is irrelevant, as is the location of any other functional directive.

eDeveloper adds functional directives to the picture string when you enter \mathcal{V} in the relevant prompts in the Picture dialog, as explained below in Defining Pictures, or when you insert functional directives manually by editing the picture string in the Picture attribute.

Note that all functional directives must appear in upper case.

Functional and Positional Directive Defaults

eDeveloper allows you to shorten the definition of the most frequently used attributes, Numeric and Alpha, whose pictures include only positional directives.

If you start the picture for a Numeric or Alpha attribute directly with a count value, eDeveloper assumes that the proper default positional directive should be repeated. The default directive chosen depends on the attribute:

Attribute	Default directive
Alpha	X (a place holder for any character)
Numeric	# (a place holder for a digit)

This way you can easily define the picture used most, as illustrated in the following table.

Attribute	Actual Picture	Equivalent Picture
Alpha	7	XXXXXXX or X7
Alpha	3U2	XXXU2 or X3UU
Numeric	4	#### or #4

Functional Directives for Numeric Pictures

The functional directives are automatically added to the picture string if required, according to the answers to the following prompts in that attribute's Picture dialog:

Directive Value	Means
Auto skip: No (default)	Select Yes to add the functional directive A to the attribute picture. This directive instructs eDeveloper to move automatically to the next field, without waiting for a next field action, when the last character of the field has been typed during the data entry.
Negative: No (default)	Select Yes to add the functional directive N to the field picture. This directive tells eDeveloper that this field may contain negative values. If Yes is specified, eDeveloper automatically enlarges the display size of the picture, as portrayed by the template, to enable a sign to be displayed.
Commas: No (default)	Select Yes to add the functional directive C to the field picture. This directive instructs eDeveloper to insert commas as thousands separators in the field. If you specify Yes, eDeveloper automatically enlarges the display size of the picture, as portrayed by the template, to include positions for commas. Commas may be replaced by any other character as specified in the Thousands Separator field of the Environment dialog. For more information, see Chapter 2, Settings.

Directive Value	Means
Left justified: No (default)	Select Yes to add the functional directive L to the field picture. This directive instructs eDeveloper to left-justify the field value when displayed. By default the value is right-justified.
Pad fill: No Character	Select Yes to add the functional directive P to the field picture. This directive instructs eDeveloper to fill the part of the displayed field that does <i>not</i> contain digits with the character specified in "Character:" If you don't specify any character, the field is padded with blanks.
Zero fill: No Character	Select Yes to add the functional directive Z to the field picture. This directive instructs eDeveloper to fill the entire displayed field with the Character value if its value is zero. If you don't specify any character, the field is filled with blanks.
Negative sign: - Suffix -	If you want to display a string instead of the conventional minus sign (-) in front of a negative value, override the - in the <i>Negative Sign</i> field with your own string. To display a string at the end of a negative value, type the string you want in the Suffix field. These strings, called <i>sign strings</i> , are added to the picture string according to the rules explained below in Synopsis of the Sign String.
Positive sign: Suffix	If you want to display a string instead of the conventional blank for the positive sign, type your own string in the <i>Positive Sign</i> field. To display a string at the end of a positive value, type the string you want in the Suffix field. These sign strings are added to the picture string according to the rules explained below, in the section Synopsis of the Sign String.

Synopsis of the Sign String - If you have specified an alternate setting for the Negative or Positive sign strings in the Numeric Picture dialog:

- The functional directives - or + are added to the end of the picture string.

- The sign directives are followed immediately by the new sign prefix, a comma, the new sign suffix, and then terminated by a semicolon (;). For example, 5.2-(,); is used to enclose a negative number within parentheses. The picture 4.3+,CR;- ,DB; specifies that positive numbers should be suffixed with the string CR and negative numbers with the string DB. The character used to split the string into prefix and suffix is a comma.
The bottom area of the dialog shows the exact template of the field as it will be displayed, that is, which mask characters are displayed and how many positions the field will occupy on screen or report forms.

Examples of Numeric pictures are displayed below.

Contents of attribute	Picture	Result	Comment
-1234.56	#####.# #	^^1234.56	Negative not allowed
-1234.56	N#####. ##	^^- 1234.56	Negative allowed
-1234.56	N#####. ##C	^^- 1,234.56	Commas in display
-1234.56	N#####. ##L	- 1234.56^^	Left-justified
-1234.56	N#####. ##P*	-**1234.56	Pad fill with asterisks
0	N#####. ##Z*	*****	Zero fill with asterisks
-13.5	N##.##- DB;	DB13.50	Sign of negative values is DB
45.3	N##.##+C R;	CR45.30	Sign of positive values is CR

Contents of attribute	Picture	Result	Comment
-13.5	N##.##- (.);	(13.50)	Negative sign prefix and suffix is
4055.3	#####. ##	\$^^4055.3 0	Mask character used in display only

The ^ symbol represents a one-space character.

Functional Directives for Date Pictures

The functional directives are automatically added to the picture string if required, according to the properties selected below:

- Auto skip (Y/N): N (default) - Type Y to add the functional directive A to the field picture. This directive instructs eDeveloper to move automatically to the next field, without waiting for a next field action, when the last character of the field has been typed during data entry.
- Zero fill: No Character - Select Yes to add the functional directive Z to the field picture. This directive instructs eDeveloper to fill the entire displayed field with the Character value if its value is zero. If you don't specify any character, the field field is filled with blanks.
- Trim text: No - Select Yes to add the functional directive T to the field picture. This directive instructs eDeveloper to remove any blanks created by the positional directives 'WWW...' (weekday name), 'MMM...' (month name), or 'DDDD' (ordinal day, e.g. 4th, 23rd). Since these positional directives must be specified in the picture string using the maximum length possible, unwanted blanks may be inadvertently created for names shorter than the specified length. The Trim Text directive will remove all such blanks.

If a space is required nevertheless, it must be explicitly inserted in the picture string as a mask character, using the ^ symbol to indicate a blank character, such as

TWWWWWWWWW^DDDD^MMMMMMMM,YYYY

Sample - Because eDeveloper defines the default picture mask for a Date attribute as ##/##/##, this area initially shows MM/DD/YY, DD/MM/YY or YY/MM/DD according to the Date mode (American, European, or Scandinavian) set in the Environment dialog, as explained in Chapter 2, Settings.

If you have changed the default picture, this area shows the exact template of the attribute as it will be displayed; that is, which mask characters are displayed and how many positions the attribute will occupy on screen or report forms.

Examples of Date Pictures - The date used in the following examples is March 21, 1997. The contents of the date attribute are therefore 729103, which is the number of days from 01/01/0001.

Picture	Result and Notes
MM/DD/YY	03/21/97
DD/MM/YY	21/03/97
YY/MM/DD	97/03/21
DD/MM/YY	21-03-97 when the Date Separator attribute is set to -
DD-MM-YYYY	21-03-1997 where - is a mask character
YY.DDD	97.081
##/##/##	03/21/97, when the Date Mode attribute is set to American 21/03/97, when the Date Mode attribute is set to European 97/03/21, when the Date Mode attribute is set to Scandinavian

Picture	Result and Notes
MMMMMMMMMMM^DDD D, ^YYYY	March^^^ ^^21st, ^1997
MMMMMMMMMMM^DDD D, ^YYYYT	March^21st, ^1997 with trimming directive
WWWWWWWWWWW^ ^W	Saturday^^^ - ^7
WWWWWWWWWWW^ ^WT	Saturday^ - ^7 with trimming directive

The ^ symbol represents a one-space character.

Functional Directives for Time Pictures

The functional directives are automatically added to the picture string if required, according to the answers to the following prompts.

- Auto skip: No (default) - Select Yes to add the functional directive A to the attribute picture. This directive instructs eDeveloper to move automatically to the next attribute field, without waiting for a next field action, when the last character of the attribute field has been typed during data entry.
- Zero fill: No Character - Select Yes to add the functional directive Z to the attribute picture. This directive instructs eDeveloper to replace any 0 of the displayed attribute with the character value. If you don't specify any character, the blank character is used.

Sample - Because eDeveloper defines the default picture mask HH/MM/SS for a Time attribute, this area initially shows HH/MM/SS also. If you change the default picture, this area shows the exact template of the attribute as it will be displayed, That is, which mask characters are displayed and how many positions the attribute will occupy on screen or report forms.

Examples of Time Pictures are displayed below.

Content (Number of seconds since 00:00 hours)	Picture	Result	Comments
30000	HH:MM:SS	08:20:00	Time displayed on a 24-hour clock.
60000	HH:MM:SS	16:40:00	Time displayed on a 24-hour clock.
30000	HH:MM PM	8:20 am	Time displayed on a 12-hour clock.
60000	HH:MM PM	4:40 pm	Time displayed on a 12-hour clock.
60000	HH:MM PM	4-40 pm	When Time Separator is set to - in the Environment dialog.
60000	HH-MM-SS	16-40-00	- is a mask character.

Functional Directive List

A functional directive may appear only once, in any position within a picture.

Directive	Attributes	Usage
T	D	Trim MMM..., WWW..., or DDDD.
##/##/##	D	Display Date parameter according to the definition in the Environment dialog, either American, European, or Scandinavian.
N	N	Negative value is allowed.

Directive	Attributes	Usage
+s{,m}	N	Add Prefix string s and Suffix string m to positive number.
-s{,m}	N	Add Prefix string s and Suffix string m to negative number.
C	N	Use commas, or the Thousands separator specified in the Environment dialog, where appropriate in numeric attributes.
L	N	Left justify. The default is Right justify.
Pc	N	Pad with fill character c for output.
Zc	N, D, T	If attribute field is zero, fill with character c.
A	all	Auto skip for input.

Positional Directives

A positional directive is a picture character that serves as a place-holder and is then interpreted with respect to its position in the picture string. For example, in the following three pictures:

UXX, XUX, XXU

- Each defines a different 3-byte Alpha attribute.
- The positional directive `U` for an Alpha attribute instructs eDeveloper to convert the character corresponding to the `U` position to upper case during data entry from the keyboard. The picture `UXX` causes the conversion of the first typed character to upper case, while `XUX` causes the conversion of the second typed character, and so on. Therefore, the position of any positional directive is critical.

Note that all positional directives must appear in upper case.

Positional directives may appear several times in a picture. For a shortcut method of specifying consecutive repeated mask characters, refer to the section on Count Value.

Positional Directives for Alpha Pictures

The table below provides a list of positional directives for Alpha pictures.

Directive Character	Holds a place for...
X	any character
U	a character that will be converted to upper case when typed from the keyboard
L	a character that will be converted to lower case when typed from the keyboard
#	a digit (0-9) only. Note that during data entry, eDeveloper verifies that the user typed a digit in the positions held by #. Any other character typed in those positions is rejected and an error message is issued

Alpha Picture properties include:

Auto skip: **No** is the default option.

Select **Yes** in the Picture dialog to add the functional directive **A** to the attribute picture. This directive instructs eDeveloper to move automatically to the next field, without waiting for the next field action, when the last character of the attribute has been typed during the data entry.

This functional directive is automatically added to the picture string if required.

Positional Directives for Numeric Pictures

The table below provides a list of positional directives for Numeric pictures.

Directive Character	Action
#	holds a place for a digit

Directive Character	Action
.	indicates the location of the decimal point

Positional Directives for Logical Pictures

X is a place holder for a character to be used in translating the internal values to the display/input values: True for 1 or False for 0. In addition, if a range was specified on the attribute, alternate display/input values can be used, such as Male, Female.

Positional Directives for Date Pictures

The table below provides a list of positional directives for Date pictures.

Picture	Meaning	Range
DD	A place holder for the number of the day in a month	1-31
DDD	The number of the day in a year	1-366
DDDD	The ordinal day number in a month	displayed as 1st, 2nd, 3rd, 4th, etc.
MM	A place holder for the number of the month in a year	1-12
MMM...	Month displayed in full name form (up to 10 'M's in a sequence). e.g. January, February. If the month name is shorter than the number of 'M's in the string, the rest of the 'M' positions are filled with blanks.	N/A
YY	A place holder of the number of the year	0-99

Picture	Meaning	Range
YYYY	A place holder for the number of the year, represented in full format (e.g. 1993)	N/A
W	Day number in a week	1-7
WWW...	Name of day in a week. The string can be from 3 to 10 'W's. If the name of the day is shorter than the number of 'W's in the string, the rest is filled with blanks.	N/A
/	Date separator position. The system will replace this character with the character defined in the Environment dialog as the Date separator. Refer to Chapter 2, Settings.	N/A
##/##/ ##	Display date as DD/MM/YY or MM/DD/YY or YY/MM/DD according to the Date Mode attribute setting in the Environment dialog.	N/A

Positional Directives for Time Pictures

The table below provides a list of positional directives for Time pictures

Directive Character(s) of Values	Function	Legal Range
HH	Place holder for the hour	00-23
MM	Place holder for the minutes	00-59
SS	Place holder for the seconds	00-59

Directive Character(s) of Values	Function	Legal Range
:	Time separator position. eDeveloper replaces this character with the character defined in the Environment Time Separator attribute.	
PM	Place holder for the AM/PM attribute. PM restricts the maximum value of the HH directive to 12	AM or PM

Summary of Picture Directives

In the following description of picture directives, the lower case characters listed here have special meanings:

- *n* - represents the count value
- *s* - represents a string
- { } - represents an optional part
- *c* - represents a mask character. To use a directive character, for example *x*, where *c* appears, preface it with a backslash, as in \x.

Positional Directive List

The positional directives are place holders for various classes of characters.

- For Alpha and Memo attributes, each *x*, *u*, *l*, or *#* directive defines the type of character. The sum of *x*s, *u*s, *l*s, and *#*s defines the size of the attribute.
- For attributes with the Numeric attribute, the *#* directives, the count values, and the optional decimal position character define the number of digits in the whole part and in the decimal part. This in turn defines the default storage field model and length of the parameter.

- Using the short form you can optionally specify the count values only, without a positional directive, implying the # directive for Numeric attributes or the X directive for Alpha, Memo and logical attributes.

Directive	Attributes	Usage
X{n}	A	Any character is accepted for input
U{n}	A	Input is translated to upper case
L{n}	A	Input is translated to lower case
#n	A, N	Digits are accepted for input
.	N	Decimal position for the Decimal separator, as defined in the Environment dialog.
YY	D	Last two digits of year (e.g., 97)
YYYY	D	Four digits of year (e.g., 1997)
MM	D	Month number (1-12)
MMM...	D	Month name (length 3 to 10 characters)
DD	D	Day number in month (1-31)
DDD	D	Day number in year (1-366)
DDDD	D	Ordinal day of month (1st, 2nd, 3rd,...)
W	D	Day number in week (1-7)
WWW...	D	Day of week name (e.g., Sunday, Monday,...), 3-10 characters.
/	D	Date separator position (for the Date separator defined in the Environment dialog)
HH	T	Hour (0-99)

Directive	Attributes	Usage
MM	T	Minutes (0-59)
SS	T	Seconds (0-59)
PM	T	Displays am/pm and changes hour from 24-hour clock, 1-23, to 12-hour clock, 13-23
:	T	Time separator position (for the Time separator defined in the Environment dialog).

Mask Characters

Any character that appears in a picture string and is neither a functional nor a positional directive for the specific attribute is a mask character. Mask characters are inserted into the actual attribute value during display. For example, the string `$#####` defines a 5-digit numeric integer attribute that is always displayed with a dollar sign preceding it.

- The position of mask characters is critical. Specifying the dollar sign at the rightmost position of the picture `#####$` causes the dollar sign to be displayed in the rightmost position of the attribute.
- Mask characters influence neither the size of the stored attribute nor its internal representation.
- Mask characters may appear several times in a picture. For a shortcut method of specifying consecutive mask characters, refer to the section on Count Value.

Syntax Rules for Constructing Pictures

Case Sensitivity

Functional and positional directives must always be specified in upper case letters. Lower case letters are interpreted as mask characters only.

For example:

The picture `xxxxx` defines a 5-character Alpha attribute.

the picture `xxxxx` defines a 4-character Alpha attribute with the intermixed mask character `x`.

Escape Character

The escape character `\` explicitly specifies that the character immediately following is a mask character. This allows you to override the implied meaning of the directive character and to use a mask character instead.

For example:

The picture `xx\xxx` defines a 4-character Alpha attribute with the mask character `x` in the middle. The example shows the character `\` telling eDeveloper to interpret the `x` that follows as a mask character.

Suppose you have a four-digit numeric field containing a weight expressed in pounds. You want to display it with a `P` in front of the value. If you simply specify the picture `P####`, eDeveloper interprets the `P` as the Pad-fill directive, which is not what you want. The correct picture for your needs is `\P####`.

Note: To use the `\` character as a mask character by itself, specify it twice, as in `\\`.

Count Value

The count value is a quick method of repeating the same character consecutively in a picture. The count value is a number you put after a character that indicates how many times the character must be repeated.

For example:

\$#4 means \$####

X6 means XXXXXX

X3U2 means XXXUU

The count value can be used for positional directives or for mask characters.

Field Class Properties

The property sheet for Field models includes Storage and SQL properties that were displayed for Column properties in the previous eDeveloper versions.

Model

You can re-inherit any broken properties or disinherit all properties for the form model. The default value is zero.

Details

- Picture -You can define the field model picture by zooming to the Picture dialog box. The default value is zero.
- Attribute - eDeveloper displays the field model attribute.
- Cell Model - You can zoom from the Vector Cell property to open a selection table of field models.

- Range - You can set the range by entering the required range values. The default value is zero.

ActiveX and OLE

- Type Library - The type library defines the object type to be used. For OLE fields, you should first select the type library by zooming from this property to browse through the objects registered in the current machine and select the type library of the object.

For ActiveX fields, the type library is automatically set after selecting the object name. Until the ActiveX object name is selected, the Type Library property is disabled.

- Object Name - This property specifies the actual object within the type library. The type library may support several type of objects.

For OLE fields, you should zoom to select the object name after you have selected the type library. The list of objects displays all the objects supported by the selected type library.

For Active-X fields, you should first zoom to select an object. The list of objects displays all objects that are registered in the current machine as insertable controls.

- Sub-Object Name - Some objects are constructed by a hierarchy of sub-objects. If the object you selected has sub-objects, you can zoom from this property to select the sub-object.

Use the sub-object field to store references of the sub-object as it may be retrieved at runtime.

Some COM objects include Collection Type sub-objects. A collection item is usually retrieved from an ID field. You can use the **MGIItemSequential** internal method to retrieve an item from the collection by specifying its sequential index.

- Instantiation - When set to Automatic, eDeveloper automatically instantiates the object as the task is opened and releases the object as the

task closes. When set to None, you can create or release the object manually by using the COMObjCreate and COMObjRelease functions.

- Remote Host - You can define a DCOM object instantiated on a specified remote host machine by entering the Host IP or Host Name. The remote host can be set by a regular string, logical name, or secret name by using the percent character, such as %DCOM_Host%. The property default value is blank. If the remote host is not specified, the object is instantiated according to the object's settings on the running operating system.
- ActiveX Default - This property is available for Active-X fields only. You can zoom from this property to open the Default Settings dialog that is provided by the object. You can change the default values that can be stored as part of the field definition. Not all objects provide a Default Settings dialog.

Input

- Select Program - Determines whether the end-user can open a program at runtime.
- Select Mode - Specifies when eDeveloper calls the program defined in the Select Program field.

Appearance

- Help Screen - Zoom to the Help list to link the Help screen to the control to which the field model is assigned.
- Tooltip - Zoom to the Help list to link the tooltip to the control to which the field model is assigned.
- Help Prompt - Zoom to the Help list to link the Help prompt to the control to which the field model is assigned.

Style

Select the form model and click the control list arrow to choose the control that you want to assign to the model, and then zoom to open the control's property sheet. The following form models are available:

- Browser - Controls that can be displayed on the browser form.
- Browser Table - Controls that can be displayed in table on the browser form.
- HTML - Controls that can be displayed on the HTML form.
- GUI Display - Controls that can be displayed on the GUI Display form.
- GUI Display Table - Controls that can be displayed in a table on the GUI Display form.
- GUI Output - Controls that can be displayed on the GUI Output form.
- GUI Output Table - Controls that can be displayed in a table on the GUI Output form.
- Text-based - Controls that can be displayed on a Text-based form.

Def/Null

- NULL Allowed: No (default)

Yes means that all columns based on this class (database columns or virtual variables in programs) accept NULL as a valid value.

No means that eDeveloper rejects NULL values on all columns based on this class.

- NULL Value: (optional)

Input to the NULL Value property is permitted only if the NULL Allowed parameter in the properties dialog box is set to **Yes**.

- NULL Display: (optional)

Input to the NULL Display property is permitted only if the NULL Allowed

property in the Model Properties sheet is set to **Yes**.

- Default NULL

The NULL Default property determines whether there will be a default value assigned to a NULL.

- Default Value: (optional)

The Default Value property assigns a specific default value.

- Database Default: The default database assigned to the field.

Storage

- Character Set

The Character Set property can be an ANSI, OEM, or Unicode character set.

- Default Storage

This property supports the transparent portability among databases during runtime that is independent of the gateway used during development.

If you select the Yes value, the following field properties are disabled:

- Stored As
- Size
- Definition

- Modifiable

This property determines if the end-user can modify the column value in runtime.

SQL

- Database Information

The Database Information property contains customization information

that is transferred to the Database Manager gateway at runtime.

- DB Column Name

This is the actual name of the column as it is defined in the underlying database.

- Type

This is the SQL data type of the column in the underlying RDBMS.

- User Type

This is the User Defined Type (UDT) as defined in the database.

Control Properties

You can assign a model to a control by selecting the required control from the Attribute column (such as Edit, Text, Push button, Check box, Radio button, and so on). The control properties for all control classes are listed below. For a description of the specific properties for a control, see the Form chapters.

Details

- Data - Data can be defined as either a variable or an expression.
- Button Style - Specifies the design of the button.
- Column Title - Lets you assign a title to a column. A multi-line title displays line breaks. If the column title is multiple lines, activate the Wide option, **F6**, in the Column Title property field.
- Enable Rich Format - Lets you work in Rich Text Format, RTF.
- Control Name - lets you name a control.
- Format - Determines the format for Edit and Push button controls.
- Label Format - Assigns a descriptive label to a Push button control, or as a format for the label of an existing Push button control.

- Label - A textual description for a Static control.
- Range - A range of values.
- Attribute - Determines the value for the selected variable.
- Default Image File Name - Specifies the name of the bitmap file for display on a Push button for an image button.
- Default Image File Resource - Specifies the path to the Default Image file.
- Return Action - Specifies the action that is sent to eDeveloper when the end-user clicks the button control.
- Sortable - Lets you sort rows in the column.
- Marking Column - Lets you select a column in a table.
- Static Type - Displays the type of Static control you have placed on the form.
- Text - Specifies the text that appears on a Static control and a Check Box control.
- Allow Dragging -When this property is set to Yes, the developer can drag the selected control within the eDeveloper application and from an eDeveloper application to another application. The Drag Begin event is raised for the control when the mouse's left button is kept clicked and the mouse device is dragged 3 pixels to any side.

OLE and Active-X controls do not support Drag functionality.

The DragSetData and DropFormat functions define the data content and format for controls that are not automatically handled for drag and drop functionality. For more information, see the DragSetData and DropFormat functions in Chapter 8, Expression Rules.

- Allow Dropping - When this property is set to Yes, the developer can drag the selected control within the eDeveloper application and from an eDeveloper application to another application. The Drag Begin event is raised for the control when the mouse's left button is kept clicked and the mouse device is dragged 3 pixels to any side.

OLE and Active-X controls do not support Drop functionality.

The DragSetData and DropFormat functions define the data content and format for controls that are not automatically handled for drag and drop functionality. For more information, see the DragSetData and DropFormat functions in Chapter 8, Expression Rules.

Input

- **Must Input** - Determines whether the end-user must enter a value to this control.
- **Modifiable** - Determines whether the end-user can change the value in the control during Runtime.
- **Multi-line Edit** - Determines whether an Edit control can contain more than one line of text.
- **Vertical Scroll** - Allows for vertical scrolling, when Multi-line Edit is set to Yes.
- **Horizontal Scroll** - Allows horizontal scrolling, when Multi-line Edit is set to Yes.
- **Allow CR in Data** - Determines whether eDeveloper exits the control or moves down one line when the end-user presses **ENTER**.
- **Select Program** - Determines whether the end-user can open a program at Runtime.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program parameter.
- **Selection Mode** - Determines whether the Browser or GUI Display List Box control is set for a Single or a Multiple item selection.
- **Password Edit** - Determines whether access to the control is limited to password users (asterisks will replace whatever is typed).
- **Allow Modify in Query Mode** - Allows the end-user to modify a record in Query mode.
- **View Currency** - Allows you to access a currency value from the European Currency Conversion table.

- Hidden Variable - Determines whether a control is implemented as a hidden parameter on an HTML form.
- Hyperlink - Defines the links for accessing an URL, jumping to a section in your Web page, or calling an eDeveloper program.
- Context Menu - Determines whether a context menu is associated with the control.

Appearance

- Font - Defines the font style.
- Color - Defines the color style.
- Visible - Determines whether the control appears to the end-user.
- Enabled - Determines if the control is enabled.
- Vertical Alignment - Defines the vertical alignment of text in a control.
- Horizontal Alignment - Defines the horizontal alignment of text in a control.
- Image Style - Controls the way an image is fitted into an Image control.
- Image Effects - Lets you add special video display effects to an image on an Image control.
- Style - Defines the appearance of controls.
- Border Style - Defines the style of the border of a control.
- Line Style - Defines the appearance of lines in a Static control.
- 3D Line - Defines the width of the line in a Static control.
- Border Width - Defines the width of the control border
- Line Width - Defines the Line width for a Static control.
- Slider Style - Defines the direction of a Slider control.
- Slider Step - Defines the increments within the Slider range.

- Choice Columns - Defines the number of columns displayed in a Radio Button control.
- Tab Control Side - Determines where the tabs appear for a Tab control.
- Number of Visible Lines - Determines the number of lines presented when the Combo Box control is accessed.
- Scroll Bar - Determines whether a Table control has a scroll bar.
- Context Menu - Determines whether a context menu is associated with the control.
- Row Divider - Determines whether a Row divider appears in the Table control.
- Column Divider - Determines whether a Column divider appears in the Table control.
- Last Divider - Determines whether the divider for the last column is displayed on the title bar.
- Multi-marking - Lets you mark multiple records.
- Title Height - Defines the title height of a Table control.
- Row Height - Defines the row height of a Table control.
- Help Screen - Determines whether a help screen is associated with the control.
- Tooltip - Determines whether a tooltip is associated with the control.
- Help Prompt - Determines whether a help prompt is associated with the control.
- Fix Size Table - Determines whether the table size depends on the size set in the form or depends on the number of records printed on the page.
- Paragraph Alignment - Defines the alignment of the controls row in a form.
- HTML Internal Attribute - Lets you select an attribute from the HTML Style repository.

- HTML External Attribute - Lets you select an attribute from outside of eDeveloper.
- Top Border - Determines whether the column has a top border.
- Right Border - Determines whether the column has a right border.

Navigation

- Placement - Determines whether a control is resized with the resizing of a form.
- Width - Determines the width of a control.
- Height - Determines the height of a control.
- X1 - The x-coordinate for the left point for a selected line control.
- Y1 - The y-coordinate for the left point of a selected line control.
- X2 - The x-coordinate for the right point of a selected line control.
- Y2 - The y-coordinate for the right point of a selected line control.

OLE

- OLE Class - The object class that can be inserted into the parameter.
- Display OLE As - Determines how to display the object.
- OLE Content - Determines how to store the object.
- Auto Link Update - Lets you automatically update a linked document's content bitmap.
- Use OLE Frame Type - Lets you display a different frame style around the object.

Form Properties

You can select the following forms from the Class column: GUI Display, GUI Output, Text, HTML, HTML Frame Set, HTML Merge, and Browser. Each form has its own set of properties, which are described in the following table.

Model

You can re-inherit any broken properties or disinherit all properties for the form model. You can also select a different model defined for the class.

Details

- **Modal Window** - You can specify that the form behave as a modal window, that is, you cannot click another window without first closing this window.
This property is available for Browser and GUI Display forms.
- **Floating Window** - You can specify that the form behave as a floating window. This window can be dragged outside of eDeveloper's main window.
- **Form Units** - Defines the units of measurement of a form.
- **Vertical Factor** - Defines the vertical placement of a control on a form grid.
- **Horizontal Factor** - Defines the Horizontal placement of a control on a form grid.
- **Show Grid** -Displays the form grid.
- **Grid X** - Determines the x-coordinate of the right point of a selected line control.
- **Grid Y** - Determines the y-coordinate of the right point of a selected line control.
- **Allow Drop** - The Allow Drop property enables the dropping of data onto the control from other controls in the same application or other applications running on the same machine. The data copied onto this

control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

- Form Name - Lets you name a control.
- Maximum Lines in Table - Determines the maximum number of lines in a table control.
- Header File Name - Lets you add an external file to the Head section of the generated form in Runtime.
- Title Bar - Determines whether the title bar is displayed.
- With Border - Determines whether a border is displayed.
- Frame Set Spacing - Determines the width in pixels of all of the frame borders in the frame. Disabled when the With Border property is set to No.
- Relative Size - If the relative size is set to Yes, the size of all of the frames in the frame set is defined as a percentage of the browser window or the container frame. When the browser window or container frame is resized, all frames within are resized by their percentage.
- HTML File - Defines the name of the template to be used in an Output Form operation.
- Token Prefix - Defines the prefix for merge tags in the template. The merge mechanism searches for tags that begin with this string and replaces the values according to the tags in the HTML file.
- Token Suffix - Defines the suffix for merge tags in the template. The merge mechanism searches for tags that end with this string and replaces the values according to the tags in the HTML file.
- XML Output - If the property is set to Yes, any merged value is converted to a valid XML form. All attributes should be converted because reserved characters may be used as part of the format of any variable or expression.

Input

- Expand Form - Determines whether to expand a form to accommodate multi-line edit controls.
- Average Palette - Determines whether eDeveloper should provide the average color from the many color palettes available.
- Hyperlink - Determines the URL called when a Text, Image, Square Hot Spot, or Circle Hot Spot control is clicked.
- Input Form - Determines if the HTML will be an Input form, where you can submit data to a web server.
- Context Variables - Lets you select the context variables defined in the Context Variables dialog box.
- Area - Lets you determine a specific area of the form (for example, header or footer).
- System Menu - Activates the System menu.
- Minimize Button - Determines whether the Minimize button is associated with the control.
- Maximize Button - Determines whether the Maximize button is associated with the control.
- Child Window - Determines whether the form will be opened as a secondary window.
- Help Screen - Determines whether a help screen is associated with the control.

Appearance

- Font - Defines the font for text that appears in a control. Zoom from this parameter to the Font repository and select the desired font.
- Color - The number of a color in the Color list. Zoom from this parameter to the Color repository and select the desired color.

- **HTML Int. Attribute** - Selects an attribute from the HTML Style repository, which will be added inside the control's tag, or an expression that will determine the HTML style.
- **Wallpaper** - The image file used as wallpaper for the form.
- **Visible** - Determines whether the control is visible.
- **Border Style** - Defines the style of the control's border.
- **Help Screen** - Determines whether a help screen is associated with the control.

Navigation

- **Left** - The x-coordinate of the upper left corner of a selected control.
- **Top** - The y-coordinate of the upper right corner of a selected control.
- **Width** - The width of the control.
- **Height** - The height of control.
- **Placement** - The left, right, top, and bottom placement values of how the control is resized with the resizing of the form.
- **Fit to MDI** - Enables you to set the GUI form of a task to fit the available space of the eDeveloper MDI. This property simulates the behavior of a maximized form.

When the Fit to MDI property is set to Yes, the Modal window, Floating window, Minimize button, Maximize button, and Child window are automatically set to No. In addition, the Startup Position property is disregarded.

- **Startup Position** - This property lets you to define the mode by which the GUI Display form opens. The Startup Position options are:
 - **Customized** - The window opens at the location defined in the Top and Left navigation properties of the form. The form size is defined from the Width and Height navigation properties.

- Centered to Magic - The window opens centered within the eDeveloper client area. The form size is defined only by the Width and Height properties.
- Centered to Desktop - The window opens centered within the desktop area. The form size is defined only by the Width and Height properties.
- Centered to Parent - The window opens centered within the parent window. The form size is defined only by the Width and Height properties.
- Default - The window opens with the default location and size provided by the operating system. The Top, Left, Width, and Height properties are ignored.

Help Properties

The Internal Help properties are:

Model: (default)

This property lets you associate a help screen to the specific field class. This help screen will be available to end-users at runtime when they request help on a column associated with this field class.

Details

- File Name - The name of the help file.
- Command - Help file commands. For example: Contents, Index, or Find.
- Key - The keys defined to navigate the help file.

Input

- Title Bar: Yes - Displays the Title bar.
- System Menu - Displays the System menu.

Appearance

- Font: 6 (default) - Defines the font size.
- Border Style: Thick (default) - Defines the style of the control border.

Navigation

- Left: 36 (default)
- Top: 0 (default)
- Width: 42 (default)
- Height: 21 (default)

Working with Models

A new object is associated with the system-based model by default. When a new application is created, it automatically accepts the property values of the system-based model. The system-based property values are displayed in italics.

Creating a Model

You can create a user-defined model by redefining the system-based property values. Double-click the new model entry to access the Property dialog. If no revisions are made in the Property dialog, the model entry will retain values of the system-based model.

When the model entry is saved, it becomes read-only. To retain application stability, existing models cannot be revised.

Deleting a Model

You can delete any user-defined model. When a model is deleted, all objects associated with the model will be undefined and will be flagged as erroneous when a check is performed.

When a model is deleted, the following warning is issued:

Associated objects will function unexpectedly.

A special warning is issued when a user-defined default model is deleted. Objects associated with the deleted default model will have *default model* displayed, and will use values supplied from the system-based model. The system-based model cannot be deleted

Breaking Model Properties

You can define an object value locally. When an object is defined locally, it will no longer inherit the current or updated property value of the associated model. Property values that have been “broken” can be re-inherited.

Selecting a Different Model for an Object

When a model is changed for another, the associated object will inherit the properties of the new model and will disregard the properties of the previous model.

Removing a Model from an Object

When the No Model option replaces an object that was associated with a user-defined model, the object will inherit the property values of the system-based model for that object class.

Expressions

When a model has an expression assigned as a property value, all associated objects have the model number, as listed in the Model repository, displayed in the Expressions field of the object. Selecting another expression for the object breaks the inheritance to the model. If you remove the expression value without replacing it, the object will not have an expression as a property value.

Rights

Application rights can be assigned to the entire Model repository. You can open the Rights dialog to set the authorization for any model in the Model repository. The following rights can be assigned:

- Query - Lets you view the models displayed in the Model repository.
- Modify - Lets you modify the Model repository content, including default models.
- Delete - Lets you delete any user-defined model. +
- Create - Lets you create a model entry in the Model repository.

Tables allow you to organize and define information categories to query, modify, add, or delete data in Runtime mode. Tables are defined by columns, which are associated with a specific field attribute (such as Alpha, Numeric, Date, Time, Logical, and so on). For a complete description of field attributes, see Chapter 3, Data Items.

In this chapter:

• Table Repository
• Column Repository
• Index Repository
• Foreign Key Repository

Table Repository

The Table repository is a composite of the Table pane and the Column pane as shown in Figure 4-1.

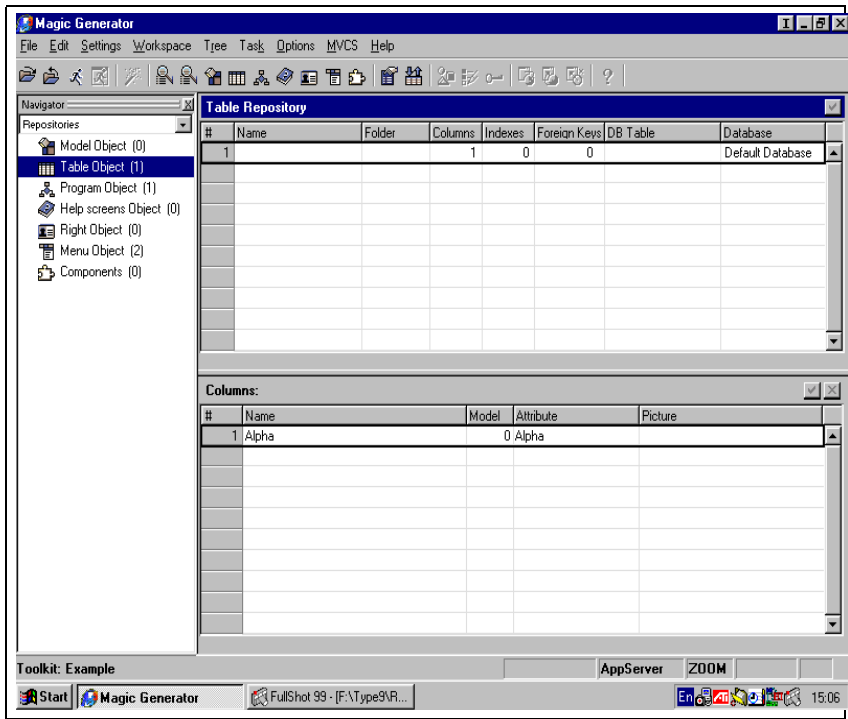


Figure 4-1 Table Repository



You can select the Table repository from the Navigation pane, Workspace menu, or click on the Tables toolbar button to create tables for your eDeveloper application. Both the Table pane and the Column pane will be displayed.

Database meta-data can be defined and maintained through the Table repository. Every modification to the Table repository can be synchronized with the underlying database, assuming that the Change Tables in Toolkit is checked in the property sheet of the selected database. For more information, see Database settings in Chapter 2, Settings.

eDeveloper neither optimizes database structures nor tunes database parameters. These tasks should be performed by a Database Administrator, DBA, with external tools. A new table created with eDeveloper is created in a generic form, and should be customized to better fit the production environment.

A table defined in the Table repository is a logical definition that is recognized by eDeveloper programs. In the database, the table can be displayed either as a table or a view. You may define several files in eDeveloper that point to the same database object.

The Table pane displays each table entry, the folder in which it is located, and statistical information regarding the following:

- Columns
- Indexes
- Foreign Keys
- DB Table
- Database
- Folder
- Public Name

The Column pane displays:

- Entry Number
- Column Name
- Model
- Attribute
- Picture

Table repository parameters are described below.

Table Repository Columns

The following describes the columns of a row in the Table repository:

(for Table identifier)

- This column contains an automatically generated sequential number used by eDeveloper as a table identifier. When you first create a table, if you have not defined an explicit DB name for the table in the Database column, eDeveloper will use the Table identifier to assemble its default table name. You cannot edit this column.

Name

- Enter a descriptive name. This name is used by eDeveloper for display of error messages, which may be seen by end-users. The maximum length for a table name is 30 characters.

Columns

- Columns defined for a table entry. eDeveloper automatically displays the most current column count. You can zoom from this column to the Column list to define columns for your table.

Indexes

- Indexes defined for a table. eDeveloper automatically displays the most current index count. You can zoom from this column to the Index repository to define indexes for your table. The use of Indexes is recommended. Additionally it is necessary to define Indexes when using some of the databases that eDeveloper can access. For more information refer to Chapter 25, SQL Considerations..

Foreign Keys

- Foreign Keys are pointers in an SQL database that let you maintain referential integrity between primary table records and records in other tables. Each record with a Foreign Key points to a record with a Primary Key and the Foreign Key repository lets you define the relationships between those records.

- You can only define a Foreign Key where a Primary Key exists.
- A Primary Key is defined for a record in the Index repository.

DB Table

- In the DB Table column you have the options to define the following:
 - An eDeveloper server name when the table resides on a remote host computer
 - A complete path for the table
 - An explicit OS entry name to override the eDeveloper default table name
 - A logical name

The maximum DB Table column length is 260 characters.

For SQL databases, the actual name of the table is defined in the underlying database. This name is limited to database specifications. For example, in Oracle the database name is limited to 30 characters that must begin with a letter.

Since the full name of an object consists of 'Owner.Tablename' or 'Database.Owner.Tablename', eDeveloper takes the owner of the table from the table properties. If no owner was specified in the Owner property (in the Table Properties dialog), then the owner is the user defined in User Name property (in the Database Properties dialog).

eDeveloper copies the table name to the DB Table column if the selected database is an SQL database. eDeveloper replaces blanks with underlines. The table name can be revised, but not left blank. For example, the table name, "my emp table", will appear as "my_emp_table" in eDeveloper. It is possible to choose a convenient naming convention for all tables by using a pre-defined eDeveloper Logical Name as a part of the table name. For more information about eDeveloper Logical Names, see the Logical Names section in Chapter 2, Settings..

For ease of maintenance and portability, it is best not to set the table location in the DB Table column, but rather to store the location in the Database

repository, because the Database repository is external to the application. By placing the table location specification outside of the application itself, the application is not “hardwired” to a particular environment.

The eDeveloper Path String

Wherever you have the option to specify a path string, it will have the format (server-name) table location where:

- server-name is one of the servers defined in the Server repository. For more information, see Chapter 2, Settings..
- table location format depends on the operating system or database being accessed. It can be a standard directory path and an OS table name, or just an OS table name, or, in some cases, may be the name of an SQL entry stored among other entries within a single OS entry.

Note: The server-name and path for a table are specified in the Database entry that the table definition uses. For more information, see Chapter 2, Settings.. You can still specify server-name and path values in the DB Table field, for compatibility with previous versions of eDeveloper.

If you leave both the DB Table field and the DB Table expression of a task empty, eDeveloper will assume that the table has the default table name, and resides in either the directory specified in the Database description or the current directory of the local workstation. If you specify a path in the Application Prefix but leave both the DB Table field and the DB Table expression empty, eDeveloper uses the path you specified for the application together with the default table name.



For SQL databases the path is ignored.

Database

- The Database name in this parameter provides access information from eDeveloper to the physical database that stores the application entries. eDeveloper uses as the default database the setting of the Default Database parameter in the Environment dialog. To choose a different database for the table, zoom from the Database column to open the

Database list. The Database list displays all the databases available to your installation. Select the Database you want from the list. For additional information on Database Settings, see Chapter 2, Settings..

Folder

- Displays the name of the folder in which the table entry is located. You can create a folder by highlighting the Table icon on the Navigator pane and clicking **F4**. Folders let you group table entries. For more information about Folders, see Chapter 1, Introduction..

Public Name

- Defines the public name of the table by which it will be called by an Internet requester or a Call Remote operation. The public name must be unique within the application file.

Table Properties

Table Properties are organized by the following tabs:

- Advanced
- SQL

Table Properties - Advanced

Access Key

The Access Key property provides the means to utilize the underlying DBMS's table security mechanism. The value of this parameter is passed to the underlying database on every open table. If an external program or other eDeveloper application tries to gain access to the table, it will have to provide the same Access Key to the underlying DBMS. Otherwise, access to the data is blocked. To make this protection installation-specific, use a Secret Name for this column. For more information, see Chapter 15, Application Properties..

The Access Key property is disabled for SQL databases.

Encrypt Table: No (default)

The Encrypt Table property is disabled when no access key is defined. When an access key is defined for the table, the table will be encrypted by specifying a **Yes** value using Access Key as an encryption seed. **No** means that the table will not be encrypted.

Note: Changing the value of the Access Key or Encrypt Table properties will cause eDeveloper to prompt the developer for change confirmation.

Cache Strategy: Position and Data (default)

The Cache Strategy specifies the cache strategy that is used with this table.

The Cache Strategy column can have one of three possible values:

Position - The cache holds information about the position of the fetched rows. This setting is relevant only when the table is used as the Main table. When you scroll backwards the data will be re-fetched by reading the rows by their physical position.

Position and Data - In addition to the position, eDeveloper will store the actual data of the row. If you read pre-fetched data, you will get old values stored previously in the cache.

None - No caching.



For more information, refer to Chapter 11, Data Management., Update/Delete Statements.

Resident: No (default)

The Resident parameter allows you to define a table as a resident table, and to determine when the table will be loaded.

The valid values for this parameter are:

- **No** - The default value, indicates that the table is not a resident table.
- **Immediate** - Indicates that the table is resident and should be loaded when the application is loaded.

- On Demand - Indicates that the table is resident but should be loaded only when it is first accessed in the application.

Identify Modified Row

This determines how eDeveloper identifies modifications made to rows by other users.

eDeveloper can select records according to:

- Position
- Position and Selected fields
- Position and Updated fields

This property applies for deferred transaction mode tasks only.



For more information, refer to Update/Delete Statements in Chapter 11, Data Management.

Size

The size of a record in a table by bytes is noted on the bottom left-hand side of the Table Properties dialog. eDeveloper updates this value automatically. In certain cases, depending on the underlying database, the actual length of a column is difficult to determine, and eDeveloper estimates it as best as it can. Columns assumed by eDeveloper to be variable in length (for example, Memo) are not included in this total length. The user cannot alter this field. It is updated automatically by eDeveloper.

Table Properties - SQL

Information for SQL Database

The Information for SQL Database parameter lets you supply database-dependent information that eDeveloper can pass to the underlying RDBMS. The use of this parameter is optional. For more information, refer to Chapter 25, SQL Considerations..

Owner

The database's owner of the table or view. This property supports logical names.

Position

This property determines the position key for a table. The available options for the Position property are:

Default - eDeveloper uses its own default as a position index for a table.

Unique Index - In other RDBMSs, eDeveloper uses the shortest unique index. This can be overwritten by using another index as the position key. If you have chosen the position as Unique Index, it is advisable to use a real index in the database to improve performance.

ROWID - In Oracle and Informix tables, eDeveloper uses the ROWID column.

Index

If the position was chosen as Index, then the Index property will allow you to choose one of the table's unique indexes.

Default Position

This is the unique identifier that eDeveloper uses as the default position. This property is for read-only.

Check Existence

This setting determines if eDeveloper checks the existence of every SQL table it attempts to access in Runtime, and to create that SQL table if it does not exist.

The As Database option uses the value of the Check Existence property from the Database Properties dialog. This option is the Check Existence property default.

Yes enables eDeveloper to create tables in the database. eDeveloper also checks for the existence of every table it attempts to access. Note that this check may cause a performance degradation.

No means that eDeveloper will not check if the table already exists before accessing it. If the table does not exist, a database error message appears.

In Runtime, you should set the Check Existence property to No for enhanced performance.

Table Type

The table type can either be Table or View. If the table type is View, the DbDel and DbCopy functions will not work, because a View cannot be created, deleted, or altered by eDeveloper. If you do a get definition on a View, you must define a unique index as the Position Key.

Hint

Some RDBMSs such as Oracle and MSSQL, allow hinting the optimizer for processing a query. In this field the programmer can enter a string that will be concatenated to the SELECT statement. For more information, refer to Chapter 25, SQL Considerations..

Yes activates the Hint property. **No** de-activates the Hint property.

eDeveloper does not evaluate the string. The developer is responsible to use the correct syntax. It is recommended to use Hints only in special cases. The Hint value can be inherited from the database properties (only when it is set to Yes).

Cursor

When using the MSSQL gateway, either internal DB commands or cursors are used. Using DB commands requires separate connections for each result set. Performance is enhanced when the result set is large.

The available options are:

Yes - enables the user of cursors on the specific table

No - disables the use of cursors on the specific table and uses DB commands instead

Default - eDeveloper uses cursors or DB commands according to the following considerations for MSSQL tables:

- eDeveloper uses cursors when the MSSQL table is designated as the main task table.
- eDeveloper uses DB commands when the MSSQL table is designated as a linked table.



For more information, refer to Chapter 25, SQL Considerations..

Array Size

The eDeveloper gateways to the various SQL databases support array processing. When fetching records from the database, the gateway does not fetch one record at a time, but rather fetches a group of records, which reduces network traffic.

The default for this property is 0. When this property is set to 0, the eDeveloper default is used. The eDeveloper default can be overwritten. When scanning a large table, increasing the array size can enhance performance. It is recommended, however, to use the eDeveloper default. Changing the array size in the table overwrites the Database property settings.



For more information, refer to Chapter 25, SQL Considerations..

Resident Tables

eDeveloper's Resident Table facility is based on the principle that certain data in an application will be used more than once. The Resident Table Facility is a tool that can be used by the eDeveloper programmer to enhance system performance by reducing disk I/O, and in the case of Client/Server configurations, to reduce network traffic. eDeveloper will read the contents of all tables defined as resident into memory, according to the setting of the Resident parameter in the Table Properties dialog.

It is important to note that resident tables are read-only, and cannot be modified while stored in memory.



For more information about the eDeveloper's Resident Table Facility, see Chapter 20, Utilities..

Table Conversion Utility

Table conversion is a process eDeveloper starts automatically if the physical structure of a table is modified and the data table exists. For example, changes such as adding a column to a table, removing a column from a table, changing a column's attribute, changing index specifications, adding or deleting a Foreign Key, or modifying a model used in a table, automatically launches the table conversion process.

It is important to note that if you change an index definition from non-unique to unique, the conversion process deletes all the rows that meet the duplicate index condition.

The table conversion process starts after you exit from a table row in the Table repository, or exit from the Table repository, or as a result of a Model repository conversion sweep.

eDeveloper maintains the structure of the table as it exists before the changes are made, along with the new structure, until the table row is left or until the table repository is left. If you cancel the conversion process, an inconsistency between the physical and logical structures of the table may remain. In this case, eDeveloper will not be able to perform an automatic conversion at a later stage and may report an error the next time it accesses the table.



For more information about The Table Conversion Utility, see Chapter 20, Utilities..

Column Repository

The Column list contains the column headings and field item information associated with each heading. To access the Column list, point to Columns in the Table repository and *zoom*.

Column Repository Fields

A description of column fields follows:

(for Column identifier)

- This column contains an automatically generated sequential number used by eDeveloper as a column identifier. You cannot edit this column.

Name

- Enter a descriptive name. If you intend to associate a Model to a column entry, you can leave the column name blank and eDeveloper automatically enters the model name. The maximum length for a Column name is 31 characters.

Model

- You can double-click the Model parameter to access the Field Model list. Field Models have the following attributes:
 1. All field model properties are inherited by the column.
 2. If the column name is empty, it is set to the field model's name.

For more information refer to Chapter 3, Models.

Attribute

- Select one of the eDeveloper data item attributes: Alpha (the default), Numeric, Logical, Date, Time, Memo, or BLOB. You can use the initial letter or click on the combo box to display the attributes. Refer to Chapter 3, Data Items., for a comprehensive description of the various attributes.

Picture

For an Alpha or Memo column, the minimum picture required is its length. In addition, you can enter other picture specifications as needed.

For a Numeric column, the minimum picture required is the number of its integer digits and, if needed, a decimal point and the number of decimal digits. You can add other picture specifications as needed.

For a Logical, Time, or Date column, you can accept the suggested picture or you can add other picture specifications as needed.

BLOB items are binary objects stored in eDeveloper without any data interpretation. Therefore, eDeveloper cannot create a BLOB item picture. For more information, see Chapter 3, Data Items..

Column Properties

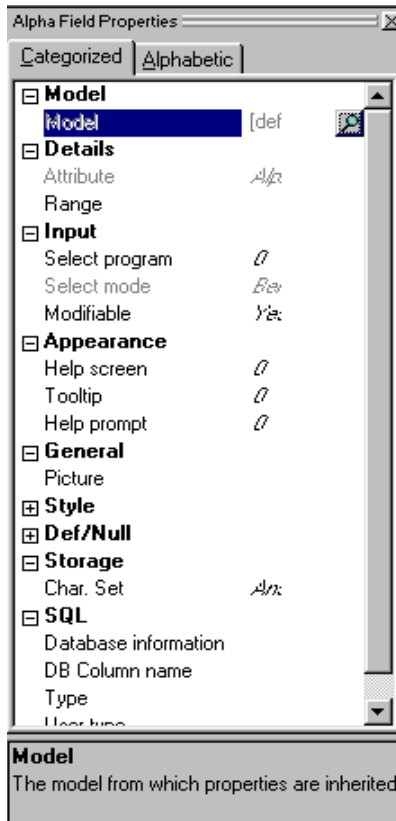


Figure 4-2 Column Properties

Field properties are associated properties that define the parameters of a particular data type. Field properties are displayed in the following sections:

Model

Lets you set or break model inheritance.

For more information see Chapter 3, Models..

Details

- Attribute - Displays the attribute of the column.
- Range - Enter valid values for the column, separated by commas. To specify a continuous range of valid values, type the minimum and maximum values, separated by a hyphen. For a complete description of Range, refer to Chapter 3, Data Items..

Input

- Select program - Use this property to associate a zoomable selection program to the data field. For more information see Chapter 6, Programs..
- Select mode - The select mode property is enabled when a select program is specified and determines when the select program will be called. The permissible values are Before, After and Prompt. For more information see to Chapter 6, Programs..
- Modifiable - You can modify the column after it has been created. If, in an Online task, you move the insertion point to a column specified as Modifiable No and try entering data there, eDeveloper will not accept the new entry.

Appearance

- Help Screen - Use this property to associate a Help Screen with the Column. The Help Screen is available to users at Runtime when they request Help.
- Tooltip - Use this property to associate a Tooltip Help to the Column.
- Help prompt - Use this property to associate a Help prompt with the column.

General

- Picture - Displays the picture dialog for the selected field type.

Style

- Browser - Displays the default form control, and the style properties for the Browser form.
- Browser Table - Displays the default form control, and the style properties for the Browser form in Table mode.
- GUI display - Displays the default form control (that is, Edit, Combo, Table, and so on) and the style properties for the GUI Interactive form
- GUI display table - Displays the default form control, and the style properties for the GUI Interactive form in Table mode.
- GUI output - Displays the default form control, and the style properties for the GUI Non-Interactive form.
- GUI output table - Displays the default form control, and the style properties for the GUI Non-Interactive form in Table mode.
- Text Based - Displays the default form control, and the style properties for the Text Base form.
- HTML - Displays the default form control, and the style properties for the HTML form.

Def/NULL

- NULL allowed: No (default)

The Null Allowed property lets you specify whether the end-user can enter a Null value in a table column. When creating a new table, the NULL constraint is derived from this property. The NULL default is derived from the DBMS section in the MAGIC.INI file.

Yes means that the column accepts the NULL as a valid value. In runtime, the NULL Allowed property lets the end-user enter a NULL value into a field before it is passed to the database.

No means that eDeveloper rejects the NULL value.

- NULL value: (optional)

Input to the NULL Value parameter is permitted only if the NULL Value Allowed parameter in the Column Properties dialog is set to **Yes**.

The calculation of the NULL Value specifies the value that should be used when this column contains a NULL value, and could be used in expressions.

- NULL display: (optional)

Input to the Displayed String parameter is permitted only if the NULL Value Allowed parameter in this same Column Properties dialog is set to **Yes**.

- NULL default

The NULL Default property determines whether a Null value or the default value will be assigned to the column. Valid values are:

- Yes - NULL is the Default Value
- No - The Default Value property will be used

- Default value

The Default Value assigned to the NULL column.

- Database default

This is the default value for the Database Default column in the database. If a program creates a record and does not select the Database Default column, then the RDBMS will assign the default value.

eDeveloper loads the database default to access the table definition. If the default is a constant, eDeveloper also loads the default database into the Default Value property to serve as the default database for eDeveloper.

When creating a new table, eDeveloper will add this string to the CREATE TABLE statement as the default value for that column. For example, when you create a database default 'defvalue' in MSSQL table, eDeveloper generates the following: `CREATE TABLE owner1.table1 (Col1 CHAR(10) NOT NULL DEFAULT 'defvalue')`

eDeveloper adds this string without any additional formatting to the CREATE TABLE statement.

Storage

- Character Set - Lets you select between ANSI and OEM character sets.

- **Stored As** - Every attribute in eDeveloper has several storage types. If not otherwise specified, the SQL gateway uses the most appropriate default mapping to the underlying RDBMS data type. For example, the Alpha attribute may be stored as a Zstring in eDeveloper. This storage is internal to eDeveloper only. When switching from one RDBMS to another, eDeveloper attempts to maintain the same storage type. For compatibility reasons, however, the storage type may differ from one RDBMS to another. Do not change the eDeveloper default storage types. For more information see Chapter 25, SQL Considerations..
- **Default Storage** - eDeveloper enables you to easily set table columns to be automatically mapped to the expected storage of the underlying DBMS according to the column's attribute and the DBMS gateway in use. This ability adds to the transparency of DBMS portability.
If set to Yes, the column will be mapped according to its default mapping, which is determined by the DBMS gateway in use.
If set to No, the column will be mapped according to the column's Storage property.
The Default Storage property is also available for virtual variables used in Direct SQL tasks.
The default value for newly created columns is No and the default value for a Direct SQL task's virtual variables is Yes.
On importing application export files of previous versions, the Default Storage property will be set to Yes if the values of the Stored As, Size, and Definition properties are the same as the default values defined by the underlying DBMS gateway that is set for the table. Otherwise, the Default Storage property will be set to No.
- **Size and Definition** - the Stored As, Size, and Definition properties reflect the storage type that describes the internal representation of the field. eDeveloper assigns the default storage type and size, which depends on the field attribute, its picture, and the database assigned.
- **Update Style** - Previous versions of eDeveloper allowed only absolute numeric updates. For example, you could only set Value X for Field FLD1 (FLD1=X)

eDeveloper Version 9 also lets you make differential updates. For example, you can update FLD1 by using the value FLD1+X (FLD1=FLD1+X). The

Update Style property has been added to the Column Properties sheet. The values for the property are:

- Absolute - a fixed value update
- Differential - differential value update

This property is only effective for Deferred Transaction mode tasks for the Numeric field type that has a normal storage type and relates to an SQL table. This property does not apply to ISAM files.



For more information see Chapter 11, Data Management., Numeric Field Updates.

SQL

- Database information - The Information for SQL Database parameter lets you supply database-dependent information that eDeveloper can pass to the underlying RDBMS. The use of this parameter is optional. For more information, refer to Chapter 25, SQL Considerations..
- DB Column name - This is the actual name of the column as it is defined in the underlying database. This name has the limitations of the specific database (such as reserved words that cannot be used in a column name).

When adding a new column to a table, eDeveloper copies the name of the column to the DB Name column, and replaces blanks with underscores. You can overwrite the name that appears in the DB column, but you cannot leave the DB Name column blank. You can also choose a convenient naming convention for all columns by adding a logical name as a prefix to the column name.

- Type - This is the SQL data type of the column in the underlying RDBMS. It is loaded when Get Definition is performed. When creating a new table in the Table repository, eDeveloper uses its own default mapping, which eliminates the need to specify the database data type. In order to force the use of a specific data type in cases where the eDeveloper default is not sufficient, a different data type may be specified.

For example, the Magic date column is mapped to the 'Date' data type in Oracle. You can, however, force eDeveloper to map the Date column to a

different data type by specifying CHAR (8) as the SQL type.

User type - This is the User Defined Type (UDT) as defined in the database. In most RDBMSs, a user defined type can be created to redefine a system datatype.

For example, the UDT 'description' is based on 'VARCHAR(20) '. Once a UDT is created, it can be used in the CREATE TABLE and ALTER TABLE statements, as well as attaching defaults and rules to it. When loading a table definition (using the Get Definition utility), eDeveloper loads the UDT of each column, and the system data type on which it is based. This is for internal use only. When creating tables in eDeveloper, eDeveloper uses only the SQL type. eDeveloper will not create the columns with a UDT



For more information, refer to Chapter 25, SQL Considerations..

Index Repository

The Index repository lets you define unique or non-unique index entries to browse through a designated table. Index entries are associated to index segments, such as identification number, email address, or telephone number information for a specific table. To access the Index repository, point to the

Index parameter of the table row you want and *zoom*. The Index repository is displayed below.

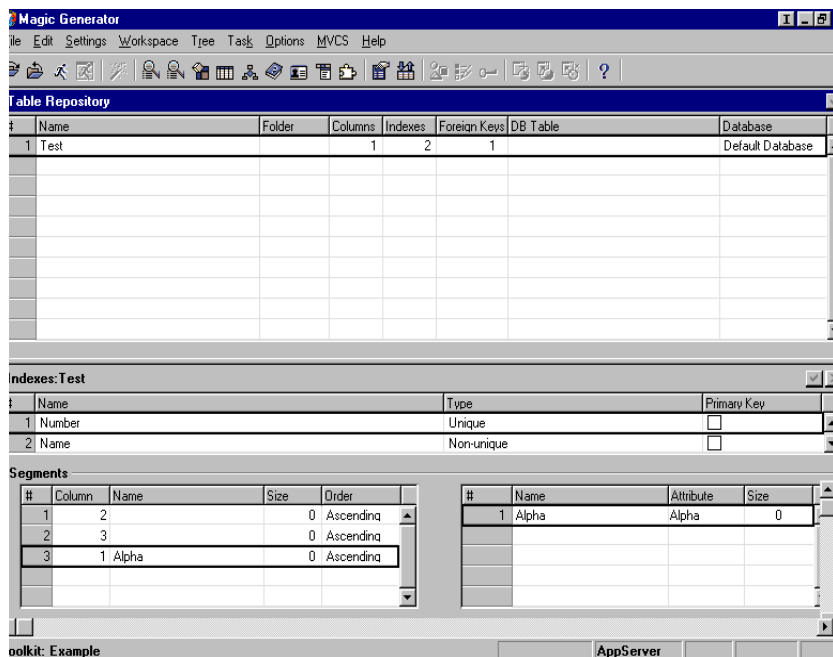


Figure 4-3 Defining an Index for a Table

The definition of the index is logical and relevant only to eDeveloper. The index in the database may differ from the index defined in eDeveloper. By using the Get Definition utility, you can access the actual index structure in the database.

It is best to build an index in eDeveloper that matches the index in the database so that no Sort operation is required by the Optimizer. The more aligned the indexes are between eDeveloper and the selected database, the faster the response time.

When defining an index in eDeveloper the following results occur:

- If you assign a real index type to a new table, eDeveloper creates an index in the database to match the eDeveloper index.

- In runtime, the eDeveloper index determines how the ORDER BY clause is added to the SELECT statement issued by eDeveloper. The segments specified in the index are the columns that are specified in the ORDER BY clause.

Index Repository Columns

This is a description of the parameters of an entry in the Index repository:

(or Index identifier)

- This contains an automatically generated sequential number used by eDeveloper as an index identifier. The cursor skips this column.

Name

- A descriptive name for the index.

Type

- Unique (the default) - means that only one row with a specific value in this index can be present in the table. Some SQL gateways require at least one unique index in eDeveloper to work with the table. For more information, refer to Chapter 25, SQL Considerations..
- Non-unique - means that duplicate index rows are allowed.

Index Segment List Columns

The order of appearance of the segments in the repository reflects their hierarchical strength (major, intermediate, minor) in order of decreasing significance. The order of appearance of segments influences the sorting sequence for this index. The Index Segment repository can store 32,767 entries.

This is a description of the columns of a row in the Index Segment repository:

(for Segment identifier)

This column contains an automatically generated sequential number used by eDeveloper as Segment identifier. The cursor skips this column.

Column (for Column number)

The number of the table column to be used as a segment. You can enter the number you want here or to zoom to the Column list displayed on the right to select the column you want.

Name

This column shows the name of the column chosen as a segment. The cursor skips this column.

Size

The cursor parks in this column only if the column attribute is Alpha. You can then shorten the size of the segment column to attempt to improve the performance of searches. If you do shorten the segment, it is advisable to define the Index as Non-Unique, to prevent loss of rows. However, specifying Non-Unique may impose performance penalties of its own, so it is not always certain that shortening a segment by a few bytes will improve performance.

When using SQL databases do not modify this property.

Order

- **Ascending** specifies an ascending sort direction of the segment.
- **Descending** specifies a descending sort direction of the segment.

The segment order may utilize an alternate collating sequence if one is specified in the Environment dialog. This happens when the segment is an Alpha column and its Translate parameter is Standard. eDeveloper supports only one alternate collating sequence at a time. It is also important to ensure that all tables being accessed when an alternate collating sequence is active were created using the same sequence. If this is not the case, eDeveloper can be confused as to the true collating sequence of these tables, and Range/Locate queries may give incorrect results.

Index Properties

Index properties are associated properties that define the parameters of a particular data type. Index properties are organized by the following tabs:

- Advanced
- SQL

Index Properties - Advanced

Direction: Two Way (default)

Two Way - tells eDeveloper that it will be necessary to enable forward and backward movement over this table, typically in an online task for databases that do not support two-way movement. It is usually beneficial to define indexes as One Way if they are to be used only in batch processing and if the database does not support two-way indexes. While some database gateways will ignore this parameter and provide two-way indexes by default, others, such as SQL, can use this information to decide whether one index is sufficient or two are required.

One Way - means that only forward access is required.



Tuning this property according to your specific database can improve performance. For more information see Chapter 25, SQL Considerations..

Range Mode: Quick (default)

Quick - means that the defined index supports a fast range execution by using the index.

Full - means that the range operation is executed by sequential scanning. A Range Mode specified as Full results in significantly slowed performance.



Tuning this property according to your specific database can improve performance. For more information, see Chapter 25, SQL Considerations..

Index Properties - SQL

Information for SQL Database

The Information for SQL Database parameter lets you supply database-dependent information that eDeveloper can pass to the underlying. The use of this parameter is optional. For more information, see Chapter 25, SQL Considerations..

DB Index Name

This is the actual name of the index, as it is defined in the underlying database. This name has the limitations of the specified database.

When adding a new index to a table, eDeveloper copies the name of the index to the DB Index Name, and replaces blanks with underscores. You can overwrite the index name, but you cannot leave the DB Index Name column blank.

For example, the eDeveloper index name 'emp ind1' becomes emp_ ind1 as the DB index name.

Index Type

The Index Type property specifies whether the index is contained in the database or defined only in eDeveloper. The permissible values are: Real or Virtual. When creating a new table, if the index is created in eDeveloper, the index type must be specified as Real.

The Index Type property has no effect on existing tables in runtime. For example, if a end-user wants to access a dataview a virtual index needs to be added.

Virtual indexes are not recommended to replace the sorting operation in the program, especially when the Sort Using RDBMS feature can be accessed. This enables the records to be fetched in a required order, so that sorting is not necessary in eDeveloper.

Hint

Some RDBMSs allow hinting the optimizer for processing a query. In this column the programmer can enter a string that will be concatenated to the SELECT statement. For more information, refer to Chapter 25, SQL Considerations..

Clustered

In MSSQL and Informix, the Clustered property specifies whether the index will be clustered when tables are created via eDeveloper. You can decide which index is clustered, otherwise the table is created without a clustered index. Assigning a clustered index to a table with a high insert ratio may cause performance problems.

A clustered index refers to the physical data, which is stored in the order of the index. A clustered index is efficient for scanning sets of data in the order of the index, and less efficient when trying to access one of the records directly.



For more information, see Chapter 25, SQL Considerations..

Drop During Reindex

This property determines whether eDeveloper will drop an existing index when the open mode of the table is Reindex in a task. For more information, see Chapter 25, SQL Considerations..

Primary Key

This check box specifies which index is the Primary Key. Only one of the unique indexes can be the Primary Key. For more information, refer to Chapter 25, SQL Considerations..

Foreign Key Repository

Maintaining the relationship between primary keys and foreign keys is known as referential integrity. Databases support the constraint listed below through their table declarations or through their declarative referential integrity.

You can define Foreign keys in the Foreign Key repository, which is a composite of two tables: the Foreign Key Definition table and the Foreign Key Segment table as displayed below.

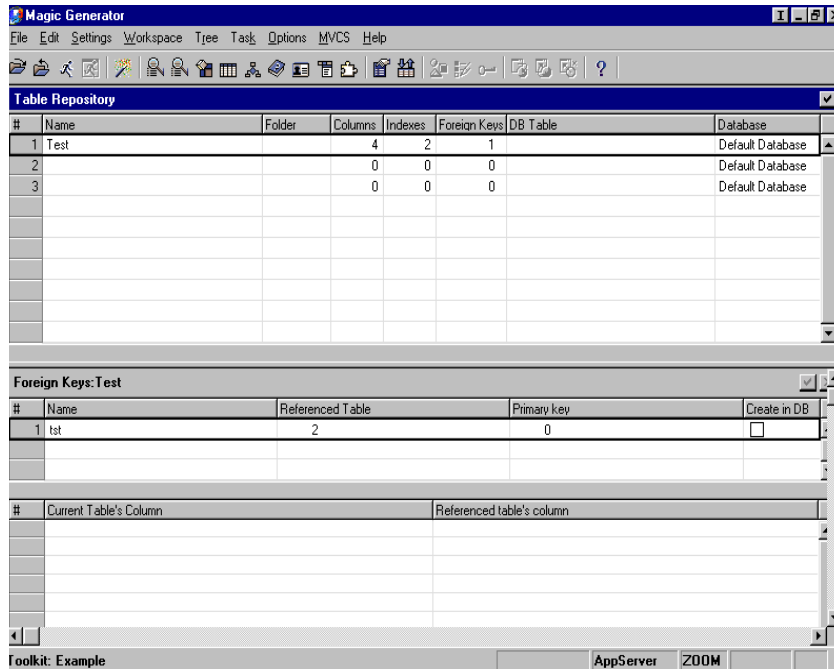


Figure 4-4 Defining a Foreign Key

Foreign Key Definition Columns

The following is a description of the Foreign Key Definition parameters:

- Foreign Key number (#) - An internal automatically generated number that represents the order of the Foreign Key in eDeveloper.
- Foreign Key Name - The name of the Foreign Key in eDeveloper. This name is used to define the Foreign Key in the database. A foreign key is a column or group of columns that relates a row in one table to a unique row in another table, usually by the other table's primary key.
- Referenced Table - The table to which the Foreign Key should refer to.
- Primary Key - The eDeveloper Primary Key name for the referenced table. The Primary Key is a column or a group of columns that has a unique key, and considered the main access route to the row. There can be one or more unique keys to a table, but there can be only one Primary Key.

- **Create in DB** - This check box lets you specify if the Foreign Key is limited to eDeveloper or also relates to the database. Create in DB is checked by default if the referenced table is in an SQL table from the same DB as the Main table. Otherwise, this check box is disabled and unchecked.

The limits listed above restrict the values of columns in the table either absolutely or in certain columns of the specific table. The current ANSI standard requires that the definition of these constraints be declared as part of the table creation or modification.

Foreign Key Segment Columns

The following is a description of the Foreign Key Segment parameters:

- **Current Table Segment number (#)** - A generated number that represents the order of the current table segment.
- **Current Table Segments** - These segments can be selected from the current table by zooming in the Column list. Each segment in the current table must have a matching segment in the referenced table.
- **Referenced Table Segment number (#)** - A generated number that represents the order of the referenced table segment.
- **Referenced Table Segments** - These segments can be automatically inserted when selecting the index defined for the referenced table. Exchanging one index for another overwrites the segments with the new index segments.

The application engine is eDeveloper's taskmaster - the runtime mode is activated by the end-user to perform the procedures comprising the application. eDeveloper lets you define the logic as a response to implicit and explicit events that may occur during the execution of a task.

In this chapter:

• Tasks
• Engine Levels and Operations Repositories
• Event Handling
• Handlers
• User-Defined Events
• Information About the Engine
• Record/Row Loop Flowcharts
• eDeveloper Cache
• Engine by Record Level

Tasks

A task is the basic object with which to construct programs to implement procedures that make up the application. Because almost every database procedure requires some variation of looping through a database table, the engine executes a task by looping through a table called the Main table of the task.

1. You define a task's Main table as a task property, selecting either one of the database tables you previously defined in the application's Table repository, or Table 0, which is a virtual scratch file in memory.
2. The second task property to specify is whether the task is Online, Batch, or Browser.

In an Online or Browser task, the end-user browses the Main table by moving from record to record. This means that the only records processed are those browsed by the end-user. If your program needs to interact with the end-user, by providing data entry screens and lookup windows, you must include Online or Browser tasks for it.

In a Batch task, eDeveloper loops automatically through the table. It simply starts with the first record in the dataview you define and visits every record until it reaches the last record. If your program has to perform automatic procedures such as generating reports and global table updates, you must include Batch tasks for it.

3. Once you have chosen the Main table and the type of looping, Online, Batch, or Browser, you can then customize the task to accomplish a specific job. You tell eDeveloper what actions, to perform and according to which rules, at each stage of the loop: task, record, control, or group for Batch tasks.

You tell eDeveloper's application engine about each task's processes by:

- Specifying the operations that must occur, while eDeveloper or the end-user loops through the table. For example, inputting data or printing.
- Placing the operations in different level Operation repositories according to when the engine must execute them.

Operation repositories are where you put all of the operations that control the task. Where you put an operation, in the Operation level, determines at which stage it is executed by the engine, see Figure 5-1.

Note that the term used for an engine stage is called *Handler level*, and a change of parameter for a Batch task is called *Group level*.

Engine Levels

The following figure illustrates the relationship between the engine's Handler levels and the various Operation repositories you can define in a task.

Task Execution Repository

Level	Event	Details	Scope	Prop	Enable	Oper
Task	Prefix Suffix					
Group	Prefix Suffix					
Record	Prefix Main Suffix					
Control	Prefix Suffix Verification Change					

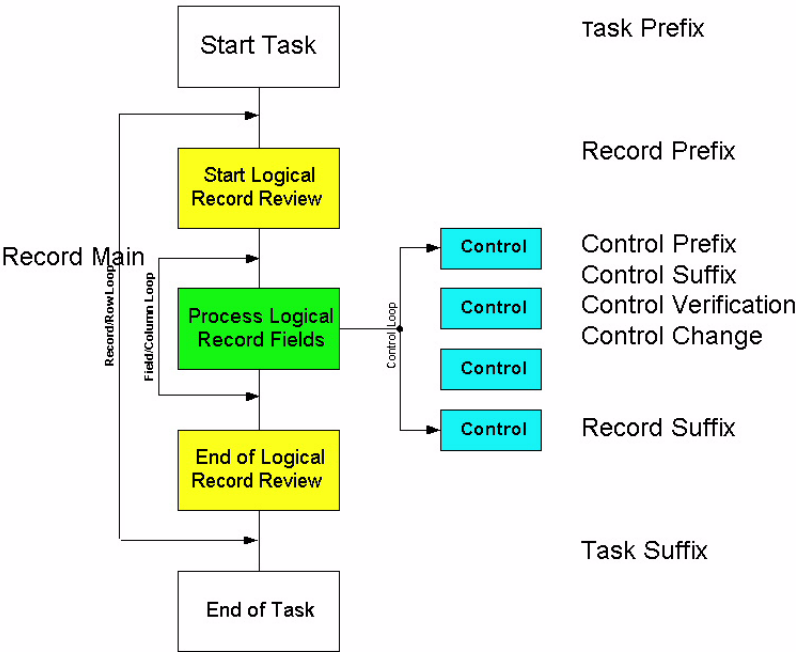


Figure 5-1 A Model of the eDeveloper Application Engine

Operation Repositories

Figure 5-1 displays the Level repository and the Task Execution repository. The Level repository organizes the access to the various Operation level tables.

Each handler row represents an operation level (Task, Group, Record, Control, or a user-defined handler). The Event column display the handlers available for a specific Operation level. For example, the Control level has four handlers - Prefix, Suffix, Verification, and Change. The fields at the intersections of rows and columns identify the various Operation repositories. The name of each Operation repository corresponds to its source row and column name, for example Task Prefix, Record Main, Control Prefix, Control Suffix, or Task Suffix.

After selecting the required Operation level and handler, you can then enter the required operations for your dataview.

The Main handler is relevant only for the record level, and therefore the only Operation repositories with the identifier "Main" are the Record Main Operation repositories.

Below is a brief listing of the various Operation repositories, presented in the sequence in which they are executed by their corresponding engine levels.

Task Prefix

In the Task Prefix Operation repository place the operations that the engine executes at the beginning of the task. The operations stored in the Task Prefix Operation repositories are for the task's initialization procedures. Task Prefix operations are often used for printing report headers and initializing local fields.

Group <break variable> Prefix

This level applies to Batch tasks only.

In the Group Prefix place the operations that the engine executes after the break field value has changed. That is, at the beginning of the group of records with a new value in the break field. This level is typically used to print report group headers. There is no limit to the number of Group levels that can be defined.

Record Prefix

In the Record Prefix, you should place the operations that the engine executes for every record immediately after the record is read from disk and before interaction starts. The Record Prefix Operation repository stores operations that are used at the initialization stage of a record: procedural operations that need to be carried out on the record before the beginning of processing its parameters.

Record Main

In the Record Main place the operations that the engine executes for building the dataview. For more information, see the Task Dataview section of this chapter.

Control Prefix

In the Control Prefix, place the operations that the engine must execute before the insertion point is moved to a particular control. The Control Prefix Operation repository stores operations that are used at the initialization stage of a specific control type. A control handler can only come after a record handler.

Control Suffix

In the Control Suffix, place the operations that the engine executes at the end of the control. The Control Suffix Operation repository stores operations that are used for the control's exit procedure, that is, just before the engine leaves the control to reset, update, or to print a control value.

Control Verification

Control Verification operations are performed whenever the insertion point is taken away from the control and whenever the control is passed through in Fast mode, before the Control Suffix level.

Control Change

In the Control Change handler level place the operations that the engine executes when a control variable is changed. The Control Change is for Online task types only.

Record Suffix

In the Record Suffix place the operations that the engine executes:

- For each record of the dataview in batch tasks.
- For each record browsed and modified in online or browser tasks. This means that if the record was not changed, the Record Suffix operations will not be executed.

Group <break variable> Suffix

This section is relevant for Batch tasks only.

In the Group Suffix place the operations that the engine executes before processing the record with a break field value that has changed. The Group Suffix Operation repository also stores the operations that are used for the printing of report group footers.

Task Suffix

In the Task Suffix place the operations that the engine executes at the end of the task. The Task Suffix Operation repository stores operations that are used for the task's exit procedure; to update fields for a calling task, or to print report totals.

Handler Level

The handler level lets you define a flow of operations that can be executed whenever a specified event occurs.

For more information about user-defined handlers, see the following Event handling section.

Event Handling

eDeveloper lets you define the eDeveloper logic as a response to implicit and explicit events that may occur during the execution of a task.

The following are terms for the event-driven engine:

- Event - An event is a logical definition of an occurrence. An event can be handled by an event handler to perform a flow of operations.
- Trigger - An event can be assigned as a trigger of another user-defined event. When the triggering event is raised, it triggers the user-defined event.
- Handler - A set of operations designated to be performed when a specified event is raised.

Event Types

Events in eDeveloper are divided into two groups, Pre-defined events and User-defined events.

Pre-defined events are divided into the following categories:

- Internal eDeveloper events. These are:
 - Events that are defined by the eDeveloper Runtime operation.
 - Events that have handlers hard-coded into the eDeveloper engine. For example: Prefix, Main, or Suffix.
 - Events triggered with a mouse button. Mouse button triggers are onclick, ondblclick, onmouseover, and onmouseout.
- System events. These events are keystroke combinations.
- User-defined events are divided into the following categories:
 - Timer events - Events that occur at a pre-defined interval.
 - Expression Evaluated events - Events that are raised when an expression evaluates to True.

Interactive Task Event Handling

Interactive task event handling lets you handle pre-defined or user-defined events for Online or Browser task types.

The Control Operation Level

The Control Operation level lets you control the behavior of a specific control type at runtime. For example, you can set and display values, check input, and implement other operations.

You define the Control level in the Task Execution repository as shown below.

Task:2 - car reservations							
#	Level	Event	Details	Scope	Prop	Enable	Oper
2	Task	Suffix					0 ▲
3	Record	Prefix					0
4	Record	Main					0
5	Record	Suffix					0
6	Control	Prefix	Ctrl: ▶			Yes	0 ▼

Figure 5-2 Defining a Pre-defined Control Handler

The Control Operation level has the following fields:

Handler Level	Select the Control Operation level
Event	Prefix, Suffix, Verification, or Change
Details	Zoom to select the control from the main screen of this program to be assigned to this handler
Scope	Not accessible
Propagate	Not accessible
Enable	Yes enables the handler No disables the handler
Operation	The number of operations in the Control Operation repository

The Handler Operation Level

The Handler level lets you assign a handler for an event. The Handler level can be created between a task and a record, or within a Group or Control Level handler. The Handler level shares all of the fields of the Control level but lets you choose the required event type from the Event field as shown below.

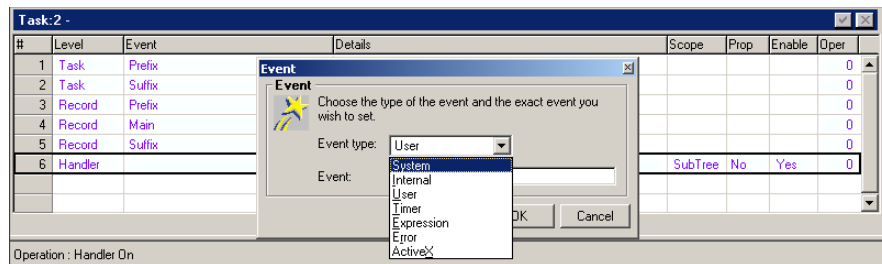


Figure 5-3 Choosing an Event Trigger Type from the Event List

You can access an event defined in the Users Event repository by selecting *Application* in the Events field. For more information on user-defined events, see the User-Defined Events section in this chapter.

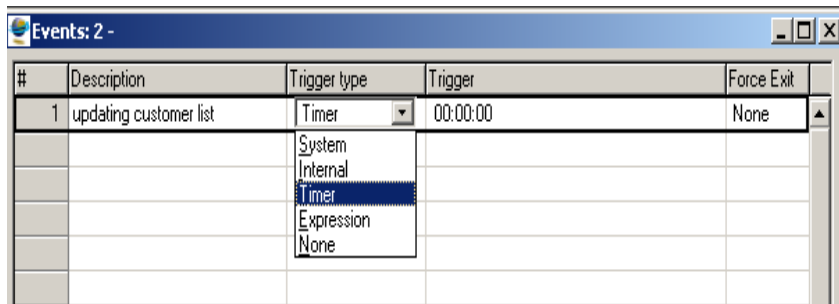


Figure 5-4 Selecting a Trigger Event for a User-Defined Control Handler

Batch Task Event Handling

Batch task processing is different from the interactive process. Unlike interactive tasks, where the runtime engine waits for input, Batch tasks process the records without waiting for input.

Batch tasks can poll pending events. You can define the Batch Task engine to poll events by a timer, by the number of records processed, or by both.

#	Batch Event Interval	Record Event Interval	Behavior
1	0	0	Never poll in that task
2	0	N	Poll every <i>N</i> records
3	M	0	Poll every <i>M</i> milliseconds
4	M	N	Poll every <i>M</i> milliseconds and <i>N</i> records

The Batch Event Interval field, in the Environment dialog, lets you set the timer for the event polling of Batch tasks. Entering a numeric value of 0 disables the Batch Event Interval. The polling of events will then be determined only by the numerical value of the Record Event Interval in the Task Control dialog.

The Record Event Interval, in the Task Control dialog, lets you define the number of records for the event polling of Batch tasks. Entering a numeric value of 0 disables the Record Event Interval. The polling of events will then be determined only by the numerical value set in the Batch Event Interval setting in the Environment dialog.

The Group Handler level is only relevant to Batch task types, and can be located only between a task and a record, as shown below.

#	Level	Event	Details	Scope	Prop	Enable	Oper
1	Task	Prefix					0
2	Task	Suffix					0
3	Group	Prefix	Var. [??] ??				0
4	Record	Prefix					0

Figure 5-5 The Group Handler Level for Batch Tasks

The Group Handler level lets you group records by a specific variable. For example, you can have a Group level for Customer Identification or Number of Purchases. When you change the variable for the Group level, you must reset the Prefix and Suffix handlers.

Handlers

The Task Break table of the Task Execution repository of previous versions has been renamed the Levels repository. The Levels repository displays a handler and its association with an existing event. The Handler table resides in a task, and displays the Handle flow, its fields and virtual variables.

In the old Task Break table each row defined all Flow operations for each Operation level (Task or Record). In the Levels repository, for eDeveloper Version 9, each line has a pulldown menu that displays its handlers: Prefix, Suffix, or Main, as shown below.

Task and Record level handlers are fixed. They are automatically generated for new tasks or records, and cannot be added or removed.

#	Level	Event	Details	Scope	Prop	Enable	Oper
1	Task	Prefix					0
2	Task	Suffix					0
3	Group	Prefix	Var: (???) ??				0
4	Record	Prefix					0
5	Record	Main					0
6	Record	Suffix					0
7	Handler		Cnt	SubTree	No	Yes	0
8	Handler	00:00:00		SubTree		Yes	0

Figure 5-6 Definition Level Repository

Group Level

The Group level is only relevant for Batch task types. The Group level can be located only between a task and a record. The Group level lets you group records by a specific variable. When you change the variable for the Group level, you must reset the Prefix and Suffix handlers. For more information, see Batch Task Event Handling on page 285.

Control Level

A Control level has been added to the task flow. The Prefix, Suffix, Verification, and Change handler types are associated with this new level. The Control level lets you define flow operations that can be executed when a specific control is entered or exited. Control handlers can be added only after the Record handler level.

Handler Level

The Handler level lets you associate a handler with an event.

Level Repository Fields

The Levels repository has the following fields:

- Level
 - Task
 - Group - For Batch task types only
 - Record
 - Control
 - Handler
- Details - display a variety of relevant control types to define the exact context of the handler.

For Group Handlers, the Details column displays the Var field, which lets you select a variable from the Variables list.

For the Control and Handler levels, the Details column displays the Ctrl field, which lets you select the control to which the handler associated.

You are required to select a control when working at the Control level. The Control Prefix, Suffix, Verification, and Change cannot be defined for an unspecified control.

You are not required to select a control when working at the Handler level. Selecting *None* in the Details property enables the handler to execute the linked event for all control types in a task.

- **Handler Type** - For error handlers, the Detail column displays the Dir field, which lets you define the engine directive that can be performed upon completion of the handler and the MSG field that defines whether eDeveloper should display the default error message. You can define an error handler for the following levels:
 - Task - Prefix and Suffix
 - Group - Prefix and Suffix
 - Control - Prefix, Suffix, Verification, and Change
 - Handler - System, Internal, User, Expression, Timer, and Error
- **Scope** - lets you determine when a handler will execute an event. The scope of the task may be:
 - Task - the handler executes the event in that task.
 - Subtree - the handler executes the event in that task and all of its subtasks.
 - Global- this is relevant only for main program handlers that are part of a component application. When set to Global, the handler is available to the host application.
- **Propagate** - you can propagate an event to an event handler defined in a higher level. In other words, once this event has been handled, eDeveloper passes the same event to the previous program, the one before it, and so on, looking for a handler for this event.
- **Enable** - enables or disables a handler.
- **Operations** - the number of operations listed in flow section.

Event Handler Behavior

- Every time an event is raised, it enters an event queue of the running context.
- eDeveloper polls the event queue for pending events at the following timings:
 - Online and Browser task – Every time the task is idle.

- Batch task – Every passed time interval according to the Batch Event Interval environment setting, or every number of processed records according to the Record Event Interval in the Task Control dialog.
- For each event that is polled from the event queue eDeveloper searches for a handler that is defined for the event. Whenever a handler for this event is found eDeveloper executes the handler.
- Upon completion of the handler, eDeveloper looks for the next matching handler only if the propagate property of the last executed handler is set to Yes or evaluated to True. Note that when the Propagate property is set, eDeveloper looks for the next handler set for the same event as the previous handler. In the case when Event A triggered Event B and the handler on Event B was found and executed, the propagate command is done only for Event B and not for Event A.
- eDeveloper looks for the matching handlers through the entire handler list from the current task up to the main program and from the bottom of each handler list to its top. This means that if there are two handlers of the same event in the same task the bottom handler is executed first.
- User events cannot be propagated.

Importing From Previous Versions

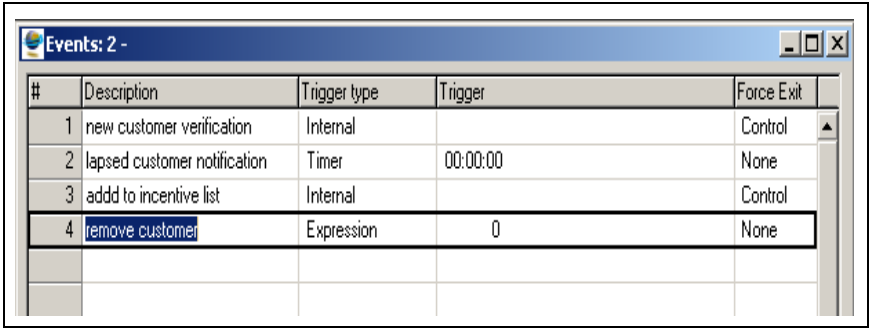
eDeveloper will make the following modifications to applications imported from previous versions:

- In general, a previous version event is created as a handler in the relative task. In this handler a Call operation is created for the specified program or subtask.
- Hot-Key events of previous versions
Such events are converted to a system event handler of the same key combination.
If an expression was set for this event, it is used as the enabling condition of the handler.
If an elapsed time value was given along with the hot key, the event is discarded and will not be part of the task. The discarded event information is shown in the log file.

- Action events of previous versions
These events are converted to an internal event handler of the same eDeveloper action.
If an expression was set for this event is used as the enabling condition of the handler.
If an elapsed time value was given along with the action this event is discarded and will not be part of the task. The discarded event information is shown in the log file.
- Elapsed events of previous versions
These events are converted to a handler of a timer event of the same time interval.
If an expression was set for this event it is used as the enabling condition of the handler.
- Events of previous versions with just an expression – If you create a program in Magic V8.xx with a single event based on an expression and import it into Magic V9.4, the time interval is '00:00:05'.
- Application events – Unlike previous eDeveloper versions, eDeveloper Version 9 does not have an Application Events repository. Handlers and events defined in the main program are treated as global for the entire application. Importing a previous version application event is converted to a corresponding handler in the main program.

User-Defined Events

You can create a user-defined event in the Users Event repository (**CTRL+K**), as shown below.



#	Description	Trigger type	Trigger	Force Exit
1	new customer verification	Internal		Control
2	lapsed customer notification	Timer	00:00:00	None
3	add to incentive list	Internal		Control
4	remove customer	Expression	0	None

Figure 5-7 Defining an Event in the User Event Repository

The Users Event repository lets you create a user-defined event and its associated triggers and levels at the application level.

For example, you can create an event called Create New Customer, which is triggered by clicking the **CTRL+F** key combination. In the Levels repository of the Task Execution repository, you create a new Handler level entry with a list of operation sequences. The Create New Customer Operation sequence is executed whenever an end-user presses **CTRL+F** in runtime.

User events are created at the task level, and are common to all the subtasks of the task to which they are defined. User events created in the root task of the Main Program are common to all the tasks in the application. If a handler does not exist for a user event, no default is available and the event is lost.

You can define a handler for the user event by creating an event Handler in the Levels repository of the Task Execution repository as shown in Figure 5-3.

The Users Event repository has the following fields:

#

An automatically generated entry number

Description

The event name

Trigger Type

System - Various keystroke combinations.

Internal - Events that have handlers hard-coded into the eDeveloper engine.

Timer - Events that occur at a pre-defined time interval.

Expression - Events that occur when an expression evaluates to True.

None - The user-defined event can be raised only by a Raise Event operation.

Trigger

The actual value that triggers the event. If None is set as a trigger, then the user-defined event can be raised only by a Raise Event operation. Please refer to Raise Event operation in Chapter 7, Operations.

Force Exit

Specifies from which level the handler should exit before performing its own sequence of operations.

- None – The handler will not commit a force exit, but performs its flow of operations within the edit mode of the control from which the event was raised.
- Control – The eDeveloper engine will first exit from the Control level, and perform the Control verification and Control suffix operations. Only after these operations have been executed will the eDeveloper engine execute the handler operations of a particular event.

- Record – The eDeveloper engine first executes the Control verification, Control Suffix, and the Record Level (performing Record Main and Record Suffix operations according to the Force Record Suffix property). Only after these operations are executed will the eDeveloper engine execute the handler operations of a particular event. The record is written to the database only after the execution of the handler.

Public Name

For the Main Program only. Defines the public name of the event by which it will be called by an Internet requester or a Call Remote operation. The public name must be unique within the application file.

Expose

For the Main Program only. When you select the Expose check box, programs from other components can call the Main Program event in the host application by using the Raise Public Event operation. When Expose is not selected, the event cannot be called by other loaded applications. You are also required to provide a public name for the user event because eDeveloper will use the public name of the user event to search for a corresponding handler.

Information about the Engine

The 14 eDeveloper Operations

You can find complete and detailed explanations of all the operations in Chapter 7, Operations The table below lists the operations and gives a short explanation of the purpose of each.

Operation Number	Operation Name	Operation Function
1	Select	Automatically defines a field as a variable available in a task. Also used to restrict the range of Main table records, to locate the first record to be processed, to assign initial values to computed variables, and to control insertion point movement. The Init expression of a Select Operation works non-procedurally.
2	Verify	Displays an error or warning message under a defined condition. Usually used for input validation.
3	Link	<p>Begins a relational (one-to-one) link between the Main table and another database table.</p> <p>Link Join establishes a many-to-one dataview relation between the Main table and linked join tables, that will be implemented by the underlying RDBMS. The operation will generate a SQL inner join statement among all the participating tables.</p>

Operation Number	Operation Name	Operation Function
4	End Link	Used to tell eDeveloper to stop selecting variables from the linked table. A Link operation must precede the Select operations.
5	Block	<p>Delimits the beginning of a logical block of operations. The purpose of a block is to enable you to put a condition on the Block operation that will control the execution of every operation inside the block.</p> <p>The If-Else Block lets you set an if-else condition in a single Block operation. If the first condition proves false, the engine will go to the Else option. You can have multiple Else options defined within one Block operation.</p> <p>The Block Loop operation instructs the eDeveloper engine to repeatedly execute the operation within the block.</p>
6	End Block	Ends the logical block of operations started by the preceding Block operation.

Operation Number	Operation Name	Operation Function
7	Call	<p>There are eight kinds of Call operations used to execute different types of subroutines:</p> <p>Call Task to call subtasks</p> <p>Call Program to call eDeveloper programs</p> <p>Call Exp to call Expressions</p> <p>Call Public to call a public program</p> <p>Call UDP to call user-defined procedures</p> <p>Call COM to call a COM object</p> <p>Call Remote to call programs from the eDeveloper Broker to a remote terminal</p> <p>Call Web Service (Web S) to call a Web service by using a SOAP protocol</p>
8	Evaluate	Executes action functions. Refer to the Evaluate section of Chapter 7, Operations Used to execute the action functions while evaluating the expression that includes them.
9	Update	Updates a variable with the evaluated result of an expression.
10	Output	Output Form writes a record to a text file on an output device. Used for printing reports or exporting data to text files.

Operation Number	Operation Name	Operation Function
11	Input Form	Reads text from an input file or device into the variables defined in a form. Used for text import programs.
12	Browse	Invokes an eDeveloper viewer or editor, with a text file.
13	Exit	Executes an operating system program from within eDeveloper. When the external program terminates, control may automatically return to eDeveloper, depending on the setting of the Wait field.
14	Raise Event	Lets you raise events during the eDeveloper engine flow. The Raise Event command is handled in the same way as System events.

Non-Procedural Operations

All operations, with two exceptions, are procedural. That is, they are executed when the engine reaches them during the sequential scanning of the Operation repositories. However, Select operations with Init expressions and Link operations with Link expressions work non-procedurally; they can be placed only in the Record Main Operation repositories, because they are used to build up the task's dataview.

The evaluation and assignment of the Init expression to a selected variable is activated automatically whenever the value of a variable in the Init expression changes. The evaluation of the Link expression and the consequent execution of the link is activated automatically whenever there is a change in the value of a variable in the expression.

Non-procedural recomputation is similar to the recomputation in a spreadsheet cell.

Guidelines to the use of the non-procedural features in operations are found in Chapter 7, Operations. There are other non-procedural mechanisms available, and they are explained throughout this manual.

The Task Dataview

Each eDeveloper task works with a set of records and their variables that you select from the application database. These variables, whether selected from the Main table or from a linked table, are the task's real variables. In addition, you may define computed variables that exist for the duration of the task's execution. These are the task's virtual variables or parameters. Real and Virtual variables, and passing parameters together constitute the task's *logical record*. The set of logical records selected from the Main table and linked tables in accordance with the task's range rules constitutes the task's *dataview*.

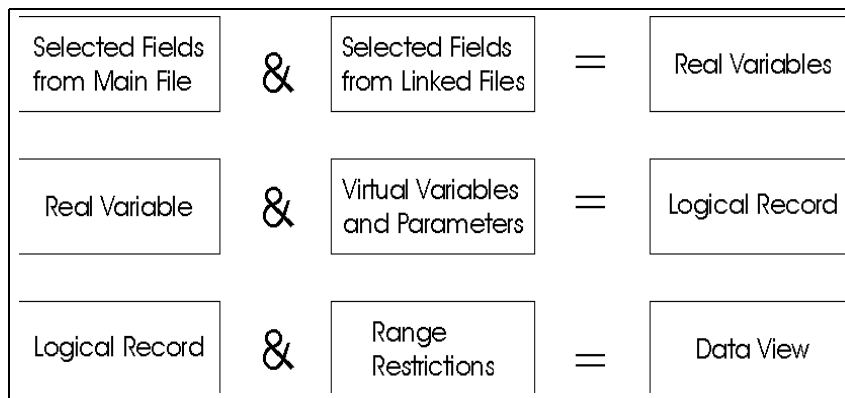


Figure 5-8 The Construction of a Dataview

All of a task's variables, real, virtual, and passing parameters, are included in the task's variable list. Each variable in the list is assigned a letter code. Whenever you want to refer to a variable in a task, you use this letter code. A subtask's variable list includes all the variables selected by its parent.

How eDeveloper Prepares the Dataview

When eDeveloper starts a task, the engine identifies which records will be included in the Task Dataview before executing operations in the Task Prefix Operation repositories. It does this as follows:

1. It examines the Record Main operations, looking for Select Real operations for variables from main and linked tables, and Select Virtual operations and parameters, and includes all these variables in the Logical Record/Row.
2. It evaluates the Task Range expression and any Range lower and upper expressions in their order of appearance in the Record Main. These Range criteria limit the records that can be included in the dataview. If you make Range active during runtime, the end-user can further limit the range of records included in the dataview. For a dataview created from an SQL table, eDeveloper also evaluates and performs the eDeveloper and the DB added where clauses.
3. If you have entries in the Sort repository, the engine sorts the Main table according to the specified index segments. If you make the Sort function active during runtime, end-users can perform their own sorts over the dataview record. Only records that comply with the range criteria are sorted.

Note: The eDeveloper task may be based on a SQL statement defined in the eDeveloper SQL Command task object. In this case, the dataview is prepared by the SQL database to which the SQL statement is directed and all the standard eDeveloper dataview preparation logic is bypassed. However, eDeveloper maintains normal operation in the stages after the dataview preparation stage. The variables used for such a SQL-based dataview are usually virtual variables, because the dataview is not based on the usual structure that eDeveloper recognizes a table definition in the Table repository.

Dataview Tuning

The Range expression in the Range/Locate dialog box provides a more flexible way to define the record set in the Task Dataview. When the Range expression is not a lower/upper value pair, it may contain non-contiguous ranges and set other complex conditions based on any of the logical record's variables. The

Task Range Order field allows you to set the actual sequence, ascending or descending, in which records will be displayed and processed. If both the Task Range property and Select Range lower/upper expressions are specified, the engine will use them all (in an “AND” relation) to determine the set of records for the dataview. Other range fields are described below.

Field models have a Range field in the Field Model Properties sheet as explained in Chapter 3, Models

The Column list lets you enter a range value for a column entry for a table as explained in Chapter 4, Tables

The SQL Where Clause determines a range based on SQL syntax using eDeveloper variables as explained in Chapter 6, Programs

The Record Dataview Instance

When the engine processes a particular logical record from the task dataview, that instance of the dataview includes values for the variables in the Variable list. The term *logical record* is used to distinguish this instance. A logical record contains one set of values including values from any linked tables, from the overall task dataview.

The Effect of Modes of Operation on Task Flow

The behavior of eDeveloper’s application engine depends both on whether the task is Browser, Online or Batch, and also on its Mode of Operation. The modes of operation pertain to the task’s Main table only. There are four basic modes of operation, shown in the table below, and they tell the engine which data manipulation operations are allowed on the data.

Mode of Operation	allows the end-user to...
Create	Create new records.
Modify	Edit existing records.

Mode of Operation	allows the end-user to...
Query	Scan through records, without allowing any update to the records. In Online, user input in this mode is used by an automatic locate function to locate records according to received keyboard input. In Batch, the task tables are opened in Read Only mode, which does not allow any update to the scanned records.
Delete	Delete records. Delete mode is active only in the Record Suffix Handler level of the Batch task.

When you define a task, you specify its initial Mode of Operation in the Task Control dialog. In the Task Control dialog, you can specify other properties regulating the opening of certain interactive windows for the end-user.

In an Online task, the end-user can change the initial Mode of Operation by selecting at runtime those modes you have allowed in the Task Control dialog. In a Batch task, the initial Mode of Operation cannot be changed during task execution.

The engine scans and executes operations specified in the Operation repositories, regardless of the Mode of Operation. You can restrict the execution of an operation based on the actual Mode of Operation during execution by using an expression containing the Stat function in the Cnd (Condition) column. Use of the Stat function is explained in detail in the Controlling the Execution of an Operation section in this chapter.

Modes of operation do not affect creation or modification of any database tables linked to the Main table. The type of Link operation used to link the table determines its creation and modification of records. Refer to the Link operation in Chapter 7, Operations

Automatic Switch from Modify to Create Mode

When the engine starts to execute an online or browser task whose initial Mode of Operation is set to Modify, if the specified Main table does not exist, or if the specified range is an empty one, the Mode of Operation is switched automatically to Create. This saves you having to make a special case of the first run of a program that both creates and updates a table.

However, an unexpected switch from Modify to Create Mode of Operation may occur if there is an irregular situation in the database table used as the Main table. For example, if the table is suddenly not present in the specified directory, or if its path was changed outside eDeveloper without a corresponding change to the table definition.

User-Initiated Temporary Switch from Modify Mode to Create Mode

While in Modify mode, the end-user can edit existing records in the task's dataview. The user may also temporarily switch to Create mode while maintaining the information displayed on the task's form. This switch to Create mode opens an empty record to be completed by the end-user according to the task's regular input design. After entering the new record with data and moving to a different record (thereby accepting the record), the task mode switches back to Modify, and, if the Reposition After Modify flag in the Environment dialog is set to **Yes**, the new record is placed according to the task's sort order. The temporary switch from Modify to Create occurs when either:

- The user creates a line (F4) while on a dataview record. The new record is then opened after the form position of the original record, or
- The user goes to the next line (↓) while on the last record in the dataview. The new record is opened as the last record on the form.

The Reposition After Modify setting is relevant only for online tasks, and does not apply to batch or browser tasks.

The Meaning of Query Mode

Engine execution flow is similar for a task in Query or Modify mode, with the following unique behavior for Query mode:

- For Batch tasks, execution flow is identical for Query and Modify modes with the exception of database update. In Modify mode, every record is read, processed, and rewritten to the database. In Query mode, all the database tables involved in the task are opened in Read Only mode and only read operations are performed on the data. Query mode should be used for output-only purposes because it is faster than Modify mode.
- For Online tasks in Query mode, all updates or deletes to records and variables are blocked, regardless of their source, whether user input or procedural task operations. However, in Query mode of an online task, a special accelerated Locate facility is available to the user. For this facility, user input is used to incrementally search the database table for the value that the user has input into the fields on the task's form. This Locate allows high-powered browsing through data without damaging its contents. The Locate facility can be disabled by the developer.

End-User Screen Interaction

This section concerns Online tasks only, and only the execution of those Record Main operations that specifically deal with end-user screen interaction.

The engine allows the end-user to move the insertion point between parkable variables in the same order as you defined their Select operations in the Record Main Operation repositories. Since the position of the variable on the screen is not relevant to the moving path of the insertion point, be sure to plan the sequence of insertion point movements required on the screen when you insert Select operations in the Operation repositories.

You control which operations the engine executes when the end-user moves the insertion point from variable to variable and from record to record, as follows:

1. By inserting the operations in the right place in the Record Main Operation repositories; that is, between the Select operations that

define variables to be displayed on the screen where the insertion point will park. See the Insertion Point Park section below.

2. By setting the Flow fields of an operation. These define how the engine scans and executes operations in the Operation repositories, and corresponding to which direction of the insertion point movement each operation must be executed. See the Flow Column section.

Insertion Point Park

An *insertion point park* is any control displayed on the screen where the insertion point is allowed to stop. A control can be an insertion point park only if it meets all of the following conditions:

1. The Select operation for this control is in the Record Main Operation repositories for the current task and not in the Record Main Operation repositories of a parent task.
2. The Cnd (condition) expression of the Select operation evaluates to True.

Note: If the task does not contain at least one parkable control, it will terminate immediately at runtime and it will display an error message.

Engine Execution Rules

This section explains how the engine executes each level of the task cycle.

Task Cycle Levels

The basic application engine cycle is shown below. Each box in the following figure represents a specific task Handler level, and its related Operation repository is depicted within the Level box.

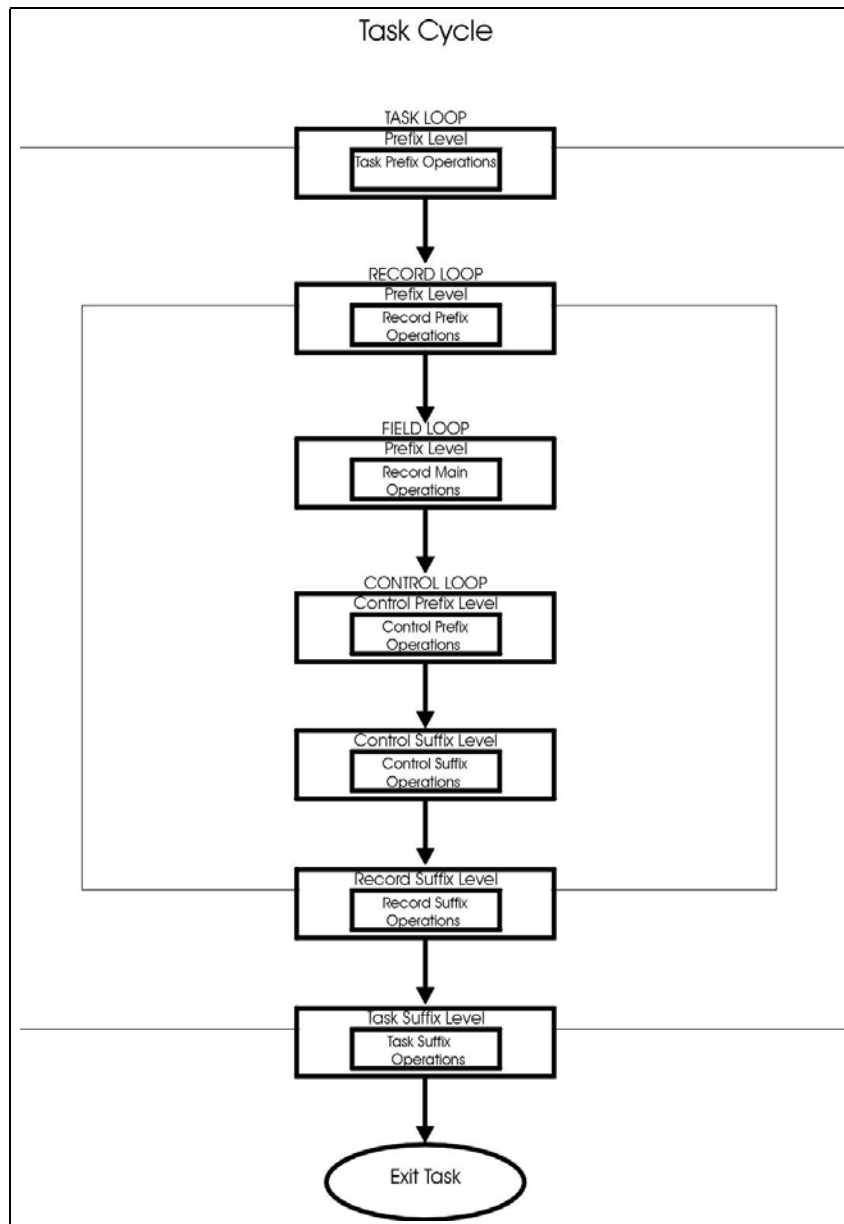


Figure 5-9 Task Cycle

The operations in each Operation repository are only part of all the activities performed by the engine at the related Handler level. In addition, the engine performs database management and other activities behind the scenes.

The two loops nested within the engine's Task cycle are:

the *Record/Row* Loop, whose levels are repeated for each record.

the *Control Loop*, which implements end-user interaction with the displayed record. The Control Loop exists for online and browser tasks. The Control Loop is described in the End-User Screen Interaction section of this chapter. The following descriptions tell you under which conditions the engine executes each level and what steps it performs.

Task Cycle

The engine executes the following steps once per task.

Task Prefix Level

1. If the task is called by another task to accept any passing parameters.
2. Establish the task dataview, that is, which records to use and how they shall be sorted, refer to the How eDeveloper Prepares the Dataview section.
3. Execute all the operations of the Task Prefix Operation repositories in the order in which they appear.
4. If this is an Online task in Modify (or Query) mode, compute Init and Link expressions to determine how many records can fit on the screen, according to the definition of the form and how many records it can display. For more information, refer to Chapter 7, Operations
5. In the Browser task, the Chunk Size Expression property in the Task property dialog can determine the number of records that can be fetched.
6. Execute the Record/Row Loop, as explained below.

Task Suffix Level

1. At the end of Record/Row Loop, execute all the operations of the Task Suffix Operation repository in the order in which they appear.
2. Exit the Task.

The Record/Row Loop

The engine executes the following steps once for every record the end-user browses in online and browser tasks and once for every record in the dataview, starting from the first record, in Batch tasks.

Record Initialization Level

1. Fetch the record into memory, so that its field values are available for processing.
2. Evaluate the Init expressions of the Select Virtual operations, from Record Main tables, and initialize the related variables.
3. If the Evaluate Task Condition setting is set to Before, see Task Control properties, evaluate the End Task expression and exit if True.
4. Execute the Record Prefix operations.
5. If the task is in Create mode, evaluate the Init expressions of the Select Fields operations, from the Record Main tables, and initialize the related variables.
6. Compute all Links for the current record.
7. If this is an Online task, step through the operations of the Record Main Operation repositories by executing the Fields Loop.

Record Termination Level

1. Execute Record Suffix Operation repositories.

If this is a Batch task, execute all the operations in the Record Suffix Operation repositories.

If this is an online or browser task, execute the Record Suffix level only if the current logical record has been changed. eDeveloper considers the current record to be changed if any of the following happened:

- a) The end-user edited a field, either by typing a value into it.
 - b) An Update operation with the Undo property set to No.
 - c) An Update operation of any kind executed in a subtask of the current task modified a variable.
 - d) An Input Form operation inserted data from an input repository in a variable.
 - e) The Force Record Suffix field, set in the Task Property sheet, evaluates to True.
2. Performs record deletion when required.

Record/Row Deletion in Online and Browser Tasks

When a Delete Line (internal event) is raised either by selecting the Edit or Delete Line menu option, by clicking **F3** in an Online task, by explicitly raising the Delete Line event in online and browser task, or by making the Force Record/Row Delete condition True, the engine activates the following Delete procedure:

- a) If the record has not been modified, execute the Record Suffix only once, in the 'Delete' mode.
- b) If the record has been modified,
 - execute the Record Suffix in the Modify mode, in order to perform any update operation defined there and complete the processing logic; *and*
 - execute the Record Suffix a second time in the Delete mode, in

order to perform any delete-related operations, such as deleting chained records in one-to-many relations, or Update operations using the Incremental mode.

Record/Row Deletion in Batch Tasks

a) If the initial mode of the task is Delete, execute the Record Suffix once in the Delete mode.

b) If the Force Record/Row Delete becomes True,

- execute the Record Suffix in the 'Modify' mode, in order to perform any Update operation defined there and complete the processing logic;

- execute the Record Suffix a second time in the 'Delete' mode, in order to perform any delete-related operations.

For all of the above three situations, after the execution of the Record Suffix in the 'Delete' mode, mark the record as *deleted*.

3. If the Record Suffix has been executed, save all the changes to disk for Main table and linked tables.

If the record has not changed, it is not saved to disk.

4. If the Evaluate Task Condition property is set to Updating Record (see Task property dialog), evaluate the End Task expression and exit the Record/Row Loop if True.

The Control Level

The Control Loop is relevant only for Online or Browser tasks.

The Control Level handler is executed for the Control Prefix or the Control Suffix. The Control Level handlers let you set values, check input, display control values, and implement other operations for a specific control. For each eDeveloper Control type, you can create a specific Control handler.

Creating a Control Handler

Unlike Task and Record handlers, which exists for every task, the control handler must be created manually.

Each control handler should be set with the required event type of (such as Prefix or Suffix) and the name of the control on the form.

Note that eDeveloper generates a control name for every variable dropped on the form or placed by the Automatic Program Generator (APG).

- **Control Prefix**

The Control Prefix level is performed whenever the end-user parks on the control for which the control handler is set.

All the operations listed in the Control Prefix level are executed before the end-user actually executes the control.

- **Control Suffix**

The Control Suffix is performed whenever the insertion point is taken away from the control (loses focus). This may occur when the end user moves to a different control or exits the record or the task.

The Control Suffix can be implicitly executed when an event, that is defined to exit a control level, is raised and handled. These internal events, like View Refresh or a user-defined event, are set to exit the control or record level.

The following sequence occurs when an event exits the Control Loop:

1. Terminate Control Editing
2. Set the runtime value of variable
3. Re-compute according to the value of the new variable
4. Exit the Control Suffix

- **Control Verification**

Control Verification operations are performed whenever the insertion point is taken away from the control and whenever the control is passed through in Fast mode, before the Control Suffix level.

- Control Change

In the Control Change handler level place the operations that the engine executes when a control variable is changed. The Control Change is for Online task types only.

Group Levels

Group levels are additional levels of execution and they are available for Batch tasks only. You use them for presenting or processing data by groups - application-breaks - in reports.

You can define multiple Group levels, based on different fields.

For each application break, you may specify operations for either or both Group Prefix and Group Suffix.

The engine executes these operations whenever it detects a change in the field used for the definition of the application break, and before the processing of the current dataview record at the Record/Row stage. This allows for operations that must be performed at the end of a block of records, such as printing subtotals and totals.

After Group Suffix operations are executed for the previous dataview record, the engine executes Group Prefix operations for the current dataview record before it is processed. This allows the engine to perform any initialization-type operations, such as zeroing counters and printing headers, before processing begins on the block of dataview records containing the new value.

The Main table must be ordered, by Index or by Sort, according to the fields specified in the Group level. Place the Group levels in the Level Definition repository in the order of appearance of the index segment fields in the Index or the Sort Index declared for the task.

Main group levels appear first, followed by intermediate Group levels, and finally minor group levels. The Group level immediately before the Record level is the lowest, and that immediately after the Task level is the highest. If the order of levels in the tables is incorrect, the breaks will execute incorrectly.

A change in a higher Group level forces a change in all lower Group levels. The Prefixes are executed from high to low, and the Suffixes low to high.

How the Engine Executes Group Levels

When the execution of a Batch task starts, the Task Prefix is executed. Then the first dataview record causes the prefixes of all Group levels to be executed, from the highest to lowest, ending with the Group Prefix of the lowest level. Until a change is encountered, the Record Suffix operations are executed for each record.

A change condition may be encountered at any Group level. The first change causes all Group Suffixes, from the lowest level up to the changed level, to be executed for the block of records already processed. Next, the Prefixes of all Group levels are executed, from the changed level down to the lowest level for the block of records to be processed.

Then the next block of dataview records is processed at the Record Level until the next change, and so on until all records have been processed. Finally, when an end condition is encountered, either at the end of all dataview records or when the End Task expression becomes True (see the Task properties dialog in Chapter 6, Programs, for more information), it is treated as the equivalent of the highest Group level, causing only the Suffixes of all Group levels from the lowest to the highest levels to be executed for the block of records already processed. Only then is the Task Suffix executed.

Group Levels Example

In a Batch task that prints running totals from an Order Lines tables sorted by Customer (high), Order Number (intermediate), and Item Number (low), the lowest Group level is based on changes in the Item Number, the second Group level is based on changes in the Order Number, and the third Group level is based on changes in the Customer Number.

During processing of the first record, the Group Prefixes of all levels are executed, from the highest to the lowest. When the first item in the first

customer's first order has been processed (e.g., counted or printed), the Suffix of Item Number (the first Group level) is executed.

After all the first order's item lines have been processed, the Suffix of the intermediate Group level, by order number, is executed, to print the total values for the first order. Group levels for both Item Number and Order Number are executed for all this customer's orders, and then the Group Prefix for the next customer is executed.

If, during the scanning of the record, a higher-level Group field changes value, all the Group Suffixes up to that level will be executed before the next record is processed.

After all of the records are processed the Task Suffix executes. In this example the total values for all customers will be printed.

Record/Row Loop Flowcharts

The following pages show diagrams that describe the engine flow of the Record/Row Loop for online, browser, and batch tasks. They can help you to find your way when you want to know exactly what the engine is doing while executing your task.

Record/Row Loop in Online Tasks

Figure 5-10 shows the execution flow of the application engine while performing the Record/Row Loop stage of an online and browser task. The details shown are not of direct concern in task development activity but they can help you to decide in which tables to insert a specific operation, or to find out why your task doesn't behave as expected

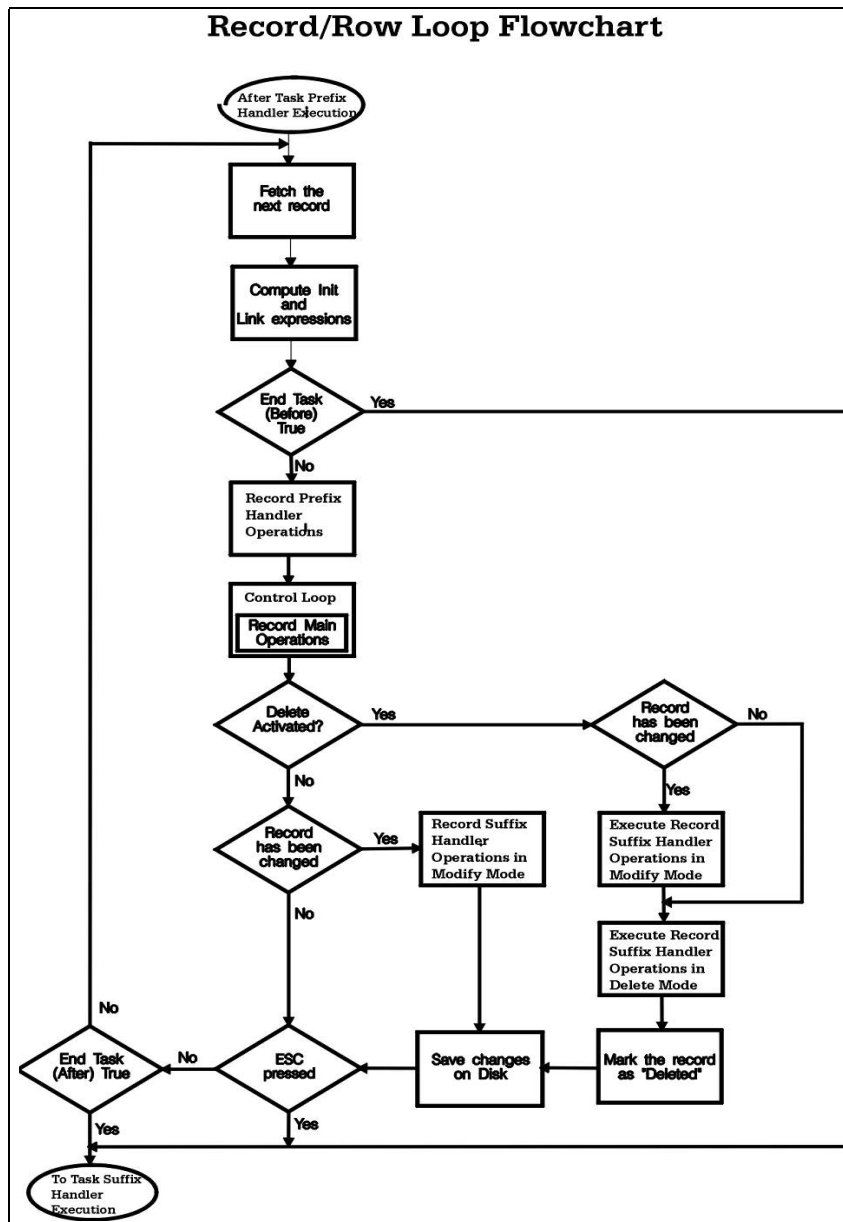


Figure 5-10 Record/Row Loop in Online Tasks

Record/Row Loop in Batch Tasks

Figure 5-11 shows the execution flow of the application engine while performing the Record/Row Loop stage of a Batch task. The details shown can help you decide in which tables to insert a specific operation, or to find out why your task does not behave as expected.

Record/Row Loop in Batch Tasks

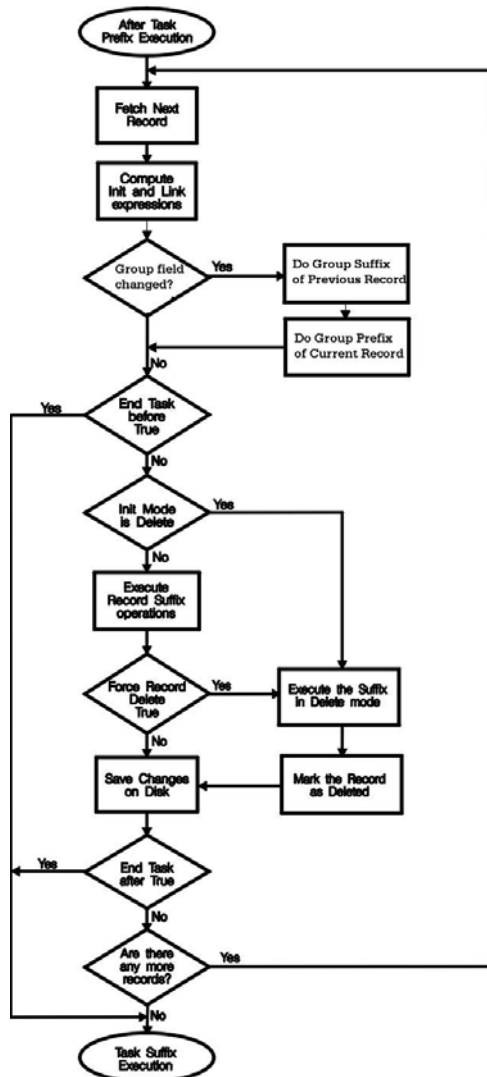


Figure 5-11 Record/Row Loop in Batch Tasks

Engine by Record Level

How the Engine Scans the Record Main Tables

The engine can scan the Record Main tables in two modes: Step mode and Fast mode.in online tasks.

Step Mode

This is the default scanning mode. The engine remains in Step mode as long as the end-user moves the insertion point from the current parkable control to the previous or next parkable control in the same record using the Next Field and Previous Field actions. Using the mouse to move the insertion point causes the engine to scan the Record Main tables in Fast mode.

While in Step mode, the engine mimics the movement of the insertion point on the screen between controls of the same record by scanning and executing all operations in the Record Main tables that meet the following three criteria:

1. The operation must be situated between the Select operations of the two parkable controls.
2. The Cnd expression must evaluate to True.
3. The first Flow field must have Interaction mode set to either Step or Combined, and its Flow direction setting (the second Flow field) must match the current direction of the insertion point movement. The Flow Interaction mode setting Combined includes both Step and Fast, and the Flow direction setting Combined includes both forward and backward insertion point movement.

If the user moves the insertion point backwards from Field B to Field A, the operations intervening between Select Real A (or Select Virtual A) and Select Real B (or Select Virtual B) are executed in reverse order. See Figure 5-12.

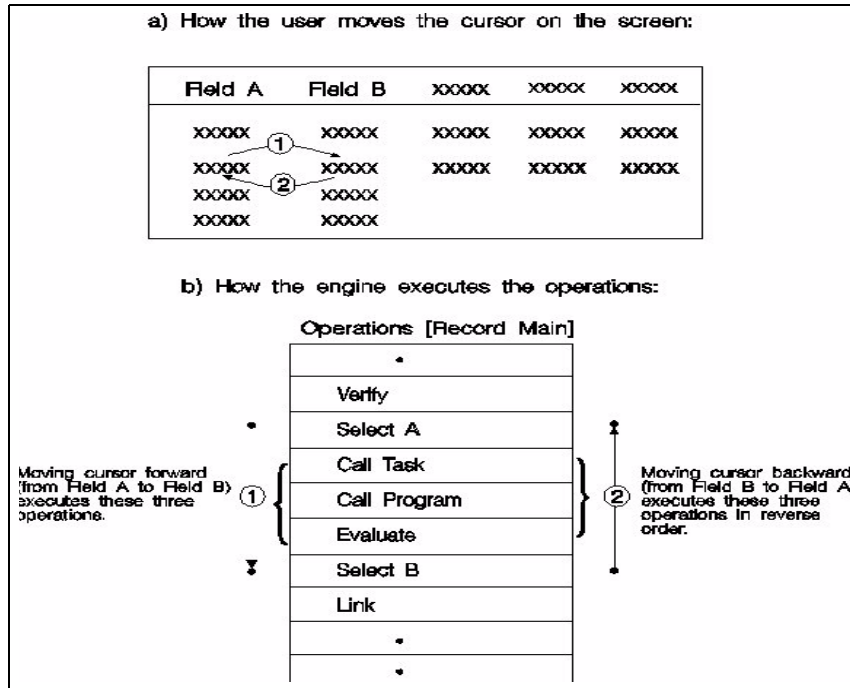


Figure 5-12 Scanning in Step Mode

By default, all procedural operations are defined as executable in Step mode only, and for insertion point movement in both directions. This is suitable for most situations and requires no programming effort. If you need finer control of execution flow, for example, if you want an operation to be executed only when the insertion point passes in a specific direction through a control, then use the second field of the Flow column of the operation to specify the direction you want.

For a more detailed explanation of this topic, refer to the section below on Controlling the Execution of Operation.

Fast Mode

Scanning in Fast mode means that the engine scans the Record Main tables, executes only those operations flagged for execution in Fast or Combined Interaction mode, and recomputes Init and Link expressions if needed.

The typical situation that causes the engine to switch automatically to the Fast mode of scanning is if, during interaction with the controls of one record, the end-user terminates editing the current record, for example, by moving the insertion point to a different record. This scenario is illustrated in Figure 5-13 below:

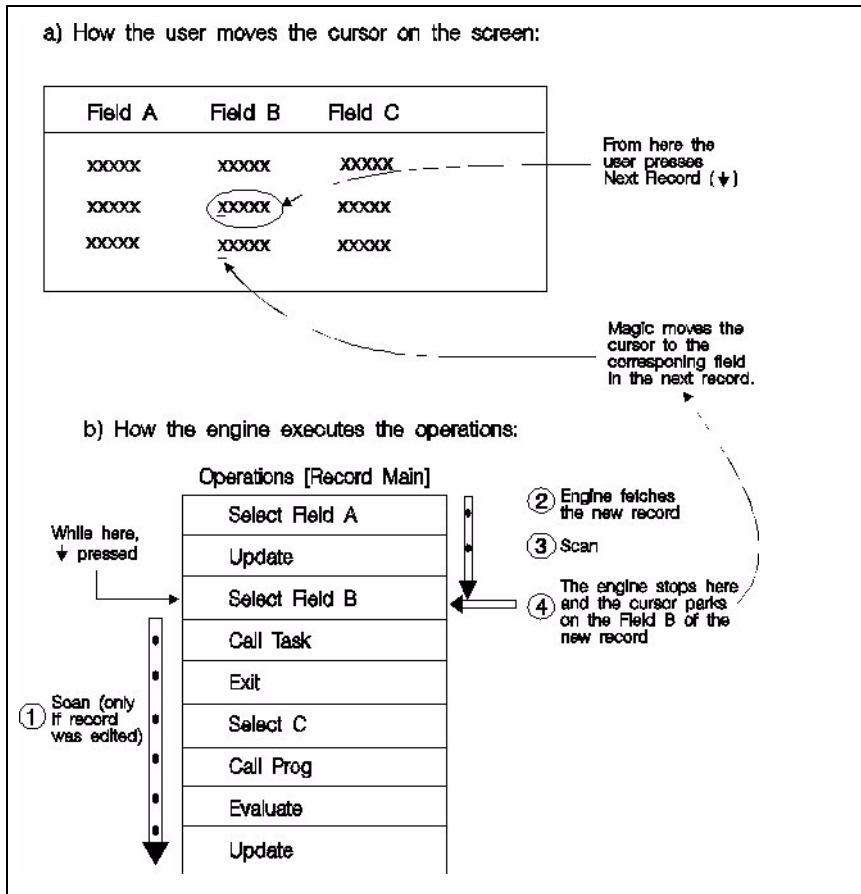


Figure 5-13 Scanning in Fast Mode

The engine executes the following sequence of actions:

1. If the current record has been edited, the engine scans the rest of operations in the Record Main tables in Fast mode. When it finishes scanning, the engine executes the Record Suffix.
2. The engine moves to the new pointed record.
3. The engine scans, in Fast mode, the Record Main for the new record, from its beginning, until it reaches the Select operation for the control where the insertion point has to park.
4. As soon as the insertion point parks, the engine returns to Step mode.

In addition to the case described above, the engine switches to Fast mode in the following situations:

1. When the end-user uses the mouse to move the insertion point within the current record.
2. When the end-user uses the End Row action to move the insertion point to the last control in the current record, the engine scans forward in Fast mode to the Select operation of the last parkable control in the record.
3. When the end-user uses the Begin Row action to move the insertion point back to the first control of the current record, the engine scans in Fast mode backward to the Select operation of the first parkable control of the record.
4. When the end-user moves the insertion point by using the mouse forward or backward to another parkable control in the current record, the engine scans in Fast mode forward or backward to the pointed parkable control.
5. When the end-user selects Edit/Delete Line (F3), the engine scans the rest of the Record tables in Fast mode.
6. When the end-user exits the task by pressing ESC, the engine scans the rest of the Record Main tables in Fast mode.

Controlling the Execution of an Operation

You can control whether or not any single procedural operation defined in an Operation repository executes. You cannot disable the execution of eDeveloper non-procedural operations; the re-computation of the Init expression in Select operations and of Link operations is considered to take precedence, in order to preserve data integrity.

The fields you use to control the execution of operations are set in the Cnd column of any Operation repository and in the Flow column for Record Main Operation repositories of Online tasks only, as shown below.

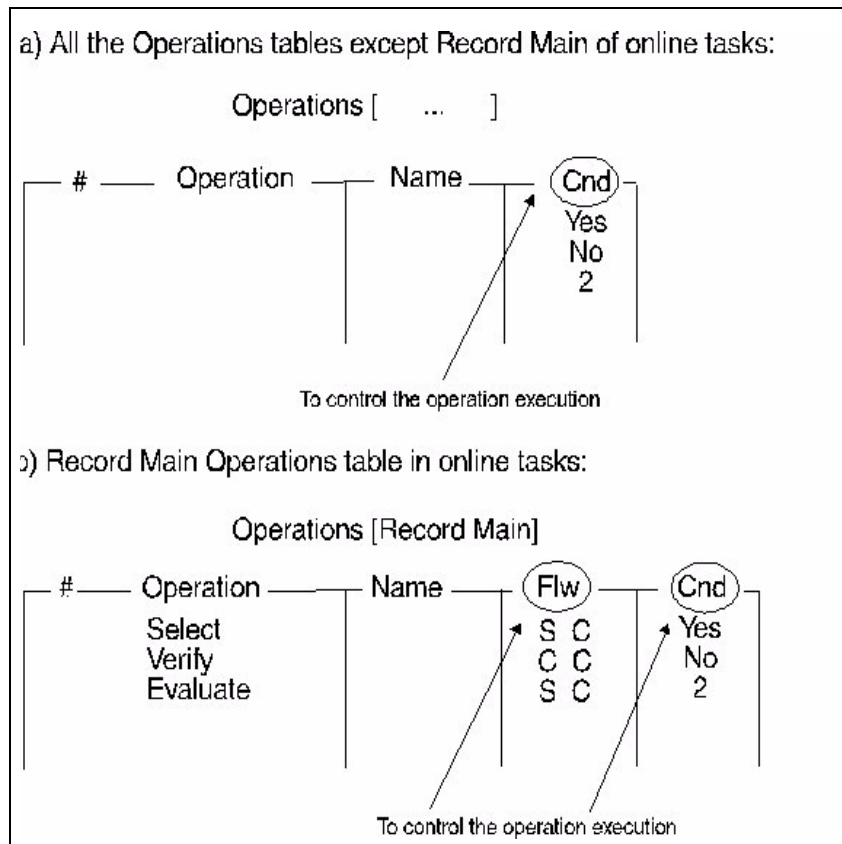


Figure 5-14 Row and Cnd Fields

When the engine scans Operation repositories to determine which operations will execute, it first checks the Cnd (condition) expression of an operation and then, for the Record Main repositories of Online tasks only, the two conditions specified in the Flow column.

The operation is executed only if all conditions are satisfied.

Cnd Expression

The last column of each operation is labeled Cnd (for Condition).

Depending on which operation you are defining, you can use the Cnd column to:

- Control the insertion point's ability to park on a control for Select operations in Online tasks.
- Change the behavior of the Link Validate operation when the link fails. For the other Link type operations, the Cnd column is irrelevant. Refer to the Link operation in Chapter 7, Operations.
- Condition the execution of all the other operations, except for End Link and End Block. End Link and End Block are not executable operations, and the Cnd column is not available for them.

The Cnd column can have the following values:

Yes - (default) - sets the condition to True and makes execution of the operation dependent on the Flow columns.

No - Select operation: prevents the insertion point from parking on the control.

Other operations: sets the condition to False and disables execution of the operation. Use this option for debugging purposes.

Expression number - refers to a logical expression in the Expressions Rules repository. If the expression evaluates to False, the engine does not execute the operation. If it evaluates to True, execution of the operation depends on the Flow columns.

For example, you can use the Cnd expression to condition an operation's execution to a specific Mode of Operation of the task. To do this, define an expression that includes the function Stat.

The syntax of the Stat function is:

Stat (generation, mode)

where

generation is one of the following numbers identifying the task that is to be tested:

0 indicates the current task

1 indicates the parent task

2 indicates the grand-parent task

and mode is a string of up to four of the following letters indicating the required Task mode to match against:

C means Create

M means Modify

Q means Query

D means Delete. The Delete status can be tested in the Record Suffix Level only.

The expression

Stat (0,'CQ'MODE)

will return a True value if the current task mode is Create or Query. If you use this expression as the Cnd expression in an operation, the engine will execute the operation only if the task currently running is in Create or Query mode.

Flow Column

The Flow column appears only in the Record Main Operation repository of Online tasks. The purpose of Flow is to allow you to make the execution of an operation dependent on how the end-user interacts with the task.

The Flow column is always empty for Link, End Link, and End Block operations. For all other operations that you define in the Record Main of Online tasks, the Flow column contains two fields: Interaction mode and Insertion Point direction.

Interaction Mode (*first Flow Column field*)

The Interaction mode is a one letter code which works according to the following rules:

When Flow Mode Code is...	the engine executes the operation if...
S for Step	The task is in Step mode
F for Fast	The task is in Fast mode
C for Combined	The task is in either Step or Fast mode
A for After	The end-user selects Zoom (F5) in the previous control
B for Before	The end-user selects Zoom (F5) in the next control

Step and Fast modes are explained in Section End-user Screen Interaction, above. The After and Before modes are called Zoom Interaction modes.

Zoom Interaction Modes

The zoom interaction mode is relevant for online tasks only. By setting the Interaction mode of an operation to A or B, the operation becomes zoomable, that is, its execution is conditioned to the end-user's selection of the Zoom option. If the end-user does not select zoom, the Zoomable Operation is not executed.

To implement this feature:

1. Designate a field to be the Zoom field and an operation to be the Zoomable Operation. The Zoom field is where the insertion point is to park when the zoom becomes enabled.
2. For the Select Field operation of the Zoom field, set the Zoom mode to A if you want the insertion point to move to the next parkable field

after the execution of the zoomable operation. Then place the zoomable operation immediately after the Select operation of the Zoom field.

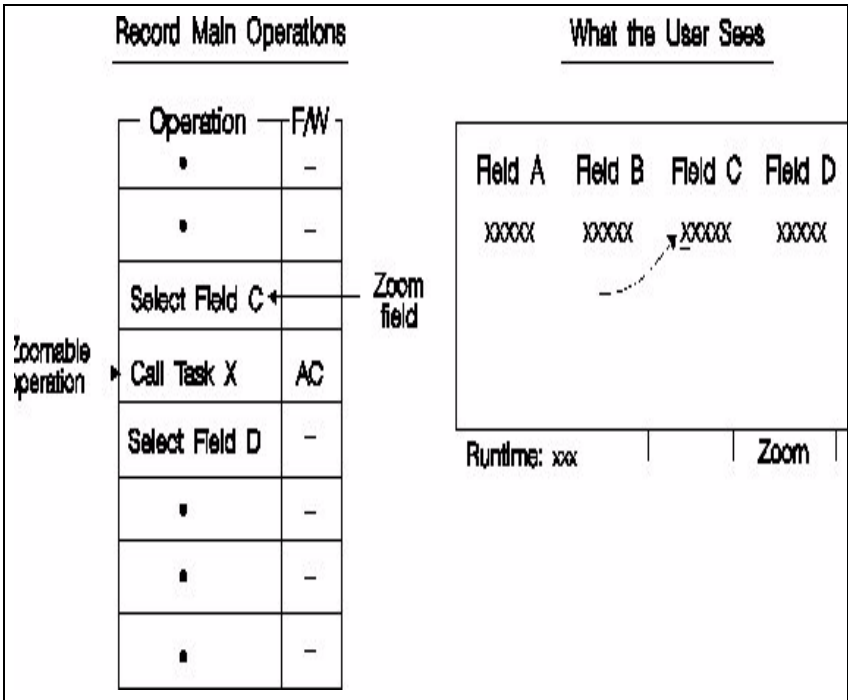


Figure 5-15 Zoom Mode 'After'

In the example shown in Figure 5-15, Field C is the Zoom field. At runtime, when the end-user moves the insertion point to Field C, the Zoom indicator appears on the message line to tell the user that zoom is active. If the end-user selects zoom, the engine executes Task X. When the called task ends, the insertion point moves to Field D.

3. Set the Zoom mode of an operation to B if you want the insertion point to remain on the Zoom field after the execution of the zoomable operation. In this case you have to place the zoomable operation

immediately before the Select operation of the Zoom field.

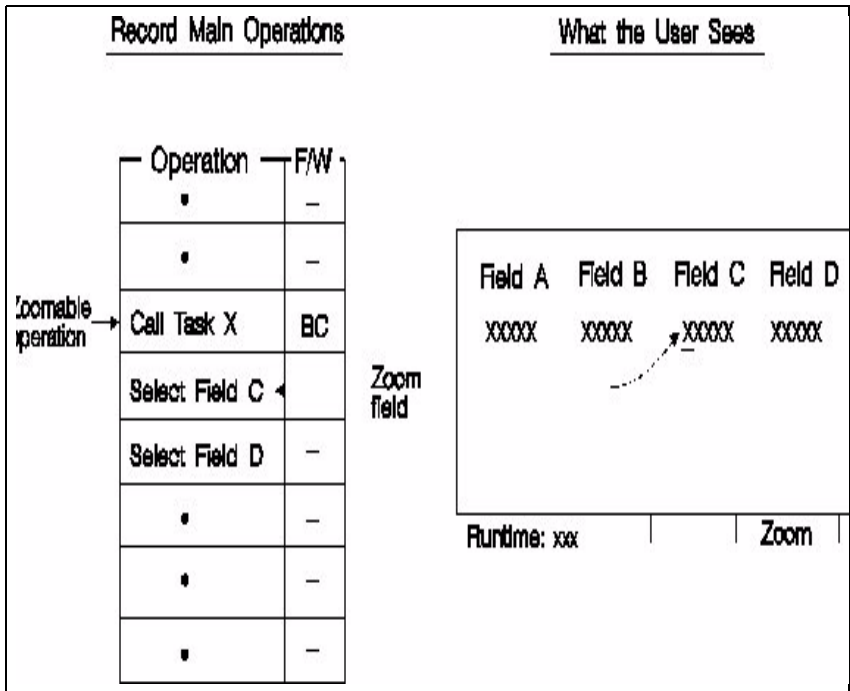


Figure 5-16 Zoom Mode 'Before'

In the example shown in Figure 5-16, Field C is the Zoom field. At runtime, when the end-user moves the insertion point to Field C, the Zoom indicator appears on the message line to tell the user that the zoom is active. If the end-user zooms from this field, the engine executes the task X. When the called task ends, the insertion point stays on Field C.

Most probably you will use the Zoom modes in conjunction with Call Task, Call Program, and Block operations, to implement Look-Up windows and pick lists. However you can use any procedural operation as the zoomable operation.

Insertion Point Direction (second Flow Column field)

The Insertion Point Direction field in the Flow column is a one letter code that works according to the following rules:

When Insertion Point Direction Code is...	The Engine Executes the Operation if the...
F for Forward	insertion point passes through it moving forward
B for Backward	insertion point passes through it moving backward
C for Combine	insertion point passes through it moving forward or backward

End-User Screen Interaction

This section concerns Online tasks only, and only the execution of those Record Main operations that specifically deal with end-user screen interaction: the engine's Field Loop.

The engine allows the end-user to move the insertion point between parkable fields in the same order as you defined the Select operations in the Record Main Operation repositories. Because the position of the field on the screen is not relevant to the moving path of the insertion point, be sure to plan the sequence of insertion point movements required on the screen when you insert Select operations in the Operation repositories.

You control which operations the engine executes when the end-user moves the insertion point from field to field and from record to record, as follows:

1. By inserting the operations in the right place in the Record Main Operation repositories, that is, between the Select operations that define fields to be displayed on the screen where the insertion point will park. For more information see the Insertion Point Park section.
2. By setting the Flow fields of an operation. These define how the engine scans and executes operations in the Operation repositories, and corresponding to which direction of the insertion point movement

each operation must be executed. For more information see the Flow Column section.

Insertion Point Park

An insertion point park is any control displayed on the screen where the insertion point is allowed to stop. A control can be an insertion point park only if it meets all of the following conditions:

1. The Select operation for this control is in the Record Main Operation repositories for the current task, and not in the Record Main Operation repositories of a parent task.
2. The Cnd (condition) expression of the Select operation evaluates to True.
3. The Flow interaction mode you specified in the first Flow field of the Select operation matches the task's current interaction mode.
4. The insertion point movement direction you specified in the second Flow field of the Select operation matches the current insertion point movement direction.

Note: If the task does not contain at least one parkable control, it will terminate immediately at runtime, with a suitable error message.

This chapter focuses on where you can create or modify eDeveloper programs and tasks.

In this chapter:

• Program Repository
• Local Variable Repository
• Expression Rules Repository
• Form Repository
• I/O File Repository
• DB Table Repository
• Sort Repository
• Event Repository
• Main Program

Program Repository

The Program repository contains an entry for each program within an application.

Properties of the Program Repository

Each entry in the Program repository has the following properties:

(for Program identifier)

This column contains an automatically generated sequential number used by eDeveloper as a program identifier. You cannot edit this column.

When you edit the Program repository by moving a program from one folder to another, or adding and deleting rows, eDeveloper automatically rennumbers the affected programs and updates their identifier numbers throughout the application. For example, it will renumber the menu entries that activate programs to reflect the modified numbers in the Program repository and maintain the association.

When you use the Program Identifier number inside an expression, you must qualify it explicitly by using the Prog literal. To do so, specify the program number *nn* as 'nn'Prog. For example, if you enter '3'Prog as a program expression in the Expression Rules repository, and later insert a new program in the Program repository ahead of the third program, thereby changing the old program entry from #3 to #4, eDeveloper will automatically change the '3'Prog to '4'Prog in the expression.

Note: When the cross-reference utility is activated, using the Prog literal will enable eDeveloper to locate an expression that refers to a program.

Name

A descriptive name

- The program name in this property is identical with the Task Name of the root task and it will appear as the label of the root task box in the Task navigator, and the Task name in the Task Properties dialog.

- Any change to this name in one of the three places is reflected immediately in the other two.

Public Name

You can define the public name of the program that is called by an Internet Requester, Call Remote operation, Browser client, or Call Public operation, the public name must be unique within the application file.

Last Update

This property is automatically filled with the date and time when eDeveloper saved the latest change made to any task in the program hierarchy.

Tasks

A task is the basic control object executed by the eDeveloper Application engine. If a task has one or more subtasks that are organized in a hierarchic structure, the top task is also known as a parent task, root task, or program. The Task navigator gives a visual representation of the related tasks of a program. If the current task is the root task, this is also the name of the program as it appears in the Program repository.

Menu Options for Tasks

The Task pulldown menu provides tool options for defining and modifying the various parts of a task. The Task pulldown menu options are:

- SQL Command - Lets you base the task's dataview on a SQL statement provided by the developer.
- Task Control - Lets you set the properties that control the runtime behavior of the task.
- Variables - Lets you access the local variable repository, where you can browse or edit the task's temporary Virtual and Parameter variables.
- Expression Rules - Lets you create or edit the task expressions.

- Forms - Lets you create or edit the task display and output forms.
- DB tables - Lets you set specific properties for the database tables used in the task.
- I/O Files - Lets you define any operating system IO device the task requires.
- Sort - Lets you specify a sort of the task dataview, to be executed as soon as the task starts running.
- User Events - Lets you define Task events, which are eDeveloper programs or tasks that can be executed at any moment during task execution.
- Range/Locate - Lets you define the range and locate functionality for any non-SQL eDeveloper application.
- SQL Where - Range and locate functionality for eDeveloper applications that use an SQL database.

Task Properties Dialog

Task Properties are defined in the Task Properties dialog, which opens automatically the first time you zoom into a new task. To once again open this dialog select Edit/Properties from the Task Execution window.

The Task Properties dialog contains three tabs:

- Properties Tab
- Advanced Tab
- Enhanced Tab

Properties Tab

Task Name

This is the name appearing in the Task box. If the current task is the root task, this is also the name of the program as it appears in the Program repository.

The Task Window title, displayed when the task is executed, is inherited from the task name; it can be overridden.

If you change the name in the Task Properties dialog, it will automatically be updated everywhere it occurs, including in the Form Repository entry and form window title. However, the Form Repository entry name will not be changed if, prior to the task name change, it differed from the task name.

Task Type: Online (default)

The Task type is either Browser, Online, or Batch.

Browser or online tasks do not always require updating data. If the task does not update data, there is no need to open a transaction. To open a browser or online task without a transaction, set the Transaction Mode to None.

For details about the differences between Browser, Online, and Batch tasks, refer to Chapter 5, Application Engine.

Initial Mode: Modify (default)

This property defines the mode of operation in which execution of the task starts. The options are: Modify, Create, Delete, Query, As Parent, Locate, Range, Key, Sort, Files, Options, and By Exp.

You can allow the end-user to change this mode during execution by setting the suitable *Allow* properties in the Task Control dialog, explained below, to Yes.

You can specify the mode of operation by assigning a specific mode in the Initial Mode property, or it can be assigned dynamically by selecting the *By Expression* option.

The modes that can be selected are shown in the table below.

Mode letter	Mode Name	Allows the end-user to...
M	Modify	change data in existing rows. In Batch, the row is read and written back to the database even if not modified.

Mode letter	Mode Name	Allows the end-user to...
C	Create	create new rows.
D	Delete	delete all the rows of the dataview. It is relevant only for Batch tasks.
Q	Query	scan through the rows and columns using directional keys or mouse, allowing for user input only if the Allow Locate in Query option control is set to Yes. In Batch, open tables in ReadOnly mode, regardless of DB Table Repository definitions, to perform read only operations.
P	As Parent	run a task or program whose mode is identical to that of its parent in runtime. If a program with the As Parent Task mode in runtime is not called by another program, the task mode defaults to Query.
L	Locate	search for specific rows, as defined by values and/or expressions.
R	Range	determine which rows will participate in the task, according to From-To values and/or expressions.
K	Key	change the Main Table index, and access and display rows according to the new index.
S	Sort	perform Online sorts by specifying the desired sort criteria. Each sort criterion defined by the end-user will create a temporary index that will be added to the Index list of the task for the duration of the task session.
F	Files	specify the names of all I/O files defined for the task and redirect any of them to a disk file, to the console, or to a printer. F mode is used mostly in Batch tasks.

Mode letter	Mode Name	Allows the end-user to...
E	By Exp	specify an expression in the Initial Mode Expression property (next), which will be evaluated dynamically at runtime to determine the Initial mode.

Initial Mode Expression

If you chose By Expression as the Initial Mode value, the insertion point moves here to allow you to specify an expression number. The expression will be evaluated at runtime to determine the Initial mode.

NOTES:

1. The result of the evaluation of the expression must be a valid Task Mode letter: M, C, D, Q, P, L, R, K, S, F, or O

For example, the expression:

`IF (A='QUERY';'Q'Mode;'M'Mode)`

means that if variable A in the task has a value of QUERY, eDeveloper will start execution of this task in Q (Query) mode. Otherwise, it will start execution in M (Modify) mode.

2. The Initial mode is evaluated as soon as task execution starts. Variables of the current task are not yet available during the evaluation of Initial Mode expressions. Therefore, if the expression uses variables, these variables must be arguments or variables of a parent task. If a legal value is not found at runtime for the expression used for dynamic definition of the initial mode of operation, a message box pops up, displaying a prompt to notify the user of the problem and to terminate the task.
3. By using the MODE literal in the Initial Mode expression, as shown in the example above, you instruct eDeveloper to check the string content for valid task modes. Any character in that string that is not a valid task mode will be cleared automatically. The value represented by the characters in the string are stored in an internal

representation. If the application is used with an eDeveloper system running with a non-English language configuration, the values in the string will be automatically changed to the corresponding values of the configuration's language.

End Task Condition: No (default)

If the value No is specified in this property:

- Browser and Online tasks end when the user ends the task, or when a developer raises an Exit event.
- A Batch task ends when all the dataview records have been processed or, if the task is abortable, when the end-user cancels the task by pressing Exit ESC.

If the Yes value is specified, the task will end before executing the record level, or after processing one row, depending on the setting of Evaluate condition. You can zoom from the End Task Condition property to the Expression Rules repository to set a condition that will activate the End Task condition.

Note: In Batch tasks an End Task condition is required to prevent an endless loop when:

- You specify the Scratch file (Main Table=0) for the Batch task to perform a loop.
- The Batch Task, is in create mode.

Note: Regardless of the End Task property setting, the Task Prefix and Task Suffix operations are always executed.

Evaluate Condition: Before entering record (default)

- Defines when eDeveloper has to evaluate and test the End Task condition:
- Before - the End Task condition test is done before entering the row.
- After - the test is made after updating the row. This way you can, for example, include in the End Task condition variables from the latest processed row.

- Immediate - tests immediately whenever the row is changed.

Termination of the Task for End Task Condition

When the Evaluate Condition property is set to Immediate, the End task condition is checked on the execution of every single operation, at all levels, and when End task = True, eDeveloper sets up an internal Terminate State. This state is checked only in the Online Record Main level, and causes the immediate termination of this level.

During the Record Main level of Online tasks, the Terminate state causes an immediate termination of the task.

During all other levels of Online tasks, and all levels of Batch tasks, the setting of the Terminate state does not cause termination of the task. The task will terminate when End Task = True at either Before or After End check.

- When eDeveloper terminates the task for End task = True, it:
 1. Executes the Record Suffix for browser and online tasks, if the row was modified
 2. Executes the group levels (if any in batch tasks)
 3. Executes the Task Suffix
 4. Ends the task

Examples of End Task and Evaluate Condition

- If you want 1 row processed, set End task to Yes and Evaluate Condition to After.
- If you want no rows processed, set End task to Yes and Evaluate Condition to Before.
- If you want 15 rows processed, specify the expression for End Task Counter(0)=15, and set Evaluate Condition to After.

Allow Event: Yes (default)

Yes means that the end-user can execute events assigned to the batch task, and that events assigned to the batch task will be handled by the eDeveloper engine.

No means that the end-user cannot execute events assigned to the batch task, and that an event assigned to the batch task will not be handled by the eDeveloper engine.

For example, while the batch task is being executed, the end-user cannot cancel the task. If a time event is defined to trigger every ten seconds, the event will be triggered, but the eDeveloper engine will not handle the event.

You can also create an expression that will determine when the end-user can trigger an event or when an event assigned to a batch task will be handled by the eDeveloper engine. To create an expression, zoom from the Allow Event property to the Expression Rules repository.

Return Value

eDeveloper lets you specify a return value for a task. This property is assigned to an expression. By zooming from this field, the Expression Rules repository opens and lets you assign the variable name or expression.

Main Table

You can define a table from the Table repository for the Main table property. This table's records will be scrolled by the executed task.

The Main Table property is not accessible if the task has a SQL command. Otherwise, the Main Table property contains an identifier number as it exists in the Table repository of the table that provides the basis for the dataview of the task. For more information, refer to Chapter 5, Application Engine.

Index

The default index is the table's first index in its Index repository. The choice of a specific index determines the fetching sequence of the dataview records for this task.

Setting the Index value to zero and not using an index expression will cause the table to be scanned in physical order without the use of any index.

The Index property is not accessible if the task has a SQL command. Otherwise, the Index property contains the number of one of the table indexes.

Index Expression

The Index Expression property is not accessible if the task has a SQL command.

Otherwise, to use the Index Expression property, you must leave the default zero as the Index definition in the Index property. The expression number you enter in Index Expression property points to an expression in the Expression Rules repository, to be evaluated at runtime.

If you use an index expression:

The result of the evaluation of the expression must be a valid index number for the selected Main table. If, at runtime, a legal value is not found for the expression used for the dynamic definition of an index, or if no expression is specified, eDeveloper uses the zero index (physical order) as default.

The Index expression is evaluated as soon as the task starts executing, before any of the task's variables are available. Therefore, an index expression that is based on variables, rather than constant values, must either use local variables that receive arguments, or use the variables of a parent task. Any Index expression based on variables that are not available will yield zero.

Note: When you use an index number inside any expression, if you want eDeveloper to update it automatically, you must qualify it explicitly using the KEY literal as follows. When you use the Index number in an expression, specify the index number *nn* as '*nn*KEY'. For example, if you enter "'2'KEY" as an index expression in the Expression Rules repository, and later insert a new index in the Index repository for the table ahead of the existing ones, thereby changing the old index entry from #2 to #3, eDeveloper will automatically change the "'2'KEY" to "'3'KEY" in the expression.

Advanced Tab

Selection Table: No (default)

This property is relevant for online and browser tasks only. If you set this property to Yes, you are instructing eDeveloper to implement this task to behave as a Selection Table task. Such a task is used to pick a specific value from a table (pick list) and then terminate.

Resident Task: No (default)

When you set the Resident task property to Yes, the called task or program is loaded into memory when its calling task is loaded, and remains in memory until the calling task is finished. This trades memory for improved performance, by attempting to keep the task in memory for repeated calling from a parent task. Use this option sparingly on systems with limited memory resources.

Chunk Size Expression: 0 (default)

This property is only relevant for browser tasks, and determines the amount of records to provide to the browser client. The end user may therefore browse through records on a local record cache without having to return to the enterprise server for every new record or page. An expression used for this property evaluates to a numeric value. This expression is computed when a task is initialized.

Exit URL

This property is only relevant for browser tasks. The property lets you define a hyperlink to an eDeveloper Program or any other URL. This hyperlink will be executed when a browser task ends.

Attached Context

This property enables you to attach a context menu structure to the current task. This property is enabled for Online and Batch tasks. In runtime, when the end-user requests a context menu while this task is in focus, the menus that display are those defined in this property, rather than the default context menus for the application. For more information, see Chapter 12, End-User Menus & Help.

Main Display: 0(default)

This property lets you use an expression to define the number of the form that will serve as your main display form. This enables you to create programs with dynamically defined displays.

When using a form number inside any expression, to update it automatically, you must qualify the number explicitly using the FORM literal as follows:

Specify the form number nn as 'nn'FORM. For example, if you enter '2'FORM as a form expression in the Expression Rules repository, and later insert a new form in the Form repository ahead of the existing numbers, thereby changing the old index entry from #2 to #3, eDeveloper will automatically change the '2'FORM to '3'FORM in the expression.

Icon File Name

This prompt allows you to specify an Icon file name to be used when the program is minimized under GUI operating systems such as MS Windows.

Enhanced Tab

Transaction Mode

There are five possible transaction mode settings for activating transaction processing:

- Deferred - Statements are stored in a cache and not sent to the physical database. These statements will be implemented at the expected commit time.
- Nested Deferred - Identical to Deferred but specifically applies to cases when a task is called from another deferred task. The child task opens as a new deferred transaction. Its transactions are committed to the database independent of the parent task.
- Active Within Transaction - Task is implemented within the parent transaction.
- Physical - Statements are implemented after the Task or Record Suffix.

- None - eDeveloper does not open a transaction for the task.
This option is available for browser tasks only.
Note: In a modal task, the Transaction Mode cannot be set to None.



For more information, see the Transaction Processing section in Chapter 11, Data Management.

Transaction Begin

Transaction Begin property options include:

- Before Task Prefix
- Group (for batch tasks only)
- On Record Lock
- Before Record Prefix
- Before Record Suffix
- Before Record Update
- None

The transaction processing may also be activated by a variable defined in the Task Properties Dialog.

Note that the Transaction Begin combo box appears as disabled when None is selected from the Transaction Mode property.

Locking Strategy

Designates the locking strategy as either No Lock or On Modify. If the Transaction Mode is set to None, the Locking Strategy is set to No Lock and is disabled. For more information refer to Chapter 23, Multi-User Considerations.

Cache Strategy

Designates the cache strategy as part of the data management options. For more information, refer to Chapter 11, Data Management. See also the Table Repository section in Chapter 4, Tables.

Error Behavior Strategy

Designates the error behavior strategy as either Recover or Abort. For more information, refer to Chapter 12, Error Handling.

Keep Created Context

When you select **Yes** or when a set expression is evaluated to True, a context opened by a **batch** or **online** task, as a response to a request, keeps its context for subsequent requests. This property can be enabled only for batch or online programs, but is not available for subtasks and browser programs. Although this property is primarily meant for batch tasks, it is also available for online tasks for debugging online programs. The default value is **No**.

Contexts using this task property are subjected to the context inactivity timeout value just like a context opened by a browser client task. The Context inactivity timeout value is reset as the request execution is completed.

Context requests can include the following HTTP arguments:

- **CTX** - the argument value is a context identifier that is stored on the server.

For example, `http://MyServer/Magic94Scripts/Mgrqisipi94.dll?APPNAME=MyApp&PRG_NAME=MyProg&CTX=9755586289880`

The CTX context identifier can be retrieved by the task that created the current URL by using the **GetParam**(CTX) function or **CTXGetId** function.

In runtime, the following behavior can occur:

- **The defined context exists and is idle.** When the context set by the CTX argument exists and is idle, the context is activated and the requested program will be executed within that context.
- **The defined context exists but is active.** When a request for a specific context is submitted and the context is responding to a previous request, the submitted request is kept pending until the previous request is completed. When several requests are waiting for execution from the same context, they are completed sequentially in the order of arrival. If the requester timeout of a pending request passes, the request will fail.

- **The defined context does not exist.** The request fails and displays this error message: -197, Context not found.
- **No context is defined.** When no CTX is set, the request opens a new context.

Note: With an open context, both batch and browser programs can be executed by using the CTX argument.

- **TRMCTX** - This argument value is Y for Yes. Any other value is regarded as No. For example, `http://MyServer/Magic94Scripts/Mgrqisipi94.dll?APPNAME=MyApp&PRG NAME=MyProg&CTX=9755586289880&TRMCTX=Y`

The TRMCTX argument can only be used when the request includes a CTX argument and the defined context exists on the server.

When the request is executed for a defined context and TRMCTX=Y, the following runtime behavior occurs:

- The requested program is executed within the set context
- As the top program is completed, the context will close. Note that browser client tasks are abruptly closed.

Client-Side Identification

Setting requests for execution within a defined context without any client identification means that any user could create a user-defined request with the same context identifier. To avoid this, the browser context requires some or all of the following client-side identification:

- Client-side IP, such as REMOTE_ADDR
- Client-side host name, such as REMOTE_HOST
- Client-side user name, such as REMOTE_USER
- SSL-related information, such as HTTPS, SSL_CLIENT_KEY_SIZE
- User-defined cookie variables

The client identification criteria can vary for every site. If a site utilizes the web server logon functionality, using the client-side user name can be sufficient. However, if the site does not support web server authentication, the client

identification can be defined by the Client-side IP, host name, or other client-side identification options.

You can assign client-side identification to the HttpSigVars Magic Request Broker setting, described under eDeveloper Requester settings in Chapter 19, Distributed Application Architecture.

Direct SQL Command

The Direct SQL Command object lets you provide a SQL statement that is passed to the underlying SQL database to provide the task's dataview, or to perform the processing required by the command.

When SELECT statements are complicated, it is faster to let the RDBMS server join and constrain the rows, bringing only the specified rows into the eDeveloper task's dataview. This is especially helpful in a client/ server environment, where decreasing network traffic improves overall system performance.

An RDBMS can perform vertical updates and deletes with one SQL statement, a cursor, and a simple transaction. eDeveloper tasks process the dataview one record at a time. In an Insert, Update, or Delete task, each record is processed separately, even though all the records may be processed as one transaction.

You can use explicit SQL where DDL operations are specific to runtime. For example, you may need to create a special table index for a specific report and then drop the index when the report is complete. You may also want to create a temporary table in the RDBMS.

In general, performing other types of DDL from eDeveloper is not recommended. RDBMS joins are usually more efficient, although not in all cases.

Using Direct SQL Command

eDeveloper lets you embed native SQL statements in your eDeveloper application with the SQL Command dialog. If the SQL Command dialog is used, the task's Main table is the SQL statement.

1. From the Workspace menu, choose Programs.
2. Create a new line.
3. Type demo for the program name.
4. Zoom to the task tree.
5. Select the SQL Command option from the Task menu (CTRL+Q)
The SQL Command dialog, as shown in Figure 6-2.

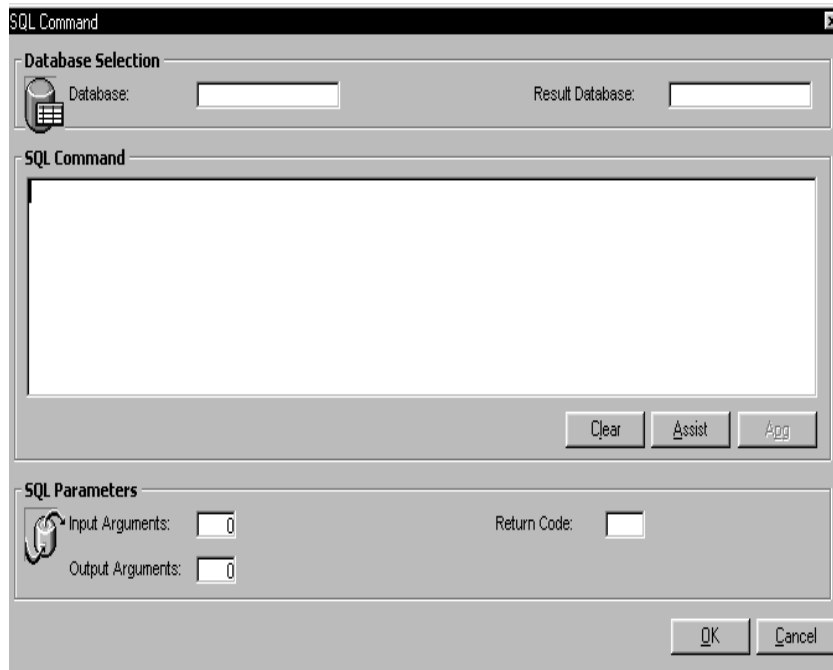


Figure 6-1 The SQL Command Dialog

eDeveloper executes the SQL command before it executes the Task Prefix level. The task can use the returned data as an integral part of its dataview.

eDeveloper does not attempt to analyze the command's syntax and semantics. This gives the programmer flexibility, but requires that the programmer plan the programming carefully. eDeveloper does not protect the data from errors in the Direct SQL commands.

eDeveloper's Direct SQL feature is a tuning tool to improve performance. Use this feature selectively, and only when its use results in significant performance improvement.

Use the Direct SQL feature:

- When you need to calculate statistics on the database, such as how many records have a specific property, or the total sum of this month's salaries.
- When you want to make a vertical update to your data, such as to increase all salaries by 5 percent, delete all old records, and copy parts of one table to another table.
- When you want to utilize existing code that was developed and compiled using the SQL DBMS tools, such as stored procedures.

Direct SQL Task Elements

Create a Direct SQL Task

1. Select the SQL command from the Task menu. The Direct SQL Command dialog appears as shown in Figure 6-1.
2. From the Database field, zoom to the Database List.
3. Select the database from which to run the explicit SQL.

SQL Command

eDeveloper does not check the syntax or semantics of the statement. The underlying database performs all statement processing at runtime. If an error occurs at this stage, the eDeveloper task terminates with an explanatory message.

You can use the SQL command object to perform global updates, global deletes, DDL statements, and PL/ SQL Blocks.

The SQL command may also contain the name of a predefined procedure that was developed and compiled using DBMS facilities (stored procedures). If such a procedure name is specified as the SQL Command, the database procedure is invoked and executed by the RDBMS. These stored procedures can be used as though they were SQL command types. All the rules regarding Direct SQL commands described here apply to stored procedures when called.

Executing Stored Procedure

You can use a stored procedure by specifying the reserved word *exec* in the following format:

`exec procedure-name parameters`

If the procedure accepts more than one parameter, separate the parameters with commas. If the procedure body is a Select statement, treat the procedure as if it is a regular Select statement. In other words, select *Options/ Generate Program*. There are three APG characteristics that should be kept in mind when in a stored procedure:

- APG invokes the stored procedure. If the stored procedure contains statements other than the Select statement, change the other statements into Remarks while using the APG.
- Stored procedures that are called from eDeveloper receive parameters by value.
- An output parameter can only result from a Select statement.

Input Arguments

You can insert a colon (:) followed by a number anywhere in your Direct SQL command as a property designator. The colon plus number combination is replaced by the property value specified in the SQL Command dialog's Input Argument table. The arguments are computed just before the SQL Command is prepared, and eDeveloper replaces the property designator with the computed value in the SQL statement's text. The statement is then passed to the RDBMS for syntax evaluation and execution.

The SQL command syntax is always re-evaluated before the command is executed, regardless of the Resident Task property value (in the Task

Properties dialog). Generally, Input arguments are used as placeholders for SQL WHERE clauses.

Output Arguments

SQL SELECT statements or stored procedures provide eDeveloper with an alternative dataview to the standard Main table ordinarily used. When the SQL Command results produce a result table or a data stream, eDeveloper provides the buffers necessary to accept the data. These buffers take the form of virtual variables defined in the SQL Command's task. The virtual variables must match, in attribute and picture, the data generated by the SQL Command.

Assist Utility

The Assist utility provides an easy way to construct SQL statements that include database names for tables and columns. The SQL Command Assist utility is not a substitute for knowing the SQL language, which is a prerequisite for using the SQL command.

Use the Assist utility to reference the names of tables and columns in the database, but be careful when you use this utility to build syntax.

For example, the Assist utility will build

```
SELECT ALL FROM EMPLOYEE
```

even though this is not valid syntax. The correct syntax is:

```
SELECT * FROM EMPLOYEE
```

Use the Assist Utility

1. Click the Assist button on the SQL Command dialog. The SQL Assistor dialog appears. as shown in Figure 6-2.
2. Select a keyword from the keywords list and press ENTER to place the selected keyword in the Statement box.

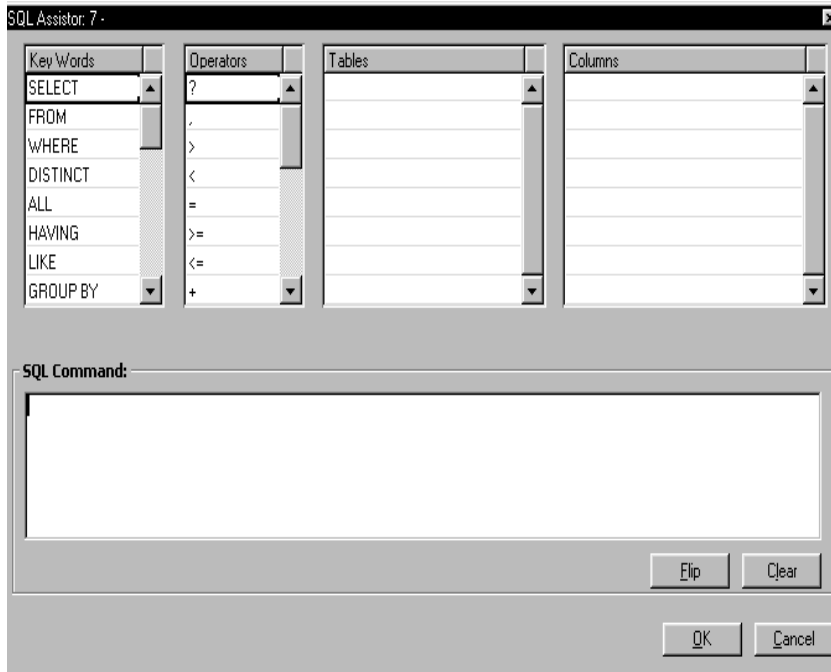


Figure 6-2 SQL Assistor Dialog

Continue the same for selecting column names, table names and operations from their lists.

You can view the table under either its eDeveloper name or its RDBMS name. By clicking the Flip button, you can toggle back and forth between the eDeveloper and RDBMS table names. Whether selected from the eDeveloper or the RDBMS name, the SQL statement shows the RDBMS name.

The Table box shows only tables that are defined in eDeveloper's Table repository. However, it is possible to enter SELECT statements that access any available tables in the database, even if those tables are not defined in the Table repository.

SQL Command Automatic Program Generator

The SQL APG utility constructs a complete eDeveloper program from SQL statements the programmer inputs. Running the SQL APG is the only way to check the SELECT statement syntax before running it. The utility generates an eDeveloper task structure based on a SELECT statement or a stored procedure.

The generated task contains the following items:

- The original SQL Command
- Select virtual operations and virtual field definitions for all the result table's columns
- A full Output Properties table with the automatic virtual variables
- A default form for user interaction

The developer only has to define input properties for the SQL command if required and change the default form if necessary.

Behavior of Direct SQL SELECT Statements

Some of the Direct SQL statements that have a result database behave differently in eDeveloper.

Direct SQL statement execution always occurs before a task prefix. Therefore, once you have entered the task prefix, the SQL statement has already been executed.

Browser Client and Online

In browser client and online tasks:

- You must browse on the SELECT statement result.
- eDeveloper creates a temporary result table or file and inserts all the records into it so that the user can scroll on the records. The table is deleted at the end of the task.

Batch

In batch tasks:

- Every record is only read once, and you do not need to go backwards.
- eDeveloper opens a cursor according to the SELECT statement and retrieves all the records one after the other.

Result Database as Input Database

Setting the same value in the Result Database field as in the Input Database field, lets you make the result database the same as the input database. In most cases, the underlying RDBMS allows an

INSERT INTO table AS SELECT...

statement, which copies all the data to a table in one command. This method is faster than opening a cursor. All of the records are retrieved from the client and inserted into the result table.

In such cases, the SQL gateway creates a table in the database and sends:

```
InsERT INTO temp_ table AS SELECT "direct SELECT  
statement"
```

The user may then scroll on that table to speed up the SELECT statement execution. This method is especially helpful when the result set is large because there is no retrieving and inserting of each record to the temporary table.

Result Database Different from Input Database

When the result database is different from the input database, a temporary table is created. A cursor is defined, and the records are retrieved and inserted one by one into the result table. The user may then scroll on that table.

Recommendations

- When the result is large, it is best to use the option that makes the result database the same as the input database. Then the Insert INTO AS SELECT statement will be used.

- When the result is relatively small, it is best to use the result database as an ISAM database that will reside on the client. Then when you create the result and scroll on it, the work is only performed on the client, which reduces network traffic.
- Use the memory gateway when the results are relatively small to enhance performance.

Restrictions on Using Direct SQL

A task with a Direct SQL command instead of a Main table is a normal eDeveloper task and has almost the same functionality. There are, however, some restrictions:

- All the tables participating in the SQL command must be from the database that is declared in the Database field of the SQL Command dialog. Mixing tables from different databases is not allowed.
- In an online task, do not update a column that belongs to the SQL command's view (the result table). Such updates are never written back to disk because the result table is deleted when the task terminates. Use links to other tables to update information.
- Do not use commands that alter the state of a record that is the current view of an ancestor task.
- During batch processing, do not update a Group level of a SELECT operation. Group Level operations require that the preceding record be refreshed before the Group Level operation. Refreshing records is impossible because the task's data stream is a one-way stream that cannot be stopped.
- Use task level transactions on batch SQL commands.
- Do not use the COMMIT and Rollback commands in your Direct SQL. If these commands are not generated through eDeveloper's standard transaction management layers, the results are unpredictable. COMMIT and Rollback functionality is achieved correctly only by choosing the right transaction mode in eDeveloper's task level table.

- The SQL statement is executed by the underlying database. The command syntax is the developer's responsibility, and the developer is not prevented from using DBMS- specific extensions. If application portability among various SQL DBMSs is required, be careful not to include DBMS- specific SQL extensions in Direct SQL. In the SQL statements created by eDeveloper this is not a problem. The eDeveloper gateways generate the correct, optimized syntax for each database.
- When using Direct SQL in online tasks, eDeveloper creates a temporary table. Optionally, the table can be created in the database. Creating the temporary table in the same RDBMS using that RDBMS's utilities is usually more efficient than having eDeveloper read and write each record.
- Range on a Direct SQL output argument is not allowed.
- To execute a stored procedure, prefix the name with the word `exec` as in:
`exec sp_order_update`

The gateway actually executes the generated program statements to recognize the correct form of the result set in the APG. So, if your statement modifies data, the modification will occur.

- DB2 and ODBC cannot be used as the Result Database of a Direct SQL task.

Binding Variables

When using Direct SQL tasks, eDeveloper sends the SQL statements to the database after creating them dynamically. The database receives the SQL statements as alpha or numeric string values, and parses and executes the SQL statements.

Parts of the SQL statement can be placeholders for predefined values. Replacing placeholders with values is called binding, which eliminates the process of reparsing SQL statements by the database.

Binding Restrictions

Binding variables for SQL statements are supported by Oracle RDBMS only.

Binding variables are supported only with select statements or stored procedures that return output.

You can specify a variable as a binding variable by using the Tilde (~), as displayed in the following example: `Select * from table where field1 = ~1 and field2 > ~2`

Binding is supported for Alpha, Logical, and Numeric eDeveloper types. Apostrophes are not required when binding alpha values. eDeveloper stores Date, Time, Logical, and Numeric eDeveloper types as numeric values.

You can use the binding mechanism when the parallel data type in Oracle is Numeric.

Note: SQL statements with bound (~) and unbound parameters eliminate the benefits of binding variables.

Allow DSQL in a Deferred Transaction

eDeveloper lets you define a Direct SQL statement for all types of task modes: browser, online, and batch.

When generating a task from the SQL Command dialog, you can select the **browser client** from in the APG Option combo box, which creates a browser-based task with data retrieved from a Direct SQL Statement.

In a browser-based task, you can specify only select statements and call stored procedures that return data. DML statements are not allowed, such as Update, Insert, and Delete operations.

If the DSQL rows are linked with other tables, the linked data for the non-DSQL rows are stored in the transaction (trans) cache, which behaves as a normal link in a deferred transaction task.

A DSQL task assigned as a browser client subform will be blocked. The timing of the SQL Command execution prevents eDeveloper from refreshing the subform.

Task Control

Task Control properties are used to control the runtime behavior of a task.

Task Control Properties as Conditions

You can specify Yes or No for most of the control properties or you can assign them the number of a logical expression to be evaluated at runtime. This way, the control property setting can be made dependent on the result of a condition expression. A condition expression that evaluates as True is equivalent to a Yes value in this property, and a condition expression that evaluates to False is equivalent to a No value in this property.

Task Control Properties Dialog

Use the Task Control properties to specify which modes of operation will be available on the task menu for the end-user to set at runtime. The setting of Task Control property also determines which modes of operation will be legal as the Initial Mode for the task.

Note: If the Initial Mode setting for the task is disabled by the corresponding Task Control property, at runtime eDeveloper issues an error message to that effect. When the end-user exits the message box, the task will terminate.

The Task Control dialog is divided into two tabs:

- Modes Tab
- Behavior Tab

Modes Tab

Allow Options: Yes (default)

No means all end-user options will be disabled; the task will run using its Initial mode and the end-user will not be able to change it.

Allow Modify: Yes (default)

No means the Modify option will be disabled.

Allow Create: Yes (default)

No means the Create option will be disabled.

Allow Delete: Yes (default)

No means the Delete option will be disabled.

Allow Query: Yes (default)

No means the Query option will be disabled.

Allow Locate: Yes (default)

No means the Locate option will be disabled. In Query mode, the automatic locate mechanism will not operate.

Allow Range: Yes (default)

No means the Range option will be disabled.

Allow Index Change: Yes (default)

No means the Index change option will be disabled. The automatic index optimization will also be disabled.

Allow Sorting: Yes (default)

No means the Sorting option will be disabled.

Allow I/O Files: Yes (default)

No means the end-user will not be allowed to redirect input or output operating system text files.

Allow Index Optimization: Yes (default)

No means the Index Optimization mechanism will be disabled.

Allow Locate in Query: Yes (default)

No means the automatic locate mechanism will not operate in Query mode.

Allow Printing Data: No (default)

Yes means the Allow Printing Data option will be enabled. This option is available only for online tasks with the Main table selected.

Behavior Tab

Open Task Window: Yes (default)

This property is relevant for Batch tasks only, because the Task Window is always open for Online and browser tasks.

If you specify No for a Batch task, eDeveloper does not display its Task Window on the screen. If you specify No for an Online task, eDeveloper will not start execution.

Close Task Window: Yes (default)

The Yes value tells eDeveloper to close (that is, remove) the Task Window when the task ends. The No value tells eDeveloper to leave the Task Window with its final content on the screen when the task ends. When executing the same task again, the previous open window is closed and a new window opened. When exiting the runtime environment, all open windows are closed. This property is only relevant for Online and batch tasks.

The following table displays the allowed combinations of the Open and Close Task Window properties:

Open Task Window	Close Task Window	Relevant for
Yes	Yes	Online and Batch tasks

Open Task Window	Close Task Window	Relevant for
Yes	No	Online and Batch tasks
No	No	Batch tasks only
No	Yes	Batch tasks only - same meaning as No

Foreground Window: Yes (default)

This property is only relevant for online and batch tasks. At runtime a task will usually open its window as the top window, meaning it will be fully displayed and will overlap all other open windows. However, eDeveloper can also open the task's window in the *background*, behind all the other active task windows. Select the No option when you want the task to substitute its task window for the standard eDeveloper background window or when you want to leave the task window open above the eDeveloper work area background. This way, you can display background information such as time or date.

When you set Foreground Display to No, you would normally also set the Close Task Window to No, to keep the displayed background window open. If you want to refresh this background window upon user demand or periodically, use the Event repository to make program execution dependent on a particular keystroke or on the passing of a specified elapsed time.

Refresh Task Window: No (default)

This property determines whether the task window is refreshed after each row is processed and its Record Suffix is executed, refer to Chapter 5, Application Engine. This property is only relevant for Online and batch tasks.

eDeveloper works on a single row at a time. Even in a multi-line display, in Line mode, only the currently-highlighted line is refreshed and displays the exact values of the dataview row. The other lines in the display are refreshed as soon as the input focus is moved to them. Therefore, it is possible that a

non-highlighted line can be displayed without reflecting its most up-to-date values as stored in its dataview row.

In another scenario, it is possible for the current dataview row to contain some columns that also participate in another row of the same dataview, because of a link operation. In this case, the other dataview rows may also be displayed on the same screen, in Line mode, or in a parent task. If such a column is changed, the changes will be displayed only for the currently highlighted dataview row. In such situations, it is a good idea to immediately reflect the changes in all the dataview rows currently being displayed, using Refresh Task Window set either to Yes or to an expression. It is preferable to use an expression, if possible, because such refreshing can slow down task execution.

Record Event Interval: 0 (default)

This property is only relevant for Batch tasks and defines after how many processed records will eDeveloper poll pending events in the events queue. This property's expression should evaluate to a numeric value and is computed at the task's initialization stage. For example, if the expression evaluates to 500, the eDeveloper engine looks for pending events in the events queue after every 500 records. If a matching handler for an event is found in the queue, the handler will be executed.

Form Records: 0 (default)

The Form Records property is relevant for Batch tasks only. It is used to produce reports on pre-printed forms, where the header and footer are in a fixed position and a fixed number of lines exist on each page. In this case, a fixed number of records should be output to each page, with the last page padded with empty lines. For more information, refer to the Output Form operation in the Chapter 7, Operations.

Cycle Record Main: Yes (Default)

If Yes is specified in this property, when the end-user finishes working on the last column of the current row the insertion point will return to the first column of the same row. This process continues until the user leaves the row and moves to another row, or exits the task.

If No is specified, the insertion point will move automatically from the last column of the row to the first column of the next row of the dataview.

This property is relevant for Online tasks only.

Confirm Update: No (default)

If No is selected, eDeveloper confirms only Delete operations. If Yes is selected, eDeveloper also confirms Create and Modify operations. You may also use an expression to override this behavior. For example, Stat(0, "C'MODE") will confirm Create operations only.

Confirm Cancel: No (default)

eDeveloper will prompt the user upon the cancellation of updates on the current record, if Yes is selected. Confirming the cancellation will cancel the changes and the record will revert back to its original values. Declining the cancellation will keep the new values in the records.

Force Record Suffix: No (default)

The Force Record Suffix property is only relevant for browser and Online tasks. If its value is No, eDeveloper executes the Record Suffix level only if the current dataview row has changed; that is, if one of the task variables was modified.

If the Force Record suffix property is Yes, or if you specify a condition expression that evaluates to True at the end of Record Main execution, eDeveloper executes the Record Suffix whether or not modifications were made to the row.

Force Record Delete: No

The Force Record Delete property allows you to delete rows from the task dataview conditionally.

The Delete condition expression is evaluated before eDeveloper begins Record Suffix processing. If it evaluates to True, the row is deleted using the same mechanism as if the user had pressed a Delete key, as described below.

Delete Mechanism

If the row has been modified, or if the task is running in Batch, the Record Suffix is processed once with the task status Modify, in order to perform any update operation defined there and to complete the processing logic. The Record Suffix is then processed a second time with task status Delete, in order to perform any delete-related operations, such as deleting chained rows in one-to-many relations, or Update option operations using the Incremental mode. For more information refer to the Update Operation in Chapter 7, Operations. If the row has not been modified, the Record Suffix is executed once only, with task status Delete. This logic is illustrated in the figure below.

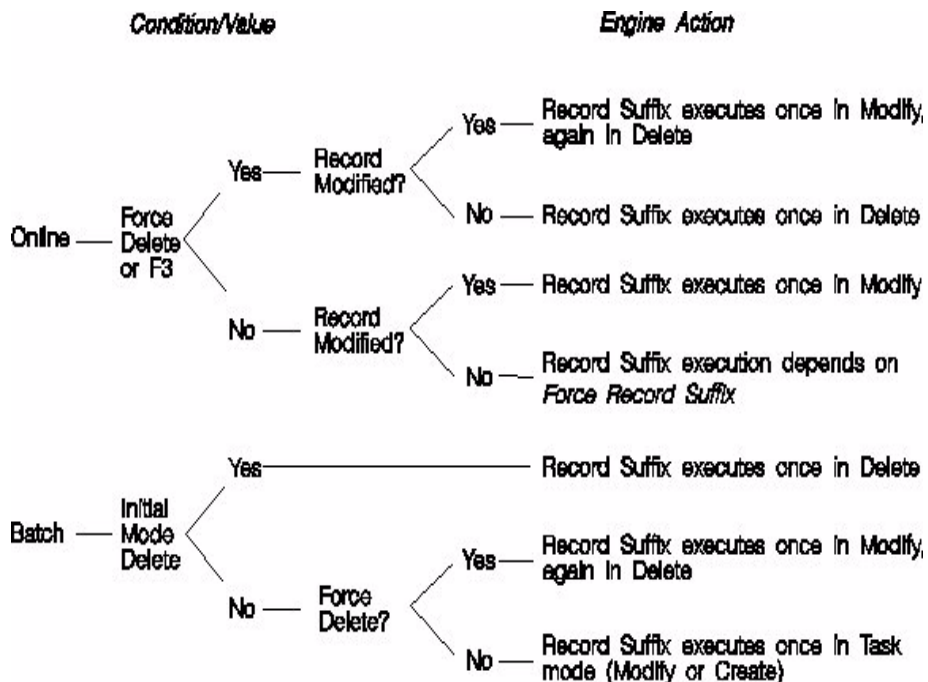


Figure 6-3 The Effect of Force Record Delete

If, with Force Record Delete, you want to avoid the double-processing of specific Record Suffix operations, use the Stat function in the condition expression specified in the Condition column for those operations.

Local Variable Repository

Almost all tasks need to store certain information temporarily for the duration of the task execution. For this purpose you can define local variables, selected from a temporary “scratch” memory area called a virtual file. The virtual file may be edited by selecting Variables from the Task menu.

All selected local variables appear as entries within a task’s Variable repository. The Variable repository behaves like the Column repository of a table. The only difference being that in the Variable repository you cannot create or delete a line using the normal table editor. A line is created or deleted in this repository automatically when you create or delete a Select Virtual or a Select Parameter operation in the Record Main Execution repository, and no other way. For more information refer to the Select Operation section in Chapter 7, Operations.

Properties of the Variable Repository

The following properties exist for each entry in the Local Variable repository:

(for Variable identifier)

This column contains an automatically generated sequential number used by eDeveloper as a variable identifier. You cannot edit this column.

Name

Enter a descriptive name. If you want the variable to inherit a field model, you can leave the name column blank and eDeveloper will copy the model name in it.

Model

You can create local variables based upon pre-defined field models. Either zoom from the model column and select the desired field model, or type in the number of the field model.

When a model is selected, its name will appear beside the model number and the attribute and picture columns will reflect the model’s attribute and picture. All other properties will be inherited from the selected model.

You may also leave the model column empty (zero) and define your local variable properties directly through the variable's property sheet. In any case, every property can be broken and be set locally

Attribute

Click the combo box to select an attribute from the Attributes list, or enter the initial letter of one of the eDeveloper data item attributes: A for Alpha (the default), N for Numeric, L for Logical, D for Date, T for Time, M for Memo, B for BLOB, O for OLE, X for ActiveX, and V for Vector. For more information, see the Attributes section in Chapter 3, Data Items.

Picture

- For an Alpha or Memo variable, the minimum picture required is its length. In addition, you can enter any other picture specifications the variable needs.
- For a Numeric variable, the minimum picture required is the number of its integer digits and, if needed, a decimal point and the number of decimal digits. You can add any other picture specifications the variable needs.
- For a Logical, Time, or Date variable you can accept the suggested picture or you can add any picture specifications you want for the variable.
- For a BLOB, there is no picture available.
- Additional details about Picture specifications are available in the Pictures section of Chapter 3, Data Items.

Local Variable Properties Sheet

The property sheet of the local variable is the same as the property sheet that is available in the Model repository for an Alpha field model.

Please refer to Chapter 3, Models for more information on setting the Field model properties

Expression Rules Repository

The Expression Rules repository is a compound window that includes:

- Expression Rules repository area at the top left corner.
- Variable list at the top right corner.
- Expanded variable area at the lower section of the window. In the expanded expression area, the expression highlighted in the Expression Rules repository area at the top of the screen is displayed with each variable identifier letter replaced by its full variable name.
- Buttons at the bottom of the window, described in the table below.

Action	Function Button	Meaning
OK ENTER	OK	Accept changes and terminate the dialog. Copy the current expression identifier number to the property from which you zoomed into the Expression Rules repository.
Cancel F2	Cancel	Cancel all changes to the Expression Rules repository and terminate the dialog without copying the current expression identifier number.
Show Expression	Show	Show the edited expression in the expanded expression area.
Function List Alt+F	Function	Display a list of all the available functions.

Action	Function Button	Meaning
Action List Alt+A	Actions	Display the Action list used for selecting actions while editing a KbPut or KbGet function.
Shortcuts List Alt+S	Short-cuts	Display a keyboard mnemonic list used while editing a KbPut or KbGet function.
Rights List Alt+R	Rights	Display a list of allowed rights used with the Rights function.
Control Name List Alt+N	Controls	Display a list of control names used with the controls in the form.
Table List Alt+T	Tables	Display a list of the available tables to be used in table-related functions.
Program List Alt+P	Programs	Display a list of the available programs to be used in program- related functions and operations.
Error List Alt+E	Errors	Displays a list of supported errors to be used in error-related functions.

Note: Place the insertion point on the Expression Rules repository to copy values from a list to an expression.

Expressions

An expression is a constant or a formula for computing a value. An expression's value can be of type Numeric (including Date and Time), String, or Logical (TRUE or FALSE). The expression consists of a sequence of

operators indicating the action to be performed, and operands on which the operation is performed. Operands may contain variables, functions, constants (called literals), or other sub-expressions.

- A variable is part of the current Variable list. The Variable list contains both Real and local variables.
- A function is one of eDeveloper's library of built-in functions. The built-in functions can be classified into the categories Date & Time, Mathematical & Trigonometric, String Manipulation, Conversion, Identification, Tests & Conditions, Logical, Database Interface, Table Management, Value Manipulation, and Action. You can access and select functions during editing by pushing the Function button. For a complete directory of available functions, refer to Chapter 8, Expression Rules.

A function usually contains one or more arguments, delimited by commas. The argument list is enclosed in parentheses. Even if the function contains no arguments it still requires a set of parentheses: for example, Date ().

Every function returns a value.

- Operators can be numeric (for example, +), string (for example, &), or logical (for example, NOT), as described in Chapter 8, Expression Rules.
- Each entry in the Expression Rules repository evaluates to a numeric value, a string, BLOB, or a logical value (True or False), depending on the type of variable and functions used. If an inappropriate combination of types is used, eDeveloper displays an error message.
- Functions can be nested using parentheses. For example, the following expression gives the real length of the string stored in the variable A: Len(LTrim(RTrim(A))) which is equivalent to Len(TRIM(A)).

Further complexity is possible by nesting logical expressions, including IF, AND, NOT, and OR.

eDeveloper will remove any superfluous parentheses.

Form Repository

The Form repository contains form definitions for a task. Each entry represents a form. The first form is the task's main window.

From the **Task** menu, click **Forms** to open the Form repository, as shown in Figure 6-5.

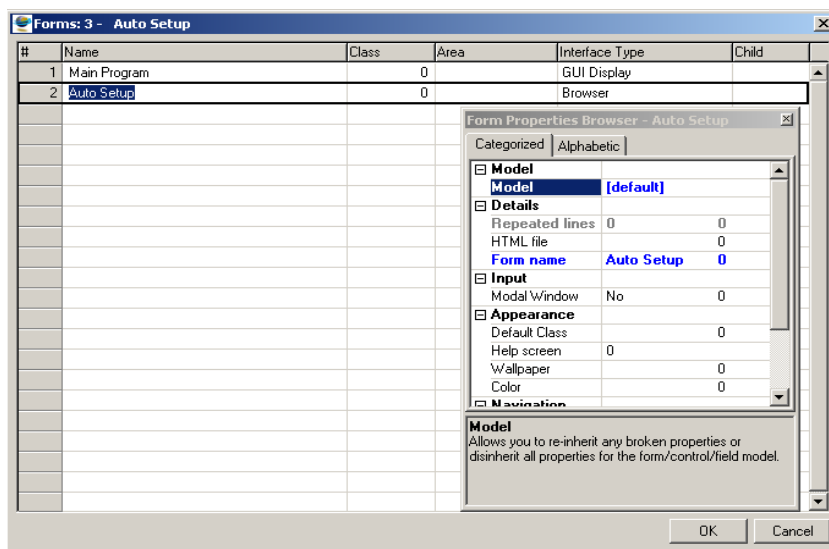


Figure 6-4 Form Repository

When you define a task, eDeveloper automatically creates an initial entry in the Form repository. This form entry is specific to the task. You cannot delete the form or move it to another position in the repository.

When you create a subtask, eDeveloper automatically appends a new entry to the root task's Form repository. This entry is for the subtask. You cannot delete this entry, although you can modify the columns associated with this form. This entry may move if you add or delete entries in the Form repository of ancestor tasks. From the Form repository of a subtask, you have access to all of the forms for that subtask and to the entries for the subtask's ancestors.

For a Class 0 form, the Form Editor displays a secondary window above the main windows of all the ancestor tasks in the program's hierarchy. This lets you see how the display appears at runtime.

For online tasks, the form is used to display the dataview and to allow end-user interaction.

For batch and browser tasks, the form can be used for feedback information, such as to show that a task is running. It can also serve as a template for end-user options, such as to set the locate and range fields for the batch process.

You can define the selected form by specifying the property value on the form properties sheet.

Double-click a form entry to start the form editor or click **Edit Main Form** from the **Options** menu, as shown in Figure 6-5.

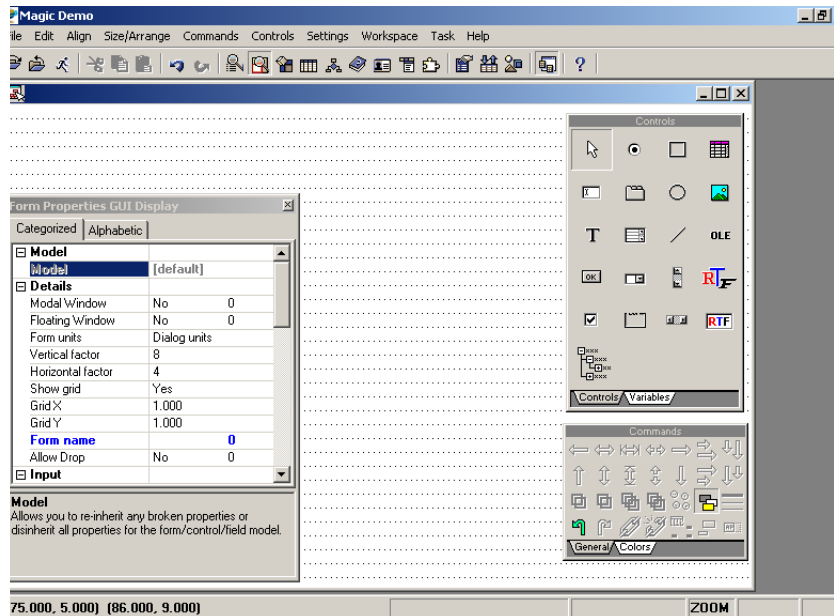


Figure 6-5 Form Editor

You can add controls to the form by selecting a control from the Control palette, and dragging and dropping the control on to the form. You can define

the control by specifying property values displayed on the control properties sheet.

Form Repository Columns

The following is a description of the form columns:

(for Form identifier)

A sequential number representing a form's position in the repository. Magic automatically generates and maintains the form identifier. You cannot edit this column. The first entry of the task is the task form, Class 0, and cannot be deleted or moved, although some of its properties, such as Name and Child can be modified.

Name

You can specify the form name. This field contains the task name or another name given to the form. When you create a new task or subtask, Magic also automatically creates the Form repository entry for that task. The name of the task, as it is entered in the Task Properties dialog, appears as the name of the form. As long as the task name and the form name are the same, any changes to the task name are automatically copied to the form name. You should provide a name for all additional form entries you make in the Form repository. The form name appears at the top of any of its associated Class 0 display windows.

Zoom or double-click on an entry in the Name column to activate the Form editor type selected for the form. If the form is a Browser or an HTML Merge form, zoom to the editor that you have specified in the Web Authoring Tool setting, from the Partitioning tab of the Environment dialog. If the form is a Web Online Response form, zoom to the Merge Command repository.

Class

Magic automatically assigns Class 0 to the task's main form. You can define additional Class 0 forms. Class 0 is reserved for interactive forms.

You can also define forms with a Class Number > 0. These forms are used for reports, I/O record layouts, and HTML output. If a task produced two reports you could assign the Class Number 1 to all of the forms of the first report, and assign the Class Number 2 to all of the forms of the second report. The actual class values are of no significance except to associate all header, detail, and footer forms within a single report, and to ensure that they are treated as a single unit in End of Page situations.

Magic will automatically adjust the Units of Measurement, Vertical Factor and Horizontal Factor settings for any group of forms with the same class number, where Class > 0, so that all the related forms have the same settings, which will be the highest resolution defined in any one of the forms of that class.

Note: Changing the form class may cause controls to be deleted.

Area

For interactive windows, Class 0 forms, the Area field is disabled. For report forms, the Area field can be set to Header (the default), Detail, Footer, Page Header, or Page Footer. For I/O record layout forms, the field is disabled.

The physical placement (nesting) of Header, Detail, and Footer forms in the Form repository is important. Each header form must be followed by its matching footer form. All header-footer pairs must precede the detail lines. The header-footer pairs must appear in their outside-in nested order. The detail lines should be placed after the innermost header-footer forms pair. Detail forms should appear in the order they appear on the report. When displayed within the report editor, Magic orders the forms to their output format of nested header, detail, footer, reflecting the sequence in which they appear on the report.

Interface Type

For Class 0 forms, the interface type is a Browser form or GUI form. For Class > 0 forms, click the Interface Type column to choose one of the following options in the table below.

Select...	For...
Browser	Browser tasks

Select...	For...
HTML	Tasks implemented in an HTML environment
GUI Display	Online interactive tasks
GUI Output	Online reports
Text-Based	Textual output
Frame Set	Creates a complex Internet document where the browser window is divided into multiple frames.
HTML Merge	Merges data from a Magic task with a predefined template file to create any dynamic HTML page based on a predefined HTML design.
Web Online Response	Specifies the commands to be executed in response to a Magic program called by a Web Online page.

If you convert a Class > 0 GUI form into a text-based form, and the GUI form contains controls that are incompatible with a text-based form, a message warning you that existing controls may be modified or deleted appears on the message line. A dialog appears asking if you want to overwrite the current display. Press **ENTER** to accept the overwrite. The form and its controls are changed to text-based rules, and mismatched controls are deleted.

Display forms have either a Browser interface or a GUI Display interface and are described in Chapter 9, Display Forms

Output forms have the following interface types: HTML, Frame Set, HTML Merge, or Web Online Response, and Text-based. They are described in Chapter 10, Output Forms.

Child

For text-based forms, the Child column is disabled. For Class 0 GUI forms, click on the combo box in the Child column of a Form repository to choose one of the available options:

- Yes - The form will be opened as a secondary (child) window.
- No (default) - The form will be opened as a main window.

You can also enter an expression to determine when the form opens as a secondary (child) window. The expression is computed when the form is opened before the Task Prefix operations are run.

Working with Forms

Forms let the end-user interact with the application. They contain controls that either display data options or let the end-user enter data.

The Form Editor displays the form's frame and content as specified in the Form repository. You can design a form by assigning form properties and selecting controls and defining their properties.

Resizing a Form

You can define the logical size of your form from the Navigation properties listed in the form properties sheet. Alternatively, you can use the **SHIFT + [Arrow keys]** to change the layout area of the form. Use the arrow keys or drag the title bar of the layout to move the form. Click and drag the right and bottom borders of the layout frame to resize the form layout area. After resizing or moving a form layout area, press **ENTER** to accept the change, or press **ESC** to cancel the change.

If you make the layout frame smaller than the layout area, eDeveloper automatically adds scroll bars to the form. You cannot change the layout area dimensions of a form in such a way that controls would be lost.

Form Units

The size and position of controls on a form are defined in terms of *form units*. Form units also determine the resolution of the form. You set the form units in the Form Properties sheet. You can define a form in dialog units (characters), centimeters, or inches. A dialog unit is the size of the form's current font. Use dialog units to maintain the same proportions in a form when using different screen resolutions.

Use the Vertical factor and Horizontal factor properties in the Details section of the Form Model sheet to determine fractions of dialog units for finer resolution.

Units of measurement for HTML forms are always in dialog units based on the form's font. The units of measurement determine the size of the grid displayed when the form is developed.

The grid specifies the row to which the control is placed on the form, and, in the case of a pre-formatted control, the number of spaces to be used for padding.

Palettes

Palettes contain icons that let you select controls or control formatting commands. eDeveloper palettes are described below:

The **Control palette** contains icons of the controls that can be placed on the form depending on the form type specified.

The **Command palette** contains icons that let you align, resize, link, group, and arrange controls for GUI and HTML forms.

The **Color palette** lets you define the color for a particular control as displayed on the palette. Each color is identified by a color button.

The **Static Table palette** provides you with controls for the precise placement and independent formatting of all HTML controls through the use of Static Table cells and rows that can easily be modified for custom design.

Form Templates

The current form layout can be saved to a file as a template. The default file extension is MFT, Magic Form Template. The template contains all controls and all control properties except for the following:

- Control Name
- Property expressions
- Data properties
- Help and Prompt Help properties
- Auto Call Program properties

The above properties will have empty property values when loaded from the template file.

If you load a template into an existing form, the template file will overwrite all the form settings, and all the controls that were on the form will be deleted.

Form Templates are limited to GUI forms. HTML templates are not accepted by the GUI Form editor.

Form and Control Properties

Form properties determine the visual display and format of a task. Control models determine the visual display and operation of controls. Form and control properties are described in Chapter 9, Display Forms, and Chapter 10, Output Forms.

Form and control properties can be a set of inherited properties. You can define form and control models in the Model repository. For more information, see Chapter 3, Models.

Keyboard Shortcuts

Keyboard shortcuts for forms are listed in the table below.

Key	Modifier	State	Operation
F5		Control is selected	Control properties

Key	Modifier	State	Operation
F5		No selection	Form properties
SPACE			Clear the selection
DELETE			Delete the selection
INSERT			New control menu
ENTER		A Static control is selected	Text entry
ENTER		A Choice control is selected	Rotate choice layer
ARROWS		Controls are selected	Move the selection
ARROWS	SHIFT	Controls are selected	Size the selection
ARROWS		No selection	Move the form
ARROWS	SHIFT	No selection	Size the form
ALT			Places a control on an existing table column or select a table column
TAB			Select next control
TAB	SHIFT		Select previous control
HOME			Select first control
END			Select last control
CTRL		Adjusting table columns	Push mode
CTRL	SHIFT	Adds a table column in front of an existing table column	Push mode
CTRL		Adds a table column after an existing table column	Insert mode

Key	Modifier	State	Operation
CTRL		Adjusting report dividers	Push mode
SHIFT		Dropping a control from within a variable palette	No title
ESC		A control is selected	Exit
ESC		Z-order display	Cancel
CTRL+↓		Moves you to the next row in a multi-marked table	
CTRL+↑		Moves you to the previous row of a multi-marked table.	
CTRL+PgDn		Moves you to the next page in a multi-marked table	
CTRL+PgUp		Moves you to the previous page of a multi-marked table	
CTRL+SPACE		Marks or unmarks a table row.	
SHIFT+PgDn		Marks the next page.	
SHIFT+PgUp		Marks the previous page.	
Shift+Down Arrow		Moves you to the next line and marks it.	
Shift+Up Arrow		Moves you to the previous line and marks it.	
ESC			Exit

Mouse Action Mapping

In the Form Editor, use the mouse to design and place controls to make the form look the way you want. Mouse actions are described in the table below.

Action	Modifier	Operation
Dragging a control		Displays bounding outline selection. Child controls are also selected.
Dragging a control	CTRL	Displays bounding outline selection. Child controls are not selected.
Click on control		Selects control. Child controls are also selected.
Click on control	CTRL	Selects a control. Child controls are not selected.
Click on choice control	SHIFT	Change layer
Click on form		Unselect control
Click on form	SHIFT	Drag the form (Class 0 forms only)
Double click on a control button		Control appears in the upper-left corner of the form
Double click on a control		Control properties are displayed
Double click on the form		Form properties are displayed
Right click		The context menu is displayed

DB Table Repository

eDeveloper automatically includes in the DB Table repository the Main table and the tables used by Link operations. These entries cannot be removed from the repository by deletion; the *Edit/Delete Line* (F3) option is not available when the insertion point parks on their rows. You can only edit their properties.

To access the Table list, zoom from the Table column.

You can include additional tables to be used and other called tasks. For example, a child task's DB Table list may be added to the parent task's DB table in order to save the time it would take to open and close tables each time the child task is called. In this case, these tables will also be listed in the DB Table repository of the child task.

When the cursor is parked on a Select Real table entry, the original name of the column is displayed at the bottom of the Task Execution repository.

Properties of the DB Table Repository

#

The sequential number automatically assigned by eDeveloper to the entries in the repository. It is not used for referring to the repository. You cannot edit this column.

Table

This column contains the repository's identifier number from the Table repository.

To add a new table, create a new line by choosing *Edit/Create Line* F4 and enter the identifier number of the table you want, or zoom to the Table list from the Table column to select the table that you want.

Name

The Name column shows the name of the repository. You cannot edit this column and the name is inserted automatically when the repository identifier number is specified.

Access

The Access property is relevant only in a multi-user environment. It determines how the repository is accessed by multiple users.

Use Read for Read Only access, or Write (the default) for Read and Write access.

For more details on the Access property, refer to the Access section in Chapter 23, Multi-User Considerations.

Share

The Share property is relevant only in a multi-user environment. It determines how other, concurrent tasks can access the table.

Use Read to allow other tasks Read Only access. Use Write (the default) to allow other tasks Read and Write access (effectively row locking). Use None for no shared access (Table locking).

The combination of Access and Share will be used by eDeveloper to determine the Open mode of the repository automatically. Thus, write access with share write implies row locking, while read and write access with share read implies table sharing. Share None implies exclusive use of the table. For more details on Access/Share mode, refer to the Share section in Chapter 23, Multi-User Considerations.

Open

Normal - the Open mode for a valid table for all normal purposes.

Damaged - The Damaged Open mode instructs eDeveloper to open the table even if it is reported as damaged by the underlying database manager. In this mode, eDeveloper will retrieve whatever rows it can from the damaged table. No indexes are available for the table during such an access. eDeveloper cannot guarantee retrieval of all the rows in this mode. Damaged Open mode should be used only for recovery attempts on the table.

- **Reindex** - The Reindex Open mode instructs eDeveloper to drop all the indexes of the table when opening it, perform processing, and rebuild the indexes upon session termination. Reindex Open mode is a performance-enhancing feature, for large table imports in Create mode, or for massive index value modification to a table. By removing all the indexes in sequential processing, eDeveloper saves the index-maintenance time normally required for every insert and modify index operation performed by the underlying database manager, such as balancing the B-tree in Btrieve. When the process ends, eDeveloper builds the index structure once for the whole table.

The Reindex Open mode depends on support by the underlying database gateway. Note that when specifying Reindex Open mode, if duplicate index values were entered in a unique index, the re-indexing process may give unexpected results. For instance, Btrieve simply aborts the re-indexing process. Other database gateways may decide to skip the duplicate rows. Refer to Chapter 25, SQL Considerations, for more information.

Fast - eDeveloper opens tables with the following default settings: Access=Write, Share=Write, Mode=Normal. If you wish to maximize integrity for the entire application but also maximize the performance for a particular table, set the Open mode to Fast. Note that Reindex mode is based on the Fast mode, which maximizes performance. If you want to maximize integrity, set the ISAM- Transaction environment property to Yes. Set the latter property to No when you wish to maximize performance.

Exp

A non-zero value here identifies an expression that will be evaluated to a DB table name at runtime. You can optionally include its logical name or explicit location (server/driver/directory) in the name. The evaluated table name is then used to redirect the table, from the default table name specified for it in the Table repository.

Cache

The Cache column is relevant for linked tables only. The setting of this property specifies if the linked table is cached or not. This setting should not be changed for the Main table.

The possible settings are:

- Yes - Use cache for the linked tables.
- No - Do not use cache for the linked tables.

Default value: Based on the cache strategy in the Table's Properties dialog.

Note that for linked tables there is no meaning for Position Cache. Therefore the Position Cache setting is considered as No Cache.

The default setting is taken from the Table Properties dialog as follows:

- Yes - If Position and Data was specified in the Table Properties dialog; or
- No - If Position or None was specified in the Table Properties dialog.

Identify Modified Row

You can have eDeveloper indicate to select rows that have been modified by another user, before updating your database.

eDeveloper can identify and select records according to the following options:

- Position
- Position and selected fields
- Position and updated fields
- As table

This property applies only for deferred transaction mode tasks and SQL tables.

I/O File Repository

In the I/O File repository you define all INPUT and OUTPUT devices to be used for reading or writing during the current task. Each I/O file must be defined before you refer to it in an Output or Input Form operation.

All subtasks, nested to any level, have access to any I/O file defined in their parent tasks. The I/O File repository of a subtask shows all the I/O files opened by its ancestors.

The IO devices defined in the Main Program are accessible to all other programs and their subtasks. The meaning of each repository property is presented below, including the options available in each property.

Properties of the I/O File Repository

#

This property contains an automatically generated sequential number used by eDeveloper as the I/O File identifier. You cannot edit this property.

Name

A descriptive name. The I/O File name in this field is used for documentation and prompting purposes only.

Media

Defines the physical support for the file: its destination as an output file or its source as an input one. The possible values for Media are:

- Graphic Printer (default) - The I/O based on this I/O file definition creates graphical information that can then be directed to a graphic printer or to a disk file.
- Printer - The file is formatted for printing and is directed to one of eDeveloper's logical printers. You can redirect a printer file to a disk file by defining an expression, in the Exp property, which evaluates to a valid file name at runtime.
- Console - The file is formatted for display and is redirected to the console. The dimensions of the console display area are defined according to the eDeveloper MDI window dimensions. The display area width will be congruent to the width of the form if it stretches beyond the eDeveloper MDI window width.
- File - Defines either an input or output disk file, depending on the value of the Access property. The file's path and name are specified by the expression referred to in the Expression property.

- Requester - The Requester I/O device is used by batch tasks of an application that are invoked by a Call Remote operation or an internet request. The output to this I/O device is transmitted back to the calling task or browser upon completion of the task that opened the device.
- XML Direct Access - This is used to query an XML file that can be queried using the XMLCnt, XMLExist, XMLFind, and XMLGet functions. You can also create, modify, or delete an XML element or attribute by using the XMLInsert, XMLModify, and XMLDelete functions.
- Variable - This is used to send and receive a text variable for input and output forms. This option can be used for the exchange of XML files. You can use the XMLCnt, XMLExist, XMLFind, and XMLGet functions when the Access column is set to Read. You can use the XMLInsert, XMLModify, and XMLDelete functions when the Access column is set to Write.

It is not necessary to specify an expression for the Graphic printer, Printer, Console media, or Requester. However, when an expression is specified, the output is directed to the file defined by the expression.

Printer

This property is relevant only when you specify Media as Printer. It refers to one of the logical printers defined in the Printer repository.



For more information, refer to Chapter 2, Settings.

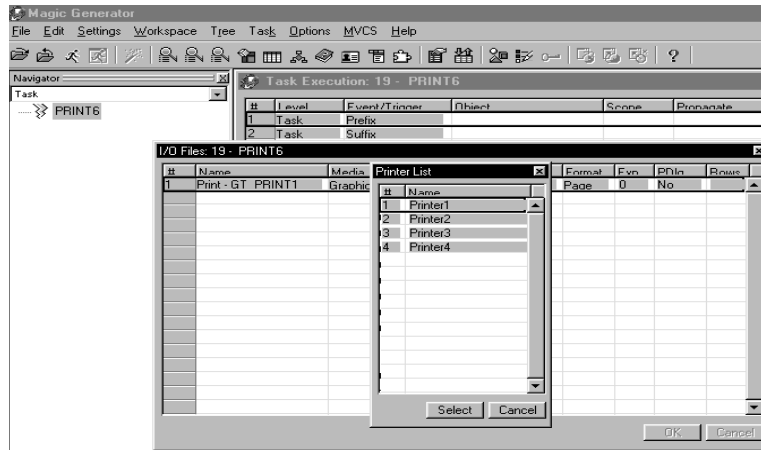


Figure 6-6 A Printer List

The value for the Printer property is the name that will establish a match between the printer specified here and one of the printers defined in the Printer repository. The default is the first printer in the Printer repository. To choose another printer, zoom to the Printer list and select the printer you want, or type in the name of the printer.

Generic error messages are displayed at runtime to indicate various error situations such as Printer not Connected, Printer Power Off, Printer not Selected, and Printer Out of Paper. Printer error messages are followed by the Confirm Cancel Operation dialog. Answering Yes causes eDeveloper to continue with program execution until it reaches the end of the data, ignoring any further output operation. At the same time, eDeveloper raises the EOF (end of file) status for the file. The EOF status can be used to terminate task execution by conditioning task termination (End Task) to the return value of the EOF function. A No value to the Confirm Cancel Operation dialog causes eDeveloper to retry the print operation. If the error is not cleared, the Confirm Cancel Operation dialog will reappear.

Access

Defines the mode in which the file will open.

The possible values are:

- Read - tells eDeveloper to open the file in Read mode. You can define Input Form operations for the file in the Task Execution repository.

When you open an XML document by defining an XML Direct Access or Variable I/O media type, you can query the inserted XML element or attribute without saving the XML document.

- Write - tells eDeveloper to open the file in Write mode. You can define Output Form operations for the file in the Task Execution repository.

After the task is closed, changes to an XML document are saved, after the task is closed, when the XML Direct Access or Variable media type is set to Write access. For information about making changes to an XML document, see the XMLInsert, XMLModify, or XMLDelete functions in Chapter 8, Expression Rules.

If the file you want to open is an output file whose name is defined by an expression, its content will be erased.

- Append - tells eDeveloper to open the file in Write mode. You can define Output Form operations for the file in the Task Execution repository. Output forms can send a text variable only if the Access property for the Output form is set to Append.

If the file you want to open as an output file, whose name, defined by the expression Exp, already exists, all new forms written to it will be added to the end of the same file.

Format

Defines which control characters are to be automatically written to an output file or are expected in an input file.

Allowable settings for Format are:

- Page - means that Top of Form and End of Line control characters are requested for the file. Top of Form control characters will appear when a page ends and End of Line control characters will appear for every line.
- Line - means that only End of Line control characters are requested for the file. They will appear for every line.
- None - means that no control character is requested for the file.

Exp/Var

Defines the number of an expression in the Expression Rules repository to be used at runtime to define the file source or destination for the operating system.

You can use the expression to define any valid operating system file name. Optionally you can include server/path to the file name or use a logical name in the expression. In these cases, eDeveloper will read/write according to the specified criteria. The expression is evaluated in the task initialization stage, and cannot change dynamically.

If the Media column is set to Variable, then you can type in a variable code or zoom from the Exp/Var column to the Variable list. Only BLOB variables defined in the Record Main can be selected in the Exp/Var column when the Media column is set to Variable.

PDlg (for Print dialog)

The valid values for the PDlg (for Print dialog) property are Yes, No, or an expression. The PDlg property is relevant only for Media=Graphic Printer. Yes specifies that when the task opens the I/O file, a Windows Print dialog will appear allowing the end-user to change print properties. No specifies that a Windows print dialog will not appear, and that the print properties will be transferred directly to the printer without end-user interaction. The expression allows dynamic selection of the Yes/No setting. The Print Dialog property default value is No.

Rows

This property is relevant only for Printer media and Console where Page format output is selected.

It defines the number of rows or lines per page (i.e., the number of printed lines between two successive Top of Form control characters) in the output file. The default value depends on the specific printer used. For more information refer to Printer Repository in Settings. The default value is 0, which produces an output page of 60 lines, including a 3-line header area and a 3-line footer area.

The default may be changed here, in the Rows property. The default may be changed at runtime by the end-user in the I/O File repository, provided you set Allow I/O Files to Yes in the Task Control dialog.

I/O Properties Dialog

Page Size

Page size defines the page (paper) size for a graphic printer.

Page Header Form

The Page Header Form specifies one form from the Form repository to be used as the page header. This form will be the automatic output for every new page as the page header.

Page Footer Form

The Page Footer Form specifies one form from the Form repository to be used as the page footer. This form will be the automatic output for every new page as the page footer.

Copies

The Copies property specifies the number of copies to be printed by the graphic printer.

Note: The Copies property is only enabled when the Graphic Printer is selected from the Media field.

Expression (Exp)

The Expression property holds an expression number, defined in the Expression Rules repository, that determines the number of copies to be printed from a graphic printer.

Note: The Copies property and The (Copy) Expression property cannot be enabled at the same time.

Orientation: Portrait, Landscape

The Orientation property determines whether the copies will print from a graphic printer in portrait or landscape orientation.

Print Preview

The Print Preview option redirects the printer output to the Print Preview system. The valid values are Yes, No, and an expression that returns a logical result. The Print Preview system allows the end-user to preview the output of the report, choose a printer, and to print.

Note: Print Preview is available only for graphical printers.

I/O Name to Use

The I/O Name to Use setting allows you to specify an open I/O from another program to be used by the current program.

Zoom to the Expression Rules repository to enter an expression that evaluates to the name of an open I/O. In runtime, eDeveloper will find an open I/O file of the name specified in this setting. If such a file is found, it will be used by the current task.

Note that in printing a report, the page header and footer are from the task that originally opened the I/O file. In addition, only the current printed task header forms are printed automatically. The header of the task that originally opened the I/O is not printed automatically.

Character Set to Use

This property is only relevant for File media.

When this property is set to ANSI, eDeveloper will not perform any translation of the character set, since the eDeveloper uses the ANSI convention internally.

If this property is set to OEM, eDeveloper will convert the data from ANSI to OEM using the OEM2ANSI translation file. If no file is defined, eDeveloper will use the operating system OEM to ANSI conversion.

Visual to Logical Translation

This property controls conversions from visual to logical for the processed data. Applies only to File and Printer media. This property applies to right-to-left applications only.

When set to Yes, the output process of each field is converted from logical to visual. The entire line is converted from visual to logical. The default value is Yes.

When set to No, no conversion occurs.

You can set an expression from the Expression field. It is required that the expression return a Boolean value.

Flip Line

This property controls the flipping of a line as it is exported or imported. It applies to right-to-left applications only.

When set to Yes, the line is flipped. When set to No, the line does not change as it is exported or imported.

The Flip Line property is enabled only when the Visual to Logical Translation property is set to Yes or evaluates to a True value.

Sort Repository

The Sort repository shows a compound window that includes:

- Segments area on the left-hand side of the window.
- Variable list on the right-hand side of the window.

Properties of the Segment Area

The order in which the segments appear in the repository reflects their hierarchical strength, from major to intermediate to minor. This order will control the sorting sequence.

Fill in the properties of a newly-created row or edit an existing one according to the guidelines described below.

(for Segment identifier)

This property contains an automatically generated sequential number used by eDeveloper as segment identifier. You cannot edit this property.

Var (for Variable #)

The letter identifier of the Real, Virtual, or Parameter variable that will be used as a segment.

You can type the letter identifier you want or you can zoom to the Variable list on the right to select the variable you want. Once you are in the Variable list you can zoom to the Column repository for additional information.

Variable Name

This property shows the name of the variable chosen as a segment. You cannot edit this property.

Size

You can edit this property only if the property attribute of the chosen variable is an alpha type. You can then shorten the size of the segment field. The leftmost characters included in the new size are used as the index. This property is for performance and memory considerations. Set the size according to the number of significant characters for the sort. Including characters beyond this size is unnecessary. If the index segment is shorter, the sort process is faster, and the sort file is smaller due to better utilization of index space. Some SQL databases do not support sorting by part of a field.

Direction

This property indicates the direction of the sort, according to the active collating sequence.

Each database you can use with eDeveloper may have its own limitations concerning, for example, the number of segments or the maximum size allowed for an index. Sort files are created using the Database for Sort/Temporary, defined in the Environment dialog.

Sorting in SQL databases may use the Sort Using RDBMS feature. For more information, see the Programs section under the Sort repository in this chapter.

Sort Type

The Sort Type property allows you to indicate to eDeveloper that the columns used in the sort are unique. This setting indicates to eDeveloper that there is no need to add a position value to the Order By clause for SQL tables.

- Unique Sort - indicates that the columns used in the sort are unique. The SQL gateway will not add a position value to the Order By clause.
- According to Index - indicates that eDeveloper will determine the uniqueness of the sort by scanning all known Unique Sort indexes. If the sort segments contain a unique index, the sort will be defined as unique. Otherwise, the sort will be defined as non-unique.

The default value is According to Index.

Sort Using RDBMS

When using SQL databases, the RDBMS engine can sort the rows dynamically by using the ORDER BY clause without the need to define an index or an index as required in ISAM files. Sort usage in eDeveloper remains unchanged. When all columns are sorted as part of the main and joined tables, however, eDeveloper allows the RDBMS to sort. When it is not possible for the RDBMS to sort, eDeveloper sorts the rows by using a temporary file in the Sort Database defined in the Magic.ini file.

Sort usage in a task is the same as defining a virtual index for that table. In addition, it minimizes the need for you to define additional virtual indexes to anticipate the index needs of your end-users.

When defining a sort, eDeveloper reissues the Select statement for the task, and replaces the ORDER BY clause that was generated according to the Index with the column names used in the Sort as long as the following conditions are met:

All sort segments are part of the main or linked tables (only with Link Inner Join or Link Left Outer Join).

All sort segments are natural data types in the database. This means that they are mapped to data types to which the database can sort. For example, MSSQL cannot sort according to a BIT data type, therefore sorting by a sort segment that is mapped to BIT in the database will not use the Sort Using RDBMS feature and will create a temporary sort table in the Sort Database defined in the Magic.ini file.

eDeveloper always requires a unique ORDER BY clause. If you do not specify that the ORDER BY clause is unique, and if eDeveloper does not determine that the sort segments in combination are unique, eDeveloper will add the position index segments to the ORDER BY clause. In addition, if any part of the columns selected in the sort were not part of the main and join tables, eDeveloper creates a temporary sort file.

When you define a sort that results in ORDER BY clause, the RDBMS optimizer might not use an index to perform a Query, but might prefer to use a different access method to perform the query in a more efficient way.

Event Repository

In each task you may define user-defined events to be handled in your application whenever these events will be raised or triggered.

A user-defined event is a logical definition of an occurrence, for example you may define an event called 'Add Customer'. The event definition by itself will not effect the application. This logical definition allows you to define the handling for such a logical occurrence.

Properties of the Event Repository

#

This property contains an automatically generated sequential number used by eDeveloper as the event Identifier. You cannot edit this property.

Description

This is a descriptive name of the event. The event name is used to select this event when a handler is defined for it.

Trigger Type

This is the pre-defined event type you wish to be used as a trigger for an event.

The available options for this property are:

- System
- Internal
- Time
- Expression
- None

When the trigger type is set to None (i.e. No trigger), this event may be raised only through the Raise Event operation.

Trigger

This property only applies when you have set a trigger type other than None. In this column you may zoom to define the exact event you wish to trigger a specific user-defined event.

- For a system type event, the Key Definition dialog will appear for you to enter the key combination to trigger the event.
- For an Internal type event, the eDeveloper Action list appears to let you select an action.

- For Timer type event, you will be able to set the time interval.
- For Expression type event, you will be able to zoom to the expression editor to define or select the required expression.

Force Exit

This property allows you to define the level from which the should exit before performing the handler of your defined event.

- **None** – The event does not force an exit to any level and the handler will be executed whenever it is reached in the engine flow.
- **Editing** – The event instructs the task to exit the edit mode of the current control before executing a corresponding handler. On handling such an event, the engine:
 - Exits the Edit mode
 - Updates the variable with the edited value
 - Recomputes all the values related to the updated variable
 - Executes the Control Change handler if the data was modified
 - Executes the Event handler
 - The engine returns to the Edit mode

When referring to the control's variable from the executed handler, the engine displays the last edited value as it is updated in the variable.

This option is required primarily for Edit, Rich Edit, and Multiple Selection List Box controls, and controls that have an editing state that differs from the variable update value.

Control – The event instructs the task to exit the current control before executing a corresponding handler. On handling such an event the following sequence will take place:

- The engine exits the Edit mode
- Updates the variable with the edited date
- Recomputes all the values related to the task

- Executes the Control Change handler
- Executes the Control Verification and Suffix handlers.
- Executes the Event handler
- Executes the Control Prefix handler
- The engine returns to the Edit mode

Record – The event instructs the task to exit the current record before executing a corresponding handler. On handling such an event the following sequence will take place:

- The engine exits the Edit mode
- Updates the variable with the edited date
- Recomputes all the values based on the variable
- Executes the Control Change, Verification and Suffix handlers
- Executes the Record Suffix if the record has been modified
- Executes the Event handler
- Updates the record in the table if the record was modified
- Executes the Record Prefix
- Executes the Control Prefix
- The engine returns to the Edit mode.

Public Name

For the Main Program, you are required to provide a public name for the user event because eDeveloper will use the public name of the user event and not of the program when searching for a corresponding handler.

Expose

When you select the Expose check box, programs from other components can call the Main Program event in the host application by using the Raise Public Event operation. When Expose is not selected, the event cannot be called by other loaded applications.

Range and Locate Properties

Use the Range/Locate properties dialog to specify ranges on the records retrieved in the task and to locate on a specific record.

The Range/Locate properties dialog is divided into two tabs:

- Range/Locate tab
- SQL Where tab

Range/Locate Tab

Range Expression: 0 (default)

The Range Expression property is one of the means of defining a task's dataview. The rows are initially included in a dataview based on the Range Lower/ Upper limit expressions specified in the task's Select operations. The Range expression allows you to refine the Range criteria further, and base them upon more complex and dynamic conditions. The Range expression is evaluated for every fetched record.

- If the Range expression evaluates to True, the row is included in the dataview.
- If the Range expression evaluates to False, the row is skipped.
- If a Range expression is not specified (that is its number is zero), it evaluates to True.

A Range expression is required when:

1. the criteria depend on values contained in variables not available at task initialization time, or
2. the range criteria are not definable by the lower limit and higher limit expressions of the Select operations.

Note: The Range expression is not sent to the database – eDeveloper filters the records that were brought from the database.

Range Order: Ascending (default)

The Range Order property value determines whether the fetch direction for the range of rows to be included in the dataview is ascending from the first row to the last, or descending from the last row to the first.

Locate Expression: 0 (default)

A Locate Expression is used to position the dataview to start from a specific row when a task starts running. When a task begins, eDeveloper scans the dataview from its beginning until the Locate Expression evaluates to True for a row. That row is fetched and the cursor is positioned on it. The user may then freely travel back or forth within the dataview using the direction keys.

Notes:

1. The Locate Expression property is relevant for online tasks only.
2. The Locate Expression property should be used in cases where the locate condition is not based on a single contiguous range, or when the search criteria are complex. Otherwise, you can use the Select operation Locate Lower/ Upper limit expressions, which will run faster.
3. All variables included in the Locate Expression are evaluated at the task's initialization; therefore they may contain only property variables or variables of ancestor tasks.
4. If a Locate Expression is not specified, the cursor is positioned on the first dataview row fetched.
5. If the starting row cannot be found when an Online task starts execution, an appropriate warning message is displayed and the cursor is positioned on the first row of the dataview or on the next closest value, if one exists.

Locate Order: Ascending (default)

This property determines whether the search direction for locating the row in the dataview is ascending from the first row to the last, or descending from the last row to the first. If the last row matching particular criteria is required, specify descending. For example, if the dataview was ordered by date, you

could use Locate Order: Descending to position on the last transaction for that date.

Note: The Locate Order property may be used even if no Locate Expression or Locate Lower/ Upper limit criteria are found.

Position: 0 (default)

This property provides a method to display a record according to its position, or a set of records beginning from a specific position. This property is relevant only for the Main table of the task. The linked tables cannot be ranged or located according to their position.

The property accepts a BLOB expression indicating the position of the record from which you want the task to start or locate.

The Position property is enabled only for tasks with a Main table.

It can accept an expression that returns a BLOB expression which holds a record's position.

There are 2 functions in eDeveloper V9 that return a record's position:

- CurrPosition

This function will return, as a BLOB expression, the internal position of the current record of the Main table of the task represented by its depth.

- ErrPosition

This function will return, as a BLOB expression, the internal position of the record on which the most recent error occurred.

For more information on these functions, refer to Chapter 8, Expression Rules.

Usage

The Usage property is enabled only when the Position property is not 0. Once the Position is selected, the Usage property will enable selecting from these options:

- Range On (default)

Indicates that only the specific record should be displayed - eDeveloper will perform a get current operation, and no open cursor.

- Range From

Indicates that all records starting from the position record will be displayed. eDeveloper will perform an open cursor with this position as start position.

Note:

For both these range options, there will be an AND operation between the range in the Select operation, the range expression on the task and this condition.

For both these range options, if this record does not exist – no records will be displayed.

- Locate

Indicates that all records (according to all other task ranges) will be displayed, and eDeveloper will locate on the record with the specified position.

There will be an AND operation between the locate in the Select operation, the locate expression on the task and this condition.

If this record does not exist – the locate operation will fail (“Record not found – positioning at beginning” warning message is displayed).

SQL Where Tab

SQL Where Range Expression: 0 (default)

A DB SQL Where range is not allowed for tasks that are executed within a deferred transaction. To deliver the SQL Where range functionality for deferred transactions, a new range was added called ‘SQL Where range’.

It is similar to the existing DB SQL Where range with one exception – it is defined using a subset of eDeveloper expressions (unlike the existing DB SQL Where range, which uses free format text). The Magic SQL Where range may only include those expressions that the engine can translate to SQL.

For example, assuming

A is real column with DB name Employee.BirthDate then writing the Magic SQL range expression `A = Date()` will be displayed in the Full SQL Where area as

`Employee. BirthDate = Date()`

(the table name will be added only when link join is involved)

Note that the Date function is used, not a DB-specific function.

This will be translated in runtime with Oracle to

`TO_Date(Employee.BirthDate, 'DD-MON-YY') = TO_Date(SysDate, 'DD-MON-YY')`

and will translate in runtime with MSSQL to

`CAST(CONVERT(CHAR, Employee.BirthDate,112) AS DateTime) =
CAST(CONVERT(CHAR, GETDate(),112) AS DateTime)`

The same task was executed without any change in the syntax of the expression against 2 different databases. eDeveloper translates the SQL Where expression to the appropriate syntax per each database.

Note: For a list of supported eDeveloper functions for the SQL Where expression, please refer to the SQL Considerations chapter.

DB SQL Where

DB SQL is the same SQL Where Clause feature that was introduced in Magic Version 8. This property is available for Physical Transaction mode tasks with SQL Main table only. In Deferred\Nested Deferred transaction mode tasks this property is disabled.

The purpose of the DB SQL feature is to enable the programmer to use the SQL- specific Where clauses (in addition to the Where clauses generated automatically by eDeveloper) without the need to use the Direct SQL tasks, and to view the Full Where clause that is generated by eDeveloper.

This is an interface for writing Range based on SQL syntax using eDeveloper columns. This range is added to the Magic Where clause that is defined in the Record Main.

You can zoom to the column list from the DB SQL field. The variable list is a list of all the variables available to use in the DB SQL property:

- Real columns from the main and joined tables - will be replaced with their DB column name.
- Variables from parent tasks - will be replaced with their values.
- Virtual and other real columns from the current task - will be replaced with their values.

The DB SQL refers to a free text form in which the DB SQL Where clause is written. Three types of strings can be written in the DB SQL area. They are as follows:

- A column number (A, B, C, etc.) prefix with a ':' sign.
If the column is from either the main or joined tables it will be replaced by its DB column name. Otherwise, it will be replaced by its value according to its attribute. For alpha columns, eDeveloper will suppress any trailing blanks, and add quotes to it.
- A column number (A, B, C, etc.) prefix by '@:' sign and is an Alpha column not from the main or joined tables. It will be replaced with its value without adding quotes to it.

When eDeveloper replaces a column with its contents, eDeveloper checks the column's attribute and storage, and if necessary adds quotes - as in the case of alpha strings. In order to prevent eDeveloper from adding quotes to the alpha columns, enabling any type of syntax to be written, the @ character should be added as a prefix to the column.

For example, assuming

A is real column with DB name Employee. jobname, and B is a virtual alpha column with description Vjobname, and its value in runtime is "AB" then writing the DB SQL range

:A like 'B%' ' will be displayed as

Employee. jobname like 'B%' '

(the table name will be added only when link join is involved)

and will translate in runtime to

jobname like 'B% '

Writing the DB SQL range

:A like :B will be displayed as

Employee. jobname like ["Vjobname"]

(the table name will be added only when link join is involved)

and will translate in runtime to

jobname like 'AB'

Assuming that C is a virtual column, and its Description (Name) is Voperation and its value in runtime is "like" then writing the DB SQL range

:A @: C :B will be replaced with

jobname [Voperation] [" Vjobname"]

and will translate in runtime to

jobname like 'AB'

Full Where Clause

eDeveloper displays the entire WHERE clause as used in the SQL statement generated by eDeveloper in the Full Where Clause area. Click the SHOW button at the bottom of the screen to refresh the Full Where clause.

The Full Where clause consists of three parts:

1. The Where clause expressions from the Record Main (consist of ranges on the Select operations and the Link Inner Join/Left Outer Join).
2. The Magic SQL Where clause, added in parenthesis with an AND clause, to concatenate it to the Record Main Range.
3. The DB SQL Where clause, added in parenthesis with an AND clause, to concatenate it to the Record Main Range. When running a task within a deferred transaction the DB SQL Where clause is ignored.

This part of the Where clause will be added only when the task Transaction mode is not Deferred\Nested Deferred.

The Full Where clause replaces every occurrence of a column from the column list. Real columns from the Main and Joined tables, are replaced in the format of 'column DB name' or 'A. column DB name', and virtual fields are replaced with their description.

The Full Where clause is added to every SELECT statement. The variables' values are evaluated only once when entering the task, and no re-compute is done.

The only time in which the DB SQL range or Magic SQL range is not used is when eDeveloper generates the SELECT statements for the internal *Get Current* and *Hook* (determined by the locking strategy) operations. In these operations eDeveloper uses the position key to get the record, and the SQL range is not needed.

Magic SQL Where and DB SQL Where Behavior

When a dataview is created for a task, eDeveloper generates a Select statement with a Where clause. The first part of the WHERE clause is based on the From and To expressions of the Select operations, in the Record Main. The second part of the WHERE clause is the Link Inner Join/Left Outer Join condition. The third part is the Magic SQL Where, which is added to every SELECT statement. The forth part is the DB SQL Where, which is added to every SELECT statement The column's values are evaluated only once when entering the task, and no recomputing is done. The only time that the SQL Where is not used, is when eDeveloper generates the SELECT statements for the internal *Get Current* and *Hook* (determined by the locking strategy) operations. In these operations, eDeveloper uses the position index to get the row, and the SQL where is not needed.

Magic SQL Where and DB SQL Where Usage Considerations

- When using virtual columns from the current task in the SQL range, the virtual columns must receive their values from the calling task's properties, otherwise their default values will be used.
Note: The *init* expression is calculated afterwards therefore it cannot be used.

- BLOBs and Memo fields cannot be used in the SQL range.
- eDeveloper does not check the syntax of the SQL range. Therefore, if an invalid SQL syntax is used, a message from the RDBMS will occur in runtime.
- eDeveloper does not check the attribute of each column used in the SQL Where string. It tries to convert each attribute to a string which is concatenated in the corresponding location in the SQL Where string. Therefore, it is important to use the attributes properly.
- When the Link Join operation is used, eDeveloper uses an alias for the Table names. Therefore, instead of Table1.Column1, it uses A.Column1. If a string is written in the DB SQL which contains a column name, the alias letter corresponding to the Table should prefix the column name.
- eDeveloper suppresses any Trailing blanks.
- Note that both the DB SQL Where range and the Magic SQL Where range are both computed once, before the task prefix.

Task Execution Repository

The Task Execution repository is a compound window that includes:

- The repository of the execution handlers at the top of the window. This portion of the window is referred to as the Handler repository.
- The Operation repository is related to the currently highlighted level, and appears at the bottom of the window. The title of the Operation repository includes the name of the execution level to which the operations are associated.

As you navigate the Operation repository, zoom to specify the properties of the current operation. Once you have entered a value in an Operation field, whenever the cursor parks on the property its content is displayed at the bottom of the repository.

The Structure of the Handler Repository

All tasks have built-in Record level handlers, containing a Record Prefix, Record Main, and Record Suffix, and built-in Task level handlers, containing a Task Prefix and Task Suffix. Tasks can also have Control levels containing a Control Prefix, Control Suffix, Control Verification, and Control Change, and Handler levels with defined events. Batch tasks can have an optional Group level.

- The Record Main is used to define a task's dataview and usually requires a related Operation repository to be filled in. The exception is in the case of subtasks that use the Scratch File as their Main repository, and are used for the manipulation of properties previously selected by any of their ancestor tasks or for operations that are not dataview related.
- Record Prefix, Record Suffix, Task Prefix, Task Suffix, Control Prefix, and Suffix Operation repositories can be empty.
- A control handler can only follow a record handler. Control Prefix places the operations that the engine executes before a specific control is entered, while Control Suffix places the operations that the engine executes after exiting a specific control.
- At the Handler level, you assign a handler for a defined event for a specific control type that can be implemented between the Task and Record levels, or within Group and Control level handlers. The Handler level shares the same properties as the Control level, but allows you to select defined events while the Control level lets you select pre-defined events.
- Batch tasks, primarily report-producing tasks, may include additional levels in the Handler repository. These levels are called Group levels, and they create control breaks. Each control break is executed when there is a change in value of a variable specified in the Handler repository as a Group Level variable.

The major control break - the one that occurs least frequently- should be positioned immediately below the Task level in the repository. The minor control break, occurring most frequently, should be the last Group Level entry in the repository, just above the Record Prefix.

For further details about the contents of the levels and their relationship to execution flow, refer to Chapter 5, Application Engine.

Handler Repository Properties

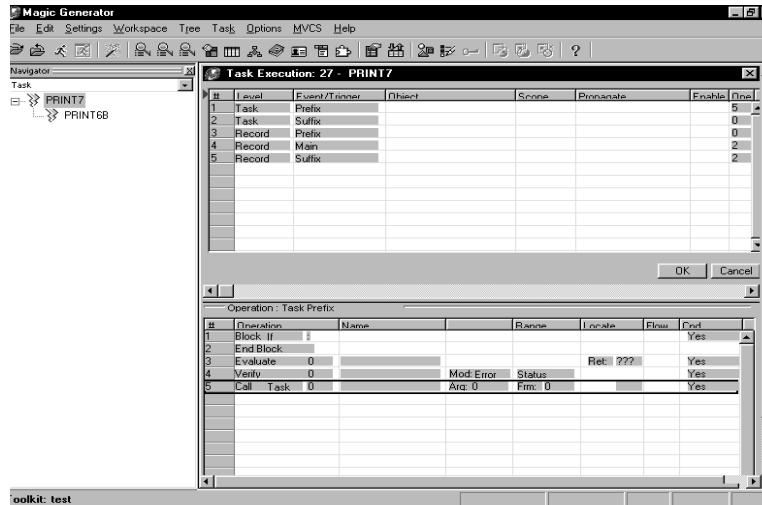


Figure 6-7 Execution Level Repository

#

The # property shows the sequence number of the execution level.

Level

Indicates the level (Task, Record, Group, Handler, or Control). You cannot edit this property. Control lets you select a pre-defined event. Handler lets you select any other event. The Group level is used for Batch events only and the pertinent task type must be indicated in the Task Properties dialogue before the Group level can be selected.

Event

You can only define an Event for the Control and Handler levels. Event for the Task or Record level are hard-coded into the eDeveloper engine and cannot be altered.

Control events are:

- Prefix - The operations that the engine must execute before the insertion point is moved to a particular control.
- Suffix - The operations that the engine executes at the end of the control.
- Verification - The operations that the engine executes when the insertion point is taken away from the control and the control is passed through in Fast mode, before the Control Suffix level.
- Change - Performs the operations that the engine executes when a control variable is changed.

The Handler level for online task types can be:

- Internal - An eDeveloper internal action from the eDeveloper action list. For more information, see the eDeveloper Action section below.
- System -These are events that are triggered by defined keystroke combinations.
- Timer - Events that occur at a defined interval.
- Expression - Events that are triggered when an expression evaluates to True.
- User - Events defined in the Events repository.
- Error - An error event raised whenever a database operation produces a database error.
- ActiveX - An ActiveX event raised for COM objects.

Details

Group Level – This level is only used for batch programs, to perform operations when one or more fields of the active key, change. Zoom from the Details column to select a variable from the Variables list.

Control Level – For Control level handlers you must specify the control on which this handler should be performed. You may zoom from this column to the controls name list, to select the relevant control by its name.

Handler Level – Like the Control level, you may select a control from the controls name list, to have this handler perform only if the defined control is parked at the time in which the event is handled.

However unlike the Control level, this property is not mandatory. If you do not define a specific control name than this handler will be executed for all controls.

Error Level – For the Error level you may define two details.

- Directive - you may choose from a list an engine directive and by that define what the eDeveloper engine should do after the error was handled.
- Msg - Where you may define if you wish eDeveloper to present its default message box notifying of the error. This detail may accept a Yes, No and expression values.

Scope

Scope is defined as one of the following for Handler levels:

- Task - The event is executed only in a single task. If the event is triggered in a subtask, the handler ignores it.
- Subtree - The event is executed in the task or any of its subtasks.
- Global - Option available for a Main Program handler. This is relevant for applications that are to be used as eDeveloper components.

Propagate

Propagate is active for Handler levels.

- Yes - When you want task event to be taken to the next handler level.
- No - Prevents the event from going to the next handler level.

Enable

Relevant for only the Control and Handler levels. This setting determines whether the handler is enabled for the selected task.

Operation

The number of operations in the Control Prefix or Control Suffix Operation repository.

Main Program

The Main Program serves as a main dispatch program from which every other program will run. This makes the Main Program active for the duration of an application's execution. The Main Program is the first to open as the application is opened and the last to close.

In the Main Program you can define the following:

- Variables
- Events
- Handlers
- Forms

These can then be accessed and used by any other program in the application. These resources are rendered global and can be applied to the entire application.

Main Program Access and Usage

The Main Program is implicitly executed when:

- An application is opened while the eDeveloper engine is in Runtime mode.

- A user, after opening an application in Toolkit mode, switches to Runtime mode.
- Running a program in Toolkit mode using either the Execute Program action from the Program repository, or the Generate Program action from the Task repository. In either case, the Main Program will run before the current program.

The Main Program is used for any or all of the following:

- As a controlling mechanism through an application's Task Prefix and Suffix, the Main Program determines the start and end of application handlers.
- Setting and maintaining global variables throughout an application.
- By defining global event handlers, the Main Program fashions a generic handling for applications.
- By controlling global events, the Main Program sets user-defined events.

Main Program Characteristics

Runtime Characteristics and Behavior

Dataview

Unlike other programs, the Main Program does not have a Main table. It can have virtual variables that are initialized at the beginning of the application's execution, and are in use during the execution of an entire application.

To allow you to open tables as soon as the application opens, the tables can be opened in the Main Program. Specific records can be read through the Link operation, and thus used throughout the application. Tables can also be specified in the DB Table repository – these tables will be only opened and not read.

Records fetched by Link operations in the Main Program cannot be updated. If an update will be attempted it will be ignored, and the particular task update phase will not exist. Therefore, the Main Program does not execute any record

locking or transaction managing commands. Table locking, however, will be available for the tables that are opened by the Main Program.

Flow

The Main Program executes as a batch task whose record cycle is performed once. This behavior is not affected by Task Ending properties. As an application opens, the Main Program's tables open and its Task Prefix is then executed. The Main Program's virtual variables are then initialized and its link records read.

The Main Program's Task Suffix is executed just before an application ends, when the application is terminated. The Main Program's tables will also close. Like any batch program, any operations that are not relevant for batch tasks but appear in the Record Main, will be ignored.

Subtasks

The Main Program can have regular subtasks, executed as a regular task. These subtasks have Main tables and functions like regular tasks. However, their variables are not global variables. Only the Main Program's root task variables are global.

Recomputation

The Main Program behaves like any other program in adhering to the rule that does not allow for the recomputation of variables from parent tasks that are updated from subtasks.

Call Expression

The Main Program cannot be called explicitly through a Call Program operation. If the Call operation is a Call Expression, and the expression in runtime evaluates to 1, an error message appears, and the Call operation will not be executed.

End Condition

When the End Condition of the Main Program evaluates to True, the whole application will stop running.

Toolkit Characteristics and Behavior

Program Repository

The Main Program will always be listed as the first program in the Program repository. It is automatically generated whenever a new application is created, and its default name will be Main Program.

You cannot delete, repeat or overwrite the Main Program. In addition, you cannot execute the Main Program – the Execute Program option will be disabled while the cursor is parked on the Main Program row.

Program Properties Dialog

The Main Program Properties dialog has the following settings:

- The Task Type is set to Batch, and is disabled.
- The Initial Mode is set to Query, and is disabled.
- The End task condition is set to No.
- Allow Events property is set to No and is disabled.
- Because the Main Program does not have a Main table, the related properties will be disabled.
- The Locking Strategy is automatically set to No Lock and is disabled. All other properties in the Enhanced tab keep their current settings and are disabled.
- All the properties in the Task Properties Advanced tab are disabled.

Task Control Dialog

The Main Program Task Control dialog has the following default tab settings:

- Modes tab: all the properties, except the Allow query property, are set to No and disabled.
- Behavior tab: The Open Task Window property is set to No. All other properties in this tab will be disabled.

Program's Form Repository

The program is created by default with a class 0 form. You can always add a form, but this form will not have a system menu.

Forms of a class greater than 0 are available for output for all the application's programs.

DB Table Repository

Although you cannot define a main table for the Main Program's main task, you can defined tables in the Main Program's DB Table repository.

The default table access type in the Main Program is set to Read, and is disabled - no locking is done in the Main Program. The default open mode is set to normal, and is disabled.

The Cache strategy is set to No and is disabled for Main Program tables.

I/O File Repository

The Main Program does not have any I/Os. The I/O File repository is therefore disabled.

Task Execution Window

The Main Program will have Task Prefix, Task Suffix and Record Main built-in handlers. The Record Prefix and Record Suffix handlers do not exist for the Main Program. Any other user-defined handler can be set to be used throughout the application.

Main Program Variable Availability

The Main Program's variables are available for all programs in the application. In the Variable list they appear before any other program's variables and will be listed alphabetically.

The variables names consist of 3 characters. When a variable is selected, its name is automatically displayed.

MVCS

Because the Main Program's variables are used throughout the application, MVCS regards it as a separate entity, thus allowing for shared access.

Modification of the Main Program resources can only be done exclusively - as is the case of tables and programs.

Whenever an object that uses the Main Program is being edited, the Main Program cannot be checked in. When in the latter case you try to check in the Main Program, the "Main Program already locked" error message appears.

Call Operation

The Main Program will not be available for Call operations – you cannot explicitly call the Main Program.

Select Program

The Main Program, because it cannot be called explicitly, will not be available for the Select Program property for column and Model definitions.

User Events

When you select the Expose check box, programs from other components can call the main program event in the host application by using the Raise Public Event operation. When Expose is not selected, the event cannot be called by other loaded applications. You are also required to provide a public name for the user event because eDeveloper will use the public name of the user event and not of the program when searching for a corresponding handler.

Regardless of whether the Expose property is selected, the user event can be raised by a program in the same application by using the Raise Public Event operation.

For more information about the Raise Public Event operation, see page 505.

Other Main Program Characteristics

In addition to the runtime and toolkit behavior described above, the Main Program has other intrinsic characteristics.

Main Program Task Prefix

The Task Prefix handler of the Main Program acts as the prefix phase for the entire application. Use this handler to define any operation you wish to perform for the initialization phase of the application.

You can initialize the Main Program's global variables using the operation, or call its initialization programs using the operation or any other relevant operation.

Main Program Task Suffix

Similar to the Main Program's Task Prefix, the Task Suffix handler can be considered as the suffix phase of the entire application.

You can use this handler to define any operation you wish to perform for the termination phase of the application.

RunMode function

Since the Main Program is executed before any other program, in both toolkit and runtime, you may use the RunMode function to distinguish the runtime state of the application.

Refer to Chapter 8, Expression Rules for more information.

Importing a Main Program

When importing from a file that contains a Main Program, you will be asked whether you want to replace the current Main Program. If you select Yes, the Main Program from the imported file will replace the current one. If you select No, the Main Program will not be imported from the imported file. A Main Program cannot be imported as a regular program.

Main Program and Components

The Main Program cannot be exported as part of a component. See Main Program and Components in Chapter 14, Components.

At the heart of each eDeveloper task is its Execution Operation repository. You fill in an Operation repository by selecting and tailoring from the list of eDeveloper's 14 high-level operations.

In this chapter:

- | |
|---|
| • Operations and Context-Dependent Behavior |
| • Operation Properties |
| • Passing Arguments |
| • Operation Usage |

Alphabetical Index to Operations

Operation Name	Operation Number	Page
Block	5	page 447
Browse	12	page 500
Call	7	page 459
End Block	6	page 458
End Link	4	page 454
Evaluate	8	page 483
Exit	13	page 503
Input Form	11	page 497
Link	3	page 437
Output Form	10	page 492
Raise Event	14	page 505
Remark	0	page 422
Select	1	page 422
Update	9	page 486
Verify	2	page 433

Introduction

At the heart of each eDeveloper task is its Execution Operation repository. There is a Task Execution Operation repository (abbreviated to Operation repository) for each level needed by the task: Task Prefix, Record Main, etc. Each Operation repository contains programming operations specific to that level. You fill in an Operation repository by selecting and tailoring from the list of eDeveloper's 14 high-level operations. You can further customize these tabularized operations by adding expressions, using selections from the eDeveloper library of over 387 functions. Together, these operations and functions embody the "language" of eDeveloper.

Each operation, its context-dependent behavior, and its properties, are described in detail below. The Argument List, used by the Call Task and Call Program operations, is also described below.

Whenever you create a new entry in one of the Operation repositories, eDeveloper inserts a Remark line as the default. You can then select one of the other 14 operations to replace this default.

Certain operations are allowed only in Operation repositories for specific levels of a Task. Watch the Operation repository when you choose Select, Link, and End Link operations: if you have selected the operation but the default operation doesn't change to display your selection, it means you have attempted to use the operation at the wrong level. The Select operation is used by eDeveloper for the definition of the dataview and therefore:

1. They can be defined in the Operation repository of the Record Main and Handler levels.
2. In Batch tasks, Select, Link, and End Link are the only operations executed at the Record Main level.

The Operation diagrams are presented according to the following conventions:

- Each operation is shown as the first operation of the Operations: Record Main repository.
- The column headings and the in-line property names shown identify places where the insertion point parks to accept the entry of a value.
- Property default values are shown.
- The specific commands available for each property are indicated (for example, zoom).

Remark

Purpose

- To insert a blank, or line of text, across the Operation repository to serve as separator between groups of operations, improving the readability of the repository, or
- To insert a comment text in the program.
- The Remark line is not an executable operation.

Usage

Program documentation.

Remark Operation Property

The default content of the entire row area is a blank in a graphical environment, or a continuous line in a text environment.

You can type alphanumeric text the full property length, by moving the insertion point right and starting to type.

Select

Purpose

To define all the task's dataview logical record elements and to declare variables to be used in the task. Declaring a variable with a Select operation is referred to as selecting the variable.

Usage

- To build the dataview logical record by doing one of the following: selecting Real variables from either the main file or linked files, selecting Virtual variables, or selecting 'Parameter' type variables to receive values from called programs and tasks. Virtual variables, once selected, are automatically inserted into the Virtual Variable list according to their position in the Operation repository, relative to any other Virtual variables. Regardless of where selected variables originate, once they have been selected in the Record Main Operation repository, they are available variables for task processing.
- To make the task dataview dependent on a variable's actual value. The task dataview is the set of all of the logical records to be used by the task. You can make the task dataview dependent on a variable's values by specifying suitable expressions in the Range property of its Select Variable operation.
- To assign default initial values to real variables whenever new records are created, define an Init expression in the Select Variable operation, as explained in the property descriptions below.
- To implement computed variables, select the Virtual Variables property.
- To receive values from a called task or program, select the Parameter variable-type option.
- To locate a record with a specific value in a variable, specify a suitable expression in the first Locate field of the variable's Select Variable operation.
- To control the sequence of insertion point movement and to control which variables the insertion point is allowed to park on, determine the sequence of Select operations. You can also set the Cnd property.

Placement

The Record Main Operation repository is where the task dataview is established. Therefore, Select Variable can be placed in the Record Main Operation repository only.

The sequence in which Select operations appear in the Operation repository determines the sequence of insertion point movement from displayed control to displayed control at runtime for Online tasks. The placement of controls when you design the Form window does not affect the sequence of insertion point movement at runtime.

The position of Select operations in the Operation repository does affect the recomputation process of Init expressions, as explained below.

Select Operation Properties

Select

Each Select operation must specify variable types that are Real, Virtual, or Parameter. Select Real is the default when you have specified a DB table as main file in the Task Properties dialog. Otherwise, Select Virtual or Select Parameter are the only variable options.

eDeveloper automatically creates a Virtual or Parameter variable in the task's local variable list for every Select Virtual operation. The variable number that eDeveloper inserts is the sequence number of the newly created Virtual or Parameter variable in the local variable list, according to its position in the Operation repository. All subsequent Virtual or Parameter variables are renumbered automatically.

Select Parameter and Select Virtual variables act the same in receiving arguments that are passed by another task, and are available for selection in any sequence. The Select Parameter option allows the developer to anticipate the expected arguments of a called program or task. If no Select Parameter variables are set in a task then the virtual variables will receive passed arguments. Refer to the Argument Matching section.

Name

The default Name is ???. Once you have defined the selected variable, this property automatically displays the name of the variable from the local variable list containing both Parameter and Virtual variables.

Init > 0 (optional)

The value of the Init property is the number of an expression. At runtime, eDeveloper evaluates the expression and assigns the resulting value as the initial value the variable will contain when it is created. The default value is 0, which means that there is no Init expression for this variable.

The Init property is one way to assign a value computed by an expression to a variable in eDeveloper, just as you would assign a formula to the cell of a spreadsheet.

The Init expression does not need an = in it. The assignment is made implicitly by eDeveloper. For example, if you need to initialize the variable Order Sum with Price * Quantity, specify the Init expression for the Virtual variable named Order Sum simply as Price * Quantity.

How Init Works

Init assigns values according to two scenarios: procedural and non-procedural. In the procedural method, the same rules apply for Virtual and Parameter variables, while different rules apply for Real and Virtual variables:

- Real Variables - are created only when the task runs in Create mode of operation. Therefore, eDeveloper evaluates the Init expression and assigns its value to a Real variable only when the task runs in Create mode of operation and before the record is displayed on the screen. Real variables within a link will be initialized by the Init expression if the link fails. Refer below to The Effect of Link Failure on Init Expressions.
- Virtual Variables - are always created on-the-fly. Therefore, eDeveloper evaluates the Init expression and assigns its value to a Virtual variable in all modes of operation and before the record is displayed on the screen.
- Parameter Variables - are created and behave like Virtual variables.

A Virtual variable without an Init expression in its Select operation receives its value through the Update operation. Such a variable is not zeroed out at the beginning of the record process and retains its previous value. eDeveloper adopts this approach to Virtual variables without an Init expression to support the use of Virtual variables as accumulators.

Automatic Non-Procedural Recomputing of Init Expressions

Regardless of the mode of operation of the task, eDeveloper automatically recomputes and reassigns Init expressions just as in a spreadsheet, when any of the variables participating in the expressions change during the execution of the Record Main or Record Suffix. eDeveloper does the recomputation and reassignment according to the following rules:

- Record Main

Init expressions are recomputed if any of their component variables have been modified and the modified variable has been selected prior to the declaration of the variable with the expression. The recomputing mechanism in Record Main works forward only.

- Record Suffix

Any changes to variables in the Record Suffix will cause recomputation of all the Record Main Init expressions that these variables participate in. The scope of recomputation is from the modified variable's selection position in the Operation repository to the end of the Operation repository.

The recomputation of Initialized variables is one of several non-procedural processes of eDeveloper's engine. For more information, refer to Chapter 5, Application Engine.

In the following explanation, a variable whose Select operation includes a non-zero Init expression number is called an initialized variable. Any variable that is a component of the Init expression, and whose modification causes the recomputation of the initialized variable, is called a modifying variable.

Suppose that Real Variable A has value 5 and Real Variable B has value 6. If Real Variable C's Select Variable operation specifies as its Init expression $A*B$, then while the end-user runs the task in Create mode, Variable C will be initialized to the value 30. However, if the end-user later changes the value of Variable A to 7, the Init expression for C will be automatically (non-procedurally) recomputed to 42 without the insertion point moving to Variable C.

A change to a modifying variable can be caused by:

- User input, as in the example above
- An Update operation
- Its usage as a property of a subtask or program
- Data imported by an Input Form operation
- Its usage as a Return Code variable

A change to a modifying variable may occur either in the Record Main (interactive) phase and any subtask called from the Main level, or in the non-interactive Record Suffix phase.

Recomputation is carried out:

- For any variable with an Init expression that contains variables
- In both Create and Modify modes of operation
- In both Online and Batch tasks
- In both Fast mode and Step mode.

Recomputation proceeds repeatedly for all recomputed Initialized variables, because a recomputed variable in its turn might be a modifying variable for other Init expressions. One cycle of recomputation may cause another, until all Init expressions are satisfied.

To be recomputed, the Initialized variable must meet both of the following conditions:

- The Initialized variable's Select operation must be positioned in the Operations repository after the Select operation of the modifying variable.
- The Initialized variable's Init expression contains the modifying variable.

The following two examples illustrate the influence of the above conditions on recomputation.

Recomputation Example 1

As one of its task initialization procedures, eDeveloper builds its recomputation tables. eDeveloper uses these tables to quickly perform any automatic non-procedural recomputation that may be required during task execution.

In this example, Condition 1 above is not met, and as a result Variable A is not recomputed.

Suppose that a task contains two Select operations in its Record Main's Operation repository and two Update operations in its Record Suffix Operation repository:

Record Main

Line No. 1 Select Variable A - Init expression: Variable B

Line No. 2 Select Variable B - Init expression: Variable A

Record Suffix

Line No. 1 Update A

Line No. 2 Update B

In the example above, eDeveloper does the following in order to build the necessary recomputed information for Init expressions:

For every variable, search through the Record Main Operation repository, from the line that selects it to the end of the repository. The search is for all Select Variable operations with an Init expression where the variable participates. Any such expression is registered in the recomputed repository.

In the example, eDeveloper will find that the Init expression in Line 2 contains variable A, and because it appears after the Select operation of Var A in line 1, will add a reference in the recomputed repository. The Init expression in line 1 will not be added to the recomputed repository, because it contains variable B, which is declared in line 2, contradicting condition 1 mentioned above.

At runtime, while there is interaction with the record, any value input into A will automatically cause a recomputation and be assigned to B. But changes to B will not be reflected in A, because there is no reference for B in the recomputed repository.

When interaction with the record terminates and execution passes to the Record Suffix, the Update for Variable A will cause another recomputation of B, because the update operation causes A to change and A has an associated entry in the recomputed repository. The update for Variable B will not cause any recomputation.

Recomputation Example 2

Suppose that the Record Main's Operation repository contains the following operations:

Line 1	Select Variable A
Line 2	Select Variable B Init: $A + C$
Line 3	Select Variable C
Line 4	Select Variable D Init: $A + C$
Line 5	Update C

Assume the task is in Modify mode and that the variables have the following values:

- $A=2$
- $B=5$
- $C=3$
- $D=5$

The user inputs the value 3 to variable A (the modifying variable). The variables then have the values:

$A=3$ (modified by user)

$B=6$ (recomputed by eDeveloper)

$C=3$ (unchanged)

$D=6$ (recomputed by eDeveloper)

Then the Update C operation at Line 5 is executed. Assume that Variable C is updated to the value 10. The variables then have the values:

$A=3$ (unchanged)

$B=6$ (unchanged - not recomputed)

$C=10$ (modified by the Update operation)

$D=13$ (recomputed by eDeveloper)

In this example, Variable B is not recomputed because Condition 1 was again not met: the Select Variable operation for Variable B is not after the Select Variable operation for the updated Variable C.

Range (optional)

Range expressions are one way to customize the task's dataview. They can define a subset of records from the physical files selected for the dataview.

You can specify two expression numbers in the Range column. The first expression represents the lower value and the second the upper value of the Selected variable used to delimit the range of the records you want to be included in the task dataview. Range expressions may be applied to any variable of the Main table, linked tables, and the virtual table.

The expressions must evaluate to a value matching the variable's attribute. If you need a logical condition to delimit the range, you can specify the condition in the Range Expression property in the Range/Locate dialog, or in the DB SQL Where clause when working with SQL databases.

Range expressions for the Select operation are evaluated at the beginning of the task. Be sure that all variables participating in expressions contain valid values when the task is activated. The variables can be variables selected in a parent task, arguments passed to the task, or any constant value.

At runtime in an online task, end-users are not allowed to browse records outside the delimited Range, but they can additionally restrict the range by selecting the default runtime menu option Range of Records from the Options menu CTRL+R.

The task dataview is the result of the logical AND of all Range lower/upper expressions from all Select operations, the expression from the Range Exp or Range values, and the DB SQL Where clause.

If no Range expressions are specified, eDeveloper assumes the full range.

Implementation of the One-to-Many Relationship

eDeveloper implements one-to-many relationships by calling subtasks, via the Call Task operation, together with putting Range lower/upper expressions on the subtask's selected variables. If the selected variables that carry such

Range expressions are Real variables which were defined as index segments at the Table Repository level, and if this index is the Main table's index for the subtask, then eDeveloper establishes the one-to-many relationship instantaneously.

Locate (optional)

You can specify two expression numbers in the Locate column, one each for lower and upper values. These values determine that the first dataview instance processed will be the first record that is found between the two expression values. The expressions must evaluate to a value matching the variable attribute.

If you need to locate the first record using a logical condition, specify this condition in the Locate Expression property in the Range/Locate dialog. For more information refer to the Range and Locate Properties section in Chapter 6, Programs.

Locate expressions do not affect the dataview itself.

The display of records after a Locate operation in an online task is determined by the setting of the Center Screen in Online property in the Environment dialog. If the Center Screen in Online is set to Yes, then the task begins with the located record presented in the middle of the Form screen, with the insertion point parked on its first parkable variable. If Center Screen in Online is set to No, then the task begins with the located record presented at the top of the Form screen.

Unlike Range, Locate allows the end-user at runtime to refer to dataview records outside the domain of Locate lower/upper expressions and to move the insertion point to those records.

If you specified both lower/upper Locate expressions for a variable which is an Index variable or a segment of an Index for this task, and no record meets the specified criteria, the processing pointer moves to the first record with a higher index than the upper expression value and the message Record not found - Positioned at next is displayed. If no record with an index higher than the upper expression value exists, the insertion pointer moves to the first record in the dataview and the message Record not found - Positioned at beginning is displayed.

If you specify only a lower Locate expression and a matching record could not be found, the processing pointer moves to the first record with a higher index, but no message is displayed. If you specified only an upper Locate expression and a matching record could not be found, the processing pointer moves to the first record of the dataview.

If the Locate Expression variable is not an index and a record matching the specified criteria is not found, then the processing pointer will be positioned at the first record in the dataview.

Flow

Flow qualifiers for the Select variable operation allow you further control over insertion point parking on the variable, beyond the logical condition you can put in the Cnd property, because they permit specification of movement direction and the interaction mode.

There are two columns for Flow properties: flow mode and interaction mode. The default value for flow mode is S (for Step) and for interaction mode is C (for Combined).

When Flow Mode is...	the engine executes the operation if...
S for Step	The task is now in Step mode
F for Fast	The task is now in Fast mode
C for Combined	The task is in either Step or Fast mode
B for Before	The end-user selects Zoom (F5) in the previous control.
A for After	The end-user selects Zoom (F5) in the next control.

When Interaction Mode is...	The engine executes the operation if...
F for Forward	The insertion point moves forward according to the Record Main Operation repository
B for Backward	The insertion point move backwards
C for Combined	The insertion point moves forwards or backwards

Cnd (for Condition)

This property contains a logical condition that controls whether the insertion point will park on a control displayed on the screen. The possible values are:

- Yes (the default) - means that the logical condition is True. At runtime the insertion point is allowed to park on this control.
- No - the logical condition is False. At runtime the insertion point skips the control.
- Expression number - Points to a logical expression whose value at runtime will determine whether the insertion point can park or not. [Zoom from here](#) to add the expression to the Expression Rules repository.

Note: The condition specified here has no influence on the operation other than the insertion point control.

Verify

Purpose

When the Mode property is set to Error and a related Condition expression evaluates to True, Verify allows you to display an error message and halt execution, so you can require the user to correct the error.

When the Mode property is set to Warn and a related Condition expression evaluates to True, Verify allows you to display a warning message without halting execution of the task.

Usage

- Input Validation
- Debugging - by inserting warning messages to trace the task execution

Verify Operation Properties

Exp > 0

After you select the Verify operation, the insertion point moves to the right part of the Operation column that displays a 0.

You can zoom from here to the Expression Rules repository and specify a string expression of one screen width, containing the message. The message characters can be a concatenation of constant text, numeric variables converted to strings, and string variables, so you can dynamically vary the contents of the message.

Name

If you selected an expression in the Exp property, this column shows the first part of the expression and the insertion point skips it when you move right.

If you left the Exp default property 0, you can type a constant string directly in the column itself. If the text is longer than one column width, an expansion box will open automatically.

If at runtime, the Cnd expression of the operation evaluates to True, the message is displayed in the message line or display box together with the sound of a beep. If the expression number is zero and there is no message in the Name column, nothing is displayed and a beep is not issued but the operation is still in effect.

Mod (for Mode) = Warning (default)

Verify operations with Mode set to Error executed in the Task Prefix level of execution cause the task to terminate (see below). Verify operations in other task levels are followed by resumption of execution according to the rules described below.

The available values for Mode are:

- **Warning** – This is the default setting. eDeveloper beeps and displays the message, but does not stop the task's execution. The user is free to ignore the message and continue. The message disappears as soon as the end-user presses a key in a message line or clicks OK in the related display box.
- **Error** – After the message is displayed, eDeveloper will behave according to the following rules:
 - If you use the Verify operation in the Record Main of an Online task, processing stops, and the insertion point parks on the last parkable control, and eDeveloper waits until the end-user corrects the error.
 - If you used the Verify operation in the Record Suffix of an Online task, processing stops, control returns to the last visited parkable control of the current record on the Form, and eDeveloper waits until the end-user corrects the error.
 - If you used the Verify operation in the Record Suffix of a Batch task, processing stops, eDeveloper skips to the next record and awaits user input. Pressing any key except ESC restarts the task with the next record. ESC terminates the task. The message should instruct the user to correct the error before pressing any key except ESC. You can use this feature as a break point for debugging.
 - If you used the Verify operation in other non-interactive levels -Task Prefix or Suffix for all tasks, Group levels for Batch Tasks, and Record Prefix - processing stops and execution of the level is aborted.
- **Revert** – This mode is available for Control and User-Defined handlers only and works the same as the Verify Error mode for the Record Main level.

When Verify Revert is executed, an error message box is displayed. The cursor returns to the previously parked control by executing the operations that precede the Verify operation in reverse order, back to the first operation in the handler.

For a Block operation, the Verify Revert mode performs the Block operations in reverse order. For a Block loop, The Verify Revert mode causes eDeveloper to exit the Block operation and perform the operations preceding the Block loop in reverse order.

Display Mode

The Display Mode property lets you specify the type of message format that is displayed for the Verify operation. The following display modes are:

- Box (default) -The message text is displayed in an dialog box. The eDeveloper engine will only continue to the next operation when the dialog box is closed.
- Status - The message text is displayed on the status line of the client window.

Flow

The Flow property is relevant only when you define the Verify operation in the Record Main Operation repository of Online tasks. The purpose of the Flow property is to allow you to make the execution of the operation dependent on how the end-user moves the insertion point on the task's main window.

Cnd (for Condition)

The Cnd property contains a logical condition that controls whether or not the operation is executed. For more information about the Cnd property.

Usage Considerations

Use the Verify operation to warn the end-user about a possible error, by setting the Mode to Warn.

Use the Verify operation in the Record Main of an Online task whenever you want to require the end-user to enter valid input, by setting the Mode to Error.

Link

General Information about eDeveloper's Link

eDeveloper's Link operation establishes one-to-one relationships between related database tables. The current record of the task main file is correlated (linked) to a specific single record of another database table (the linked file) whose index segments contain values which match the Link criteria (Link expressions). Links can be established to:

- perform validity checks, to verify that a particular record exists in the linked table,
- extend the record dataview by selecting variables from linked files, or view, modify or create records in the linked table.

The Link mechanism is part of eDeveloper's record dataview preparation: once the link is established, Select Variable operations can be used to add variables of the linked table to the record dataview. Linked variables are treated exactly the same as the variables selected from the Main table.

An important characteristic of the link is that it is dynamic. If the Link operation's expression values change during the processing of the Main table, the link is automatically re-established and new linked record variable values are set.

Link Categories

There are two reasons to use Link:

- to access an existing record from the linked file, or
- to create new records in the linked file.

Link types are explained with the Link operation later in this chapter. Link types for accessing existing records are: Validate, Query, Write (only if the linked record exists) and Join. Link types for creating new records are: Write, if the linked record does not exist, and Create.

Regardless of the purpose of the link, all variables selected from a linked file, including the variables that contain the access index segments, can be modified during task execution. The modified linked file rows are written back to the disk once all Record Suffix operations are executed, just as for modified main file rows and resident tables.

Link Order

The Link is part of the dataview definition, and therefore the link operations are included in the Record Main level definition of the task only. The minimal configuration of operations for a link consists of the following, shown in the order in which they are placed in the Operation repository:

- Link operation to specify which database table must be linked, which Link type, and by which index.
- Select Variable operations, with Locate lower/upper expressions to select the linked file index segments and to define the Link criteria using Locate expressions for each index segment.
- End Link operation to terminate the link's specifications.

The creation of the minimal Link configuration is a semi-automatic feature of eDeveloper. Once you have selected a Link operation, eDeveloper generates a matching End Link operation immediately after it. Further, once you have selected a file to link to and an index to be used as the Link Index to the file, eDeveloper creates a set of Select Variable operations between the Link and End Link operations, corresponding to every Index segment of the specified index. For example, if the access Index used contains two Index segments, two Select Variable operations are automatically created following the Link operation, each selecting an index segment of the specified link index. To complete the link specification, you just have to define Locate expressions for each index segment.

If you change the Index specified for the linked file, eDeveloper creates a new set of Select Variable operations for the new index segments.

To use the link to implement table lookup and to allow the task to refer to other variables belonging to the linked file, these variables must have Select operations between the Link and End Link operations. The variables are joined to the record dataview.

You cannot nest one link inside another link, even if the Index value used to establish the second link is selected by a Select Variable operation defined for the first link. In such a case, create a new link immediately after the first one, and use the linked variables for the locate expression of the index segments. The result is similar to nesting the two links.

Link Criteria

If the link's purpose is to access an existing record rather than to create a new one, the link is considered successful if eDeveloper finds a linked record whose index segment values match the values resulting from the evaluation of the Link expressions. The Return Code value for a successful link is True (Logical) or 1 (Numeric). If no matching record is found, the link fails and the Return Code is set to False (Logical) or 0 (Numeric).

Note: It is recommended to use Logical variables for link return codes, since the meaning is consistent with the Logical attribute. Numeric return codes are supported only for compatibility with previous versions of eDeveloper. eDeveloper will automatically recognize whether the return code variable is Numeric or Logical.

The following matching criteria can be used for links:

- Single value for matching - To specify a single index value for matching, make the Locate lower expression identical to the Locate upper expression for each index segment.
- Range of values for matching - When you define a range of possible matching values for one or more index segments by using two different Link expressions as Locate lower/upper expressions, eDeveloper will establish a link to the first record of the linked file for which the value of an index segment is included within the range of values specified. By using the Ascending/Descending property in the Link operation, you can control the order of access to the linked file, and thereby determine whether the match is with the first or the last record of the range.
- Partial Matching - To specify partial matching, define Locate lower/upper expressions for only those index segments you want used for matching. eDeveloper will establish a link to the first record of the linked file for

which the values of those index segments specified match the range of values specified.

- Extended Matching - In addition to the Locate lower/upper expressions you define in Select Variable operations for the linked file, you can also:
 - Specify additional Locate lower/upper expressions for any of the link's selected variables and on any Virtual variables selected within the link. eDeveloper will establish the link only if all Locate lower/upper expressions defined inside the link are satisfied. The only difference between link expressions for index segments and link expressions for other selected variables is that eDeveloper uses a different technique to perform the match: index segments are matched using the very fast index access, and all other variables are matched using a sequential search starting at the point where index segments were matched.
 - Specify Range lower/upper expressions for any of the link's selected variables and for any Virtual variable selected within the link. In such a case, all Range lower/upper expressions are evaluated when the task is invoked (regardless of whether the Select Variable operations they are associated with are defined over the main file or over any of the linked files). Later, the link is established, and the selected variables of the linked record are added to the record dataview. Then eDeveloper checks that the link's selected variables having Range expressions match these Range criteria.

If not, the current main file row is skipped and the next is read.

Establishing the Initial Link

Link operations are part of the instructions to eDeveloper for creating the dataview's Logical Record, just as Select operations are. Thus eDeveloper establishes initial Links during the record dataview preparation prior to the execution of the Record Main operations. During Record Main processing, eDeveloper can re-establish the links non-procedurally.

Recomputing Link Expressions

Link is one of eDeveloper's non-procedural facilities. During the execution of Record Main or Record Suffix operations, whenever the value of a variable that is a component of any of the Locate lower/upper expressions used with link's selected variables changes, and is declared in the Operation repository prior to the link, eDeveloper recomputes the Link expression and re-establishes the link. Thus all the link's selected variables can receive new values. Refer to the description of Recomputing Init Expressions in the Select operation.

A modifying variable is a component variable of the link expression, declared in the Operation repository prior to the Link definition, and whose modification causes the recomputation of the link. The modifying variable can be changed because of user input, an Update operation, data import by an Input Form operation, another link's return code re-evaluation, or as a result of having been passed as an argument and changed in the called task.

Link recomputation can occur in Create or Modify modes, in Online or Batch tasks, in Step or Fast modes. The change which triggers the recomputation can occur in either the Record Main or Record Suffix stages of processing.

Warning: Be careful when you use the Update operation in the Record Suffix Operation repository on a variable that is a component of a Locate lower/upper expression of a link. The execution of the Update operation may cause the Link expression to be re-evaluated and the link to be re-established, thus causing the loss of the updated values of other variables selected from the Linked record. Conversely, you can use such an Update operation in an Online task's Record Main to force a revaluation of the link.

The Effect of Link Failure on Init Expressions

When a link has failed, all Init expressions defined in the Select operations specified for that link are still evaluated and assigned to related variables following the same rules as select variables that are not within a link.

Note: The Create type of link always fails, by definition.

Link Usage

- To implement one-to-one relationships between the main file and other database tables.
- To search for individual rows in a file.
- To validate information against data from another file.

Placement

You can define Link operations in the Record Main Operation repository only, where the task dataview is established.

Link Operation Properties

The Operation column for Link operations is divided into three areas: the word Link, the link category (Write, Query, Create or Join), and the identification number of the linked file.

When you select the Link operation, eDeveloper displays Link Query as the default.

Move the insertion point to the word Query if you want to change to one of the other categories. Link Category - Click on the combo box to see the five link categories and select the one you want.

- Query links are used to search for an existing record.
- A Write link is used when the matching row may or may not exist, and you want to read it if it exists or create one if none exists.
- Create link creates a new row with the row index value specified.
- Link Join operations, Inner Join or Left Outer Join, uses the RDBMS Join capabilities for the reading and hooking modes of the task in conjunction with the eDeveloper mechanisms.

Link Types

Query

Validation = Yes

When you define a Link Query operation with the Validation property set to yes, eDeveloper attempts to establish the link. If the link is successful, data from the linked record's selected variables are included in the record dataview.

If the link has failed for online tasks, the execution of the task is interrupted and the following error message is displayed, accompanied by a warning beep:

Record not found in table: <filename>

where <filename> is the name of the linked file taken from the Table column of the Table repository.

eDeveloper then waits for the user to input another value for the link key or to select Edit/Cancel to undo the record updates to this point.

For linked variables whose Select operation has no Init expression, the variables are filled with zeroes (if they are Numeric, Date or Time variables), with blanks (if Alpha), left empty (if Blob or Memo), or with the logical value False (if they are logical variables). The Default values determined in the Application properties can be overridden at the Model and Column levels.

For linked variables whose Select operations have Init expressions, the variables are filled with the value of the expression, according to the Init evaluation rules for the Select operation.

If the link has failed for batch tasks, the execution of the task continues and ignores the failed link.

Validation = No

When the Link category is Query, eDeveloper attempts to establish the link.

If the link is successful, data from the linked record's selected variables are included in the record dataview.

If the link fails:

- The linked variables whose Select operations have no Init expression are filled with zeroes if the variables are Numeric, Date or Time, with blanks if they are Alpha, left empty for variables that are Blob or Memo variables, with the logical value False (if they are logical variables). The Default values determined in the Application properties can be overridden at the Type and Column levels.
- The linked variables with Init expressions are initialized with the value of the expression according to Select operation Init rules.

Hint: When using the Link Query, you can customize an error message to be displayed when the link fails. To do this, insert a Verify operation with a Condition expression based on the return code property of the link, as shown in the explanation of Ret (below).

Write

When the Link category is Write, eDeveloper attempts to establish the link. If the link is successful, data from the linked row's selected variables are included in the dataview.

If the link fails, eDeveloper creates a matching record by filling the linked variables according to the following rules:

- Linked variables whose Select operation has no Init expression, are filled with zeroes if they are Numeric, Date, or Time variables, with blanks if they are Alpha, or left empty with Blob, or Memo variables. The Default values determined in the Application properties can be overridden at the Type and Column levels.
- Linked variables whose Select operations have Init expressions, and which are in an Online task or in a Batch task in Modify mode, are initialized with the value of the expression.

Create

When you define a Link Create, eDeveloper doesn't attempt to establish the link and always creates a new record filling the linked variables as follows:

Linked variables without Init expressions are filled with zeroes if they are Numeric, Date, or Time variables, with blanks if they are Alpha variables, or

left empty if they are Blob or Memo variables. The default values determined in the Application properties can be overridden at the Type and Column levels.

Linked variables with Init expressions are initialized with the Init expression value.

InnerJoin

The Link Inner Join operation uses the RDBMS Join capabilities for the reading and joining of different database tables, in conjunction with eDeveloper mechanisms, to create a one-to-one relationship. When the user updates a record or when an eDeveloper recompute is taking places, the Link Inner Join operation will behave as a Link Query operation with the Validation property set to True.

The Link Inner Join operation is valid only for SQL databases. Only tables from the same database can be joined.

The Link Inner Join operation will generate an SQL inner join statement among all the participating tables.

In a regular Link operation (Query), for each row being read from the Main table, eDeveloper adds a second SELECT statement to retrieve the linked row. The rows for the Main table are read by a loop that fetches all the rows as described above. To perform the link, every time the FETCH operation of a row in the Main table is performed, another cursor is opened to retrieve the rows of the linked table. If there are two links on a Main table, eDeveloper opens a cursor for the Main table and in a loop opens a cursor per each linked row. For example, a task with 10 records in the Main table and 2 linked tables will open 21 cursors: one cursor to fetch all the 10 records in the Main table, 10 cursors to fetch one record at a time from the first linked table and another 10 cursors to fetch one record at a time from the second linked table.

A regular Link operation is different than a Link Inner Join operation:

- A Join of several tables makes them into one entity, and only one cursor will be opened and retrieved.
- If the row in the Main table does not exist in the joined table, the row will not be retrieved.

- A regular link in eDeveloper results in slower response time than a Link Inner Join because the regular link communicates and requests more from the SQL Database than a Link Inner Join.

It is advisable to:

- Use a view that joins tables in reports and read- only queries.
- Cache the linked tables, especially if the linked tables contain only a few rows, or if the linked tables result in repeatable identical rows.

Link Inner Join differs from Link Query (with Validation=True) in the following ways:

- In Link Query (with Validation=True), a sub-select is done to fetch each linked row. If a linked row is not found, the corresponding row from the Main table is still available.
- In Link Inner Join, an inner join SELECT statement is used for the main and joined table. For this reason, integrity constraints are important.

When a dataview is created for the task, eDeveloper generates a Join statement for all of the Joined tables. eDeveloper creates a WHERE clause that consists of the constants and the matching clauses with the names of the columns from the Main table and the Joined table. Then eDeveloper builds the rest of the dataview.

When Link Inner Join generates a Join Statement, it will always return a True variable. When the Link Inner Join behaves as a Link Query (with Validation=True), it will attempt to establish a link where data from the linked record's selected columns are included in the record dataview. Listed below are general rules about Link Inner Join Operation usage:

- All the index segments of the joined table must be selected and each of the index segments should have a Locate expression.
- For the Joined tables, an exact range is needed. Therefore, the Minimum Locate expression must be equal to the Maximum Locate expression.
- Locate expressions on Joined tables can only be:
 - A constant

- A variable reference to a column from the Main table that was selected in the task
- A variable reference to a column from a join table that was previously selected in the task

The cache that the Link Inner Join operation utilizes is cleared in the following cases:

- When changing the index
- When a column from a joined table is updated
- When a locate/range/sort is performed
- When changing the task mode to Create

Usage Considerations

- The Link Inner Join operation is valid only for SQL databases. Only tables from the same database can be joined
- Because DB2 and Informix require separate locks for each table, Link Inner Join in batch tasks with immediate locking strategies will be implemented as a Link Inner Join operation without a lock for each table joined, and as a Link Query operation for each row fetched
- When eDeveloper generates a SELECT statement with multiple tables, eDeveloper uses aliases for the table names, such as in A for a Main table, B for the first join and so on
- Do not try to create duplicate records when you define a link based on a unique index. eDeveloper will not create duplicate records and will not issue any message about the failed attempt.

Left Outer Join

eDeveloper can implement the Link operation as a Left Outer Join if both the Main and Joined tables are SQL tables from the same database. Also, the relationship between the Main and Joined tables must be a many-to-one relationship: at least one main record that links to either one or no supplemental records in the Joined table. A Left Outer Join operation joins a

record from the Main table in eDeveloper to either a matching record in the Joined table, or a set of Null values returned if no matching record is found.

When a matching row is found, a true value is returned. If no matching row is found, a false value is returned.

The Left Outer Join Link Operation syntax for each available DBMS is described in the table below.

DBMS	Syntax Structure	Example
MSSQL, DB2, and ODBC	SELECT {column list} FROM main_table Left OUTER JOIN linked_table ON (link_condition)	SELECT Dname, Ename FROM departments a Left OUTER JOIN employees b ON (a.deptno=b.deptno)
Oracle	SELECT {column list} FROM main_table, linked_table WHERE main_table.column=linked_ta ble.column	SELECT Dname, Ename FROM departments a, employees b WHERE a.deptno=b.deptno(+)
Informix	SELECT {column list} FROM main_table, OUTER linked_table WHERE main_table.column=linked_ta ble.column	SELECT Dname, Ename FROM departments a, OUTER employees b WHERE a.deptno=b.deptno

Left Outer Join and ISAM Databases

In the Record Main table of the Task Execution repository, you can select the Link Outer Join option for the link regardless of the main table source. For ISAM tables, eDeveloper opens the table and retrieves the linked data as a Link Query operation.

Left Outer Join Usage Considerations

Important considerations when using the Left Outer Join Link are:

- The developer cannot nest Left Outer Join Link operations. If the developer attempts to do a Left Outer Join Link to a control that is already part of a Left Outer Join Link block, the following error appears:

Error. Link operations cannot be nested.

- If the developer attempts to link a secondary table that is from a different database than the Main table, the following error appears:

Error. All Join tables must be from the same database.

- If the developer attempts to assign a non-unique index to the database files of the Left Outer Join Link operation, the following error appears:

Error. Databases have no unique indexes.

- If the developer does not select all of the index segments, the following error appears:

Error. All Join table's index segments must be selected and have locate expressions.

- If the developer does not indicate Min and Max Locate expressions for all index segments, the following error appears:

Error. All Join table's index segments must be selected and have locate expressions.

- If the developer defines the index segments with a complex expression (A+B), the following error appears:

Error. Locate on Link Inner or Outer Join fields must be either a variable or a constant.

- If the developer defines a virtual variable within the Left Outer Join block and sets the locate expression with the defined virtual variable, the following error appears:

Error. Locate on virtual variables are ignored inside a Link.

Link Properties

Table Identification Number Property

This part of the Link operation properties contain the Table Identification number of the table to be linked.

Zoom from the Table Identification property for a list of supplemental tables that are in the same database as the Main table.

Indexing

Only a unique index can be used between the main and the linked file, a many-to-one relationship. Zoom from the Index property for a selection list of unique indexes.

Locking

When eDeveloper needs to lock a record, eDeveloper generates a lock in the underlying RDBMS.

In SQL gateways that use logical locking, the logical locking is done simultaneously for the main and the joined tables when Access/Share modes on the Main table are Write/Write.

In SQL gateways that use physical locking, the SQL gateway checks whether the RDBMS allows a FOR UPDATE clause for a joined table Select:

- If the RDBMS, such as Oracle, allows a FOR UPDATE clause for a joined table Select, the Lock is done simultaneously for those main and joined tables that are in Write Access mode.
- If the RDBMS, such as DB2 or Informix, does not allow a FOR UPDATE clause for a joined table SELECT, the lock is done separately for each table.

If a join table was opened in the task with read access, eDeveloper will try to lock only the Main table and not the joined table.

When eDeveloper needs to update the database, it updates each joined table separately according to the position index for that table. For more information, refer to Chapter 25, SQL Considerations.

Direction: Ascending (default)

The direction in which eDeveloper scans the records, according to the selected access index. Specify Asc for Ascending (the default) or Des for Descending. By specifying Des, you reverse the sequence of all scan operations for all Locate lower/upper and Range lower/upper expressions associated with the Select Variable operations inside the link.

Flow

This property is irrelevant to the Link operation and the insertion point skips it.

Cnd (for condition)

By entering Yes, No, or an expression in the Cnd column of the task's Flow table, you define the Link condition. The expression should evaluate to a logical value.

If the Cnd column value is Yes or evaluates to a True value, the Link condition fetches the required record.

If the Cnd column value is No or evaluates to a False value, the record will not be fetched. The record link will behave as a failed link, and the following will occur:

- All values of the Link operation's selected columns return to their default values.
- The return value of the link is False.
- Update operations do not take effect until the condition evaluates to True.
- If the link condition is evaluated to False at the end of the Record Suffix, any modification is disregarded.

Link Property Dialog

The validation condition properties are located within the Link Property dialog. Link Properties define the way the link condition will be evaluated for any Link operation. The dialog contains the following properties:

Return: for Returned Value

The Return property is optional. It allows you to specify a variable that will receive the return code indicating whether the link succeeded or failed. At runtime, after link execution, the variable specified in the Ret property will receive the value True (Logical) or 0 (Numeric) if the link was successful, and the value False (Logical) or 1 (Numeric) if the link failed.

Usually a Virtual variable is used for a Ret property, defined outside of the link. The Virtual should be defined as Logical or Numeric in the Local Variable repository.

Hint: You can use the return code to customize an error message for link failure to the end-user, by specifying a link type other than Validate and inserting a Verify operation, following the End Link operation, with a Condition expression as follows:

X <> 0	for a Numeric return variable
not X	for a Logical return variable

Where X is the letter code of the variable selected for the Ret property.

Validation

The Validation property, located in the Link Properties dialog box, is optional and is enabled for Link Query and Link Inner Join operations.

The Validation property combined with the Link Query replaces the use of the Link Validate operation from previous versions, which has been removed.

Imports from previous versions of eDeveloper will be automatically created as Link Query operations with the condition of the Link Validate operation placed in the Validation property.

In the Link Query operation, this property behaves as follows:

Yes - means that the logical condition is always True. eDeveloper attempts to establish the link. If the link is successful, data from the linked record's selected variables are included in the record dataview. If the link has failed, eDeveloper issues an error message.

No (the default) - means that the logical condition is always False, which causes the validation to always behave as a Link Query operation.

Expression number - Zoom from the Validation property to specify an expression that will determine the behavior of the link validation at runtime. If the expression evaluates to True, the link validation works normally. If the expression evaluates to False, the link validation works as a Link Query operation.

In the Link Inner Join operation, this property behaves as follows:

When you use Link Inner Join, you are fetching the data from the database using a SELECT JOIN statement.

When you update the linked column in the main table, you are not sending a JOIN statement; you are only sending a SELECT statement to the linked table to bring the proper data.

No - means that if in the linked table there is no corresponding record there will be no error message and when the record is refreshed, it will not be displayed.

Yes - means that eDeveloper will attempt to establish a link. If in the linked table there is no corresponding record there will be an error message.

Evaluate Link Condition

When the property is set to Task, the condition will be evaluated only once before the entire view is retrieved. If the condition returns a False, the link will not be part of the SELECT statement. Should the condition evaluate to True, the link will not be recomputed as a link query.

When the property is set to Record, and using either a Link Inner Join or Link Outer Join, a record is fetched. Should the link condition evaluate to True, the link will be recomputed as a regular query link. If the link condition evaluates to False, the linked data will not be fetched and it will be replaced by the default values.

End Link

Purpose

The End Link operation closes the link opened by its counterpart Link operation. End Link indicates to eDeveloper that the list of variables from the linked file has ended. All the variables selected by Select Variable operations placed in the Record Main Operation repository between a Link and an End

Link operation will come from the linked file. Any variable whose Select Variable operation is outside a link will come from the Main table.

Usage

Refer to the explanations of the Link operation above.

Placement

The End Link operation is allowed only in the Record Main Operation repository.

End Link Operation Properties

The End Link operation has no properties.

Usage Considerations

eDeveloper automatically defines the End Link operation when you define a Link after the Link operation in the Operation repository.

Do not remove or override the eDeveloper-generated End Link operation if you have not deleted its related Link operation.

If you erroneously deleted an End Link operation, restore it by defining it manually.

The Syntax Checker tells you if an invalid Link configuration exists in the Operation repository, such as nested links or unpaired Link/End Link operations.

Block

Purpose

Block and End Block operations enclose a group of procedural operations within a logical block. The execution of all operations in the block is dependent on the Block operation type and the condition of the Block operation.

Usage

The developer can determine the condition of the execution of an operation set as a single unit in one place.

The developer can create a set of operations where one set is executed when a condition is met and another set is executed when the condition is not met.

The developer can repeat the execution of a set of operations.

Block Operation Types

There are three Block operation types:

- If
- Else
- Loop

You can determine the Block operation in the Type box at the right of the operation name.

If

The If type is the basic Block operation.

The If type must correspond with an End Block operation. When the eDeveloper engine reaches the If type, it evaluates the operation condition. If the condition is met, the operations within the block are executed. If the

condition is not met, the eDeveloper engine skips to the operation after the corresponding End Block operation. The Block If operation and the corresponding End Block Operation can be nested in another Block operation.

Else

The Block Else operation is an extension of the Block If operation.

The Block Else Statement does not have a corresponding End Block operation of its own because it is considered to be within the same logical block as the preceding Block If operation. You may define several Block Else operations as an extension of the same Block If operation.

When the Block If operation condition is met, the operations defined from the Block If operation to the first Else operation are executed, and the eDeveloper engine skips the remaining Block Else operations until the next End Block operation.

When the condition of the Block If operation is not met, the eDeveloper engine skips to the first Block Else operation and evaluates the condition. If the condition of the Block Else operation is met, the eDeveloper engine executes the Block Else operations until the next Block Else operation and skips to the End Block operation.

When the Block Else operation condition is not met, the eDeveloper engine skips to the next Block Else operation until all the Block Else operations have been evaluated.

Block Loop

The Block Loop operation instructs the eDeveloper engine to repeatedly execute the operation within the block.

When the eDeveloper engine reaches a Block Loop operation, it evaluates the Block Loop condition. If the condition is met, eDeveloper executes the set of operations within the block. When the eDeveloper engine reaches the corresponding End Block operation, it returns to the eDeveloper Loop operation and evaluates its condition again. As long as the condition is met, the eDeveloper engine will execute the Block Loop's set of operations. If the condition is not met, the eDeveloper engine skips to the End Block operation and continues the task flow.

You can monitor the number of times the Block Loop operation was executed by using the LoopCounter function. For more information, see the LoopCounter function in Chapter 8, Expression Rules.

Flow

The Flow property is available only when the Block operation is defined in the Record Main Operation repository of an Online task. The purpose of the Flow property is to allow you to make the execution of the Block operation dependent on the end-user's interaction with the calling task's main window.

Cnd (for Condition)

The Cnd property contains a logical condition that determines whether or not the operation execution occurs.

Usage Considerations

Block If or Loop and End Block operations can be nested to any level. eDeveloper follows the nesting hierarchy by automatically matching each Block operation with the corresponding End Block operation.

eDeveloper allows for only one Block Else option without a condition within a Block-End Block, and this unconditional option has to be the last sequentially.

A Block- Else option will not have its own End Block.

A Block Loop operation set to a permanently True condition will run indefinitely.

End Block

Purpose

The End Block operation closes the block opened by its counterpart Block operation. All operations between the Block and End Block operation belong to the same block.

Usage

Refer to the explanations of the Block operation above.

End Block Operation Properties

The End Block operation has no properties.

Usage Considerations

- eDeveloper defines this operation automatically when you define a Block, following the Block operation in the Operation repository.
- Do not remove or override the eDeveloper-generated End Block operation if you have not deleted its related Block operation.
- If you erroneously delete an End Block operation, restore it by defining manually.
- The Syntax Checker tells you if an invalid Block configuration exists in the Operation repository, such as unpaired Block/End Block operations.

Call Operations

Purpose

There are eight kinds of Call operations used to execute different types of subroutines:

- Call Task to call subtasks
- Call Program to call eDeveloper programs
- Call Exp to call Expressions
- Call Public to call a public program
- Call UDP to call user-defined procedures

- Call COM to call a COM object
- Call Remote to call programs from the eDeveloper broker to a remote terminal
- Call Web Service (Web S) to call a Web service by using a SOAP protocol

You can optionally pass arguments to the called subroutine and receive values back from it.

Passing Arguments

eDeveloper uses the following two devices for passing arguments:

- The Argument dialog associated with the Call operation in the calling task. This dialog contains all the variables or expression values you want to pass as arguments.
- The Local Variable repository of the called subroutine. Beginning at the top of the Local Variable repository, you define Virtual variables corresponding precisely in sequence and data type to the arguments in the Argument dialog.

The Argument List

The Argument list, accessed by zooming from the Arg column, displays the passed arguments on the left-hand side, and the Variable list on the right-hand side. You access the list of variables by zooming from the Var column in the Argument list.

The following entry columns appear in the Argument list.

#

The sequential repository entry number automatically assigned by eDeveloper. The insertion point skips this column.

Var (for Variable)

The Var column, together with the Name column, is used to identify a variable or expression to be passed to the subroutine, with the option to receive back data from the subroutine. This method of passing an argument is known as *by-reference*. The other method, *by-value*, is explained below, under Exp.

Zoom to the Variable list to select a variable, or type the letter code of the variable and go to the next variable. The letter code is displayed in this column and the variable name appears in the 'Name' column.

If you want to pass a constant value as an argument, skip the Var column and move to Exp.

The Var column is skipped if an expression is already defined.

Exp

The Exp column is used to pass to the subroutine a constant value by means of an expression. This is the by-value method of passing an argument to a subroutine. Note that you cannot receive data back from the subroutine in this property. The Expression number appears in this column and the Name column shows the first section of the expanded expression text. You can zoom to the Expression Rules repository to select an existing expression or to define the expression you want to pass. The expression may evaluate to any valid eDeveloper data type.

The Exp column is skipped if a variable is already defined as an argument. Remove the variable identifier if you want to use an expression as an argument instead.

Skip

Allows the user to skip an entry by checking the Skip column. If an entry is checked, no value will be passed to the receiving variable.

Description

Description of the passed variable or expression.

Prm Desc

Name of the parameter or virtual variable that will receive the passed argument.

Attribute

Attribute of the parameter to be matched with a passed argument. This attribute could be logical, alpha, or numeric.

Picture

The picture setting will indicate how the matched parameter or variable will appear. For example, a passed argument may be received by a 5-character string, as specified in the Prm Picture setting.

How eDeveloper Passes Arguments

In runtime eDeveloper passes arguments in the following ways:

1. When, at runtime, eDeveloper performs a Call Task, Program, or Exp operation in the program flow, it copies the current values of the variables listed in the Arguments list of the current task into the corresponding variables, defined as either Parameter or Virtual type variables.
2. When a Select Parameter is defined in the called program or task, arguments will be sent to Parameter type variables in sequential order. In this case, any Virtual variables will be ignored. If no Select Parameter is set in the called task or program, arguments will be passed to Virtual variables.

The receiving variables of the called task contain a copy of the information passed from the calling task. These variables can be used for any purpose in the called task: form display or user input.

3. Upon termination of the called task, the Virtual or Parameter variables used to receive passed arguments are copied back to all of the calling task's list of variables. There is no backtracking to variables that use

an expression for passing data in the called task. Hence, all passed arguments that are variables reflect any changes to their counterpart called task Virtual or Parameter variables. This has the effect of what is usually called *argument passing by reference*. All of the passed Expression operation values are one-way only, which means that the values are used in the called task, but any changes made in the called task are not returned to the calling task. This has the effect of argument passing *by value*.

Because argument variables receive back any changes to corresponding variables in the called task, a recomputation of Init and Link expressions may occur in the calling task, according to the rules defined in the Select and Link operation descriptions. Changing a value in a called task means changing it by end-user input or by an Update operation. If the changed argument variable is displayed on the screen, its display will be refreshed as soon as the called task returns to the calling task.

Call Operation Qualifiers

Call, Call Category, Identification Number of Called Entity

The Operation column for Call Operations is divided into three areas: the Operation list, the Call category - Task, Program, Exp, Public, UDP, COM, Remote, and Web S - and the identification number of the called entity.

1. When you select the Call operation, eDeveloper displays Call Task as default. Move the insertion point to the word Task if you want to change to one of the other categories.
2. Call category - Click on the combo box to see the eight Call types and select the one you want.
3. Number identifier - depends on the specific Call category (see below).

Name

The content of this column depends on the specific Call type and is explained later.

Arg: 0 (default)

The Arg field is optional and shows the number (quantity) of arguments passed to the called task and defined in the Argument list.

If you want to pass arguments, zoom from here to the Argument list and define the variables you want.

Frm (for Form)

The Form property is optional and shows the identification number of a Form in the calling task. If you leave the default setting, which is 0, then the Main window of the called task will appear in its own window. If you specify a number in this property, then the corresponding form will be used as the window to contain the called task's main window. That is, when the called subtask or program starts running, its main window will appear inside of the window specified by Form.

The form specified in this property must be defined as Class 0, and can be whatever size and position you like. Do not define any text or variables in this window. For more information about Open Window as Child, refer to the Forms properties in Chapter 10, Form Concepts.

Flow

The Flow property is available only when you define the Call operation in the Record Main Operation repository of Online tasks. The purpose of the Flow property is to allow you to make the execution of the Call operation dependent on the end-user's interaction with the task's main window.

Cnd (for Condition)

The Cnd property contains a logical condition that determines whether the operation execution occurs or is bypassed.

Call Task

Purpose

To invoke the execution of a subtask from a parent task. Each subtask can manage a main file different from the parent's main file and bring up its own window.

Usage

- To invoke subroutines.
- To implement One-to-Many relationships.
- To implement pick list windows.
- To implement any type of pop-up windows.

Call Task Operation Properties

You enter the Call Task Properties dialog by pressing CTRL+P or right-clicking the mouse in the Task Prefix Operation repository.

Lock

When a task or a program is called, eDeveloper suspends processing the current view record until the call returns. The called task or program has access to all the current view record's variables; in the case of programs, this access is to variables passed as arguments, while tasks have direct access. In either case, the called task or program may also update some of the current view record's variables.

When using the On Modify Locking Strategy (the Locking Strategy is specified in the Task Properties dialog, see Chapter 6, Programs), the view record is locked immediately on the first change, or after a complete variable was accepted.

If a called subtask or a program is about to change the current record, the developer should ensure that the current record is locked. Because subtasks and programs are complete execution units that can take some time to execute, it should be made impossible for other users to gain access to the

current view record and modify it while the subtask or program is executing. Otherwise, a situation could arise where the current record could not be saved when the called subtask or program terminates. Therefore, the current record should be locked before executing the subtask or program. However, if the called subtask or program is not about to modify the current view record, the current view record should not be locked before executing the subtask or program, so as not to block other users' access to resources.

The purpose of the Lck property is to control locking when calling task or programs. Allowed values are:

- No - means that eDeveloper will not lock the current view record when executing the task or program. It is the programmer's responsibility to verify that no updates will occur from the called process to the calling record variables. If such updates do occur they may be lost if the current view record cannot be saved due to changes made by other users.
- Yes - means the current view record will be locked immediately on executing the called task or program. Specify Yes for this property always when the called task or program are going to update the current view record, or the updates performed by the called task or program must be done together with the current view record.
- Expression - an expression can be used to specify the value of the Lck property. Zoom to the Expression Rules repository.

There are a few exceptions that will not lock the current dataview record even though the developer specifies Yes as the Lock property:

The locking strategy of the task is defined as Before Update or No Lock. In this case, the task locking behavior overrides the Lock property of the Call operation.

No arguments are passed to the called program, or all the arguments are passed as expressions (by value).

Sync Data

This property is enabled for Call Task, Call Program, Call Exp, and Call Public operations at the Record, Control, and Handler levels. This property is enabled

for Deferred, Nested Deferred, and Within Active Transaction modes. The property:

- Lets eDeveloper control the execution order of Data Manipulation statements.
- Defines a logical expression that can be evaluated as Yes or No.

Destination Frame

This property is relevant for browser-based programs, and is used to open a task in a named frame within a particular frame set.

Exp

This property is used to dynamically direct browser-based programs or tasks to a particular frame within a frame set.

Usage Considerations

- The Call Task operation is often used to implement zoomable windows. For a detailed explanation about using the Zoom modes of the Flow property refer to the Task Dataview section in Chapter 5, Application Engine.
- A called subtask automatically has access to all variables of its ancestors and it may update them. Therefore you do not usually need to pass arguments to a subtask. If you want to call a subtask multiple times during task execution, each time with different values, then you would need to pass a argument to contain the different values.
- A called subtask automatically has access to the operating system text files opened in its ancestors. For example, a subtask which prints does not need to define its own operating system text file if the file is already defined in its parent task.
- A called Online subtask must have at least one variable selected in its Record Main Operation repository and assigned to its Main window where the insertion point can park. Otherwise the subtask will not run.

Call Phantom Tasks

A phantom task is a closed task with a window not in focus. You can open a phantom task by triggering the corresponding Call operation defined in a handler. Phantom tasks can also be called from a program, public program, or expression.

When clicking a phantom task window, the engine scans for the corresponding Call operation in parent and ancestor task handlers, and goes to the handler with the corresponding Call operation.

The engine scans the following handlers:

- Control Prefix
- Control Verification
- Control Suffix
- System
- User Event
- Internal Event

System, User, and Internal events can be defined with either control-specific or not control-specific handlers. The engine scans for handlers located in the phantom's parent or ancestor tasks from bottom to top.

To find the corresponding handler, the following three conditions must evaluate to True:

- The handler property is enabled
- The Call operation corresponds to the Phantom task
- The Block condition is not a Block Loop

Note: Block Loops are always evaluated as False.

If no valid Call operation is located, the engine stays on the last selected control and does not open the phantom task.

If the corresponding Call operation is found in a handler that is not control-specific, the engine only triggers the event to activate the Call operation for

the phantom task. After the phantom task is executed, the engine resumes to the regular behavior of the current event as defined in the Event repository.

If the conditions evaluate to True for the Call operation in a control handler, the engine moves to that control and executes the corresponding Call operation for the phantom task.

If the corresponding operation is located in the Control Prefix handler, the engine executes the Call operation and parks on that control. If the corresponding Call operation is found in the Control Verification or Control Suffix handlers, the engine executes the Call operation and parks on the next control.

Call Program and Call Exp

Purpose

To invoke the execution of a program defined in the Program repository. Call Program and Call Exp operations are similar to the Call Task operation. The differences between called programs and subtasks are:

- Programs share data via passed arguments, while subtasks inherit access to all their ancestor variables automatically.
- Programs are reusable, while subtasks are permanently attached to their parent task and, therefore, cannot be called by other tasks.
- Call Exp allows dynamic calls to selected programs at runtime.

Usage

To invoke subroutines which need to be used throughout the system.

Call Program Properties

Only the properties which are specific to the Call Program appear below. All the other properties common to all the Call types are explained above in the Call Operation Qualifiers section.

The only difference between the Call Program and the Call Exp operation is the way you tell eDeveloper which program you want to call.

Call Category

You must click the Call Category combo box to select the Program option from the list of Call categories.

Identification Number of the Called Program

The Identification Number of the Called Program is required. It is the number of the program as it appears in the Program repository.

Type the number of the program you want or zoom to the Program list to select the program.

Name

After you have selected the program, its name is displayed in this column.

Call Exp properties

Only the properties which are specific to the Call Exp operations appear below. All the other properties common to all the Call types are explained above in the Call Operation Qualifier section.

The only difference between the Call Program and the Call Exp operation is the way you tell eDeveloper which program you want to call.

Call Category

You must click the Call Category combo box to select the Exp option from the list of Call categories.

Identification Number of the Called Expression

The Identification Number of the Called Expression is required. It is the number of the expression as it appears in the Expression Rules repository, which evaluates, at runtime, to the Identification Number of the program to be executed.

Warning: To allow eDeveloper to automatically update the program number inside the expression when the sequential number of the referred program changes in the Program repository, use the literal Prog in the expression. For example, suppose you want to execute either program number 3 or program number 4, depending on the value of the logical variable BA. The Exp you define should be IF(BA,'3'Prog,'4'Prog). For more information, see the discussion of Literals in the Chapter 8, Expression Rules.

Name

After you have selected the expression, the first part of its definition appears in the Name column, for display only.

Usage Considerations

- The Call Program operation is often used to implement zoomable windows. For details about using the Zoom modes of the Flow property, refer to Chapter 5, Application Engine.
- A program can be invoked in three ways: as a program called by the Call operation, using the CallProg function, and as a standalone from a menu entry. When it runs as a standalone program, any argument passed is assigned null values: numeric, alpha, or blob.
- A program can call itself and even pass arguments to itself. That is, eDeveloper programs are fully recursive up to the maximum execution nesting depth.

Call a Public Program

The Call Public Program option lets an eDeveloper application call other components that are only recognized by the application at runtime.

Examples are:

- Components for an eDeveloper application that were not yet created or were created separately from the eDeveloper application
- A local application that can call dynamic programs

You can zoom from the last cell in the Operation column to open the Call Public Program dialog box as shown in Figure 7-1.

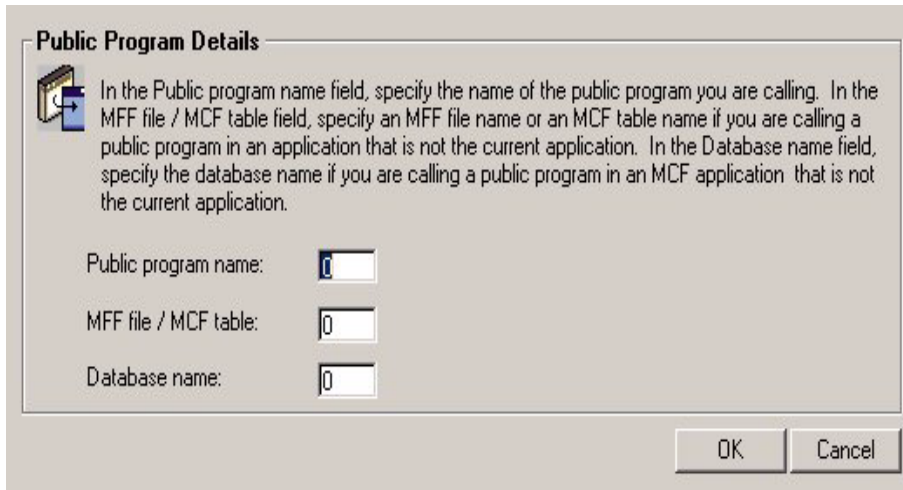


Figure 7-1 Calling a Public Program

You can call a public program from:

- A database application by selecting expressions for the application database, table, and public program.
- A flat-file application by selecting expressions for the application file and public program. A flat-file application does not need a database.
- The current application by selecting an expression for the public name. eDeveloper searches for the local program in the host application, the loaded components, predefined components, and other previously loaded applications. The first program found to match the public name is executed. If a corresponding program is not located, the operation fails.

The selected expression must return a string value. When an expression number is entered into the field, the expression is displayed. The Public name for an event is case sensitive.

After the Call Public Program fields have been specified, the expressions appear in the Call Public Operation Name column.

Runtime Behavior

When the database name is not a blank or Null value:

- The database name is matched with one of the databases in the Database list of the current eDeveloper session.
- The string of the second expression is regarded as an application table name.
- The string of the first expression is regarded as the public name of a program in the application.

If the database name is a blank or Null value:

- The string of the first expression is regarded as a public name of a program in the flat-file application. The public name is defined in the first expression.
- The string of the second expression is regarded as a flat-file name.

If the Database Name, Magic Control File application (MCF) or Magic Flat-File application (MFF), and Public Name are valid values, the following behavior occurs:

- The application is loaded.
- The Task Prefix operations of the main program are executed.
- The program that corresponds to the public name is executed, and the arguments of the Call operations are passed to the program.
- The Task Suffix operations of the main program are executed.
- When the called program ends, the Call operation is completed. Arguments passed as variables are updated. The variable set as the return value is also updated.

Loading the Application

The application remains resident as an eDeveloper component application until the runtime execution of the host application is terminated. Global handlers of the loaded external application may be invoked when the underlying event is

triggered. Global handlers of the loaded external application may be invoked when the underlying event is triggered.

All public programs of the loaded application are available by using external requests such as remote calls, Web services, Enterprise Java Beans (EJB), and Internet requests.

Closing the Host Application

When the host application is closed, the Task Suffix operations of the loaded application are executed.

The sequence of the Task Suffix execution is as follows:

- Host Application Task Suffix
- Components Task Suffix
- External Application Task Suffix

Handling Exceptions

The handler fails because:

- The specified table is not a valid application table.
- The specified file is not a valid Magic application flat file (MFF).
- The application is from a different eDeveloper version.
- The application has been located, but the public program has not been located.



The Flow Monitor reports the handler exceptions.

Call UDP

Purpose

To invoke the execution of a 3rd generation language program and optionally to pass arguments. eDeveloper uses the C language calling convention when calling this program. The call is done in memory as if the user procedure is an internal subroutine of eDeveloper, using the simplest call or jump instruction of the machine.

Usage

Implementation of eDeveloper extensions: user-supplied routines that perform specific tasks or functions not supported internally in eDeveloper.

Utilization within eDeveloper of existing code written outside of eDeveloper.

Implementation of specialized calculations that have been optimized for speed in the external procedure.

Call UDP Operation Properties

Only the properties which are specific to the Call UDP operation appear below.

Call Category

You must click the Call Category combo box to select the UDP option from the list of Call categories.

Identification Number of the UDP Expression

The Identification Number of the UDP Expression is required. It is the number of the expression in the Expression Rules repository that at runtime returns a string with the name of the called program.

Name

After you have selected the expression, the first part of its text appears in the Name , for display only. For example, if you are calling the Set Focus function,

@user32.setForegroundWindow is displayed.

Cnv

You can select from one of the C conventions below:

- C - Enables a dynamic and direct call to a DLL from within eDeveloper.
- Standard - Enables a dynamic and direct call to a DLL from within eDeveloper to an external Stdcall function.
- Fast - Enables a dynamic and direct call to a DLL from within eDeveloper to an external Fastcall function.

Call UDP Argument Example

If you call the SetForegroundWindow function,
@user32.setForegroundWindow is displayed.

The Argument values are entered as described below:

44 - A string where each character represents the argument type. The last character

represents the return value type of the function.

Argument types are:

- 1** - Char
- 2** - Short
- 4** - Long
- F** - Float
- 8** - Double
- D** - Double pointer
- E** - Float pointer
- L** - Long pointer
- A** - Null terminated string pointer

V - Void pointer

0 - Void Arg1, Arg2 - The function arguments from the DLL.

@user32.SetForegroundWindow - External function name called by the Call UDP operation

WINHWIN - eDeveloper's Window's window handler function

RETUDL - Return value handler

Usage Considerations

The eDeveloper distribution media contains sample C language programs that may be called from eDeveloper using the Call User Proc operation. All the necessary libraries for linking your function to eDeveloper's User Procedure facility are also provided on the distribution media.

Call COM

The Call COM operation lets an eDeveloper program call an ActiveX and OLE COM object, which means that all available COM objects can now be accessed from an eDeveloper application.

For information about Calling a COM object, see Chapter 15, COM Object Support.

Call Remote

The Call operation defines the service, program, arguments, and properties that will be executed when calling a program from a remote terminal. You cannot use the Soap protocol to define service parameters from a remote terminal.

Call Remote Operation Properties

Only the properties which are specific to the Call Remote operation appear below.

Wait

When a Call Remote command is encountered during runtime, the runtime engine checks for a valid server, and then passes the requested command to the eDeveloper RT requester. If the operation's Wait property is set to No, or evaluates to No, eDeveloper will continue. If the Wait property is set to Yes, or evaluates to Yes, eDeveloper waits for the completion of the request or until a time-out failure occurs. After the request has been completed, the variables that were sent are recomputed, and the flow of the program continues.

Call Web Service

Purpose

Call Web Service lets you call a Web service by using the SOAP protocol.

Call Category

You must click the Call Category combo box to select the Web S option from the list of Call categories.

Web Service Parameters

When you select Call Web S (Service) from the Operation column in the Record Main section, you can only select a service that has been defined in the Service

repository as a SOAP type. Double-click on the cell to the right of Web S. to open the Web Service dialog box as shown in Figure 7-2.

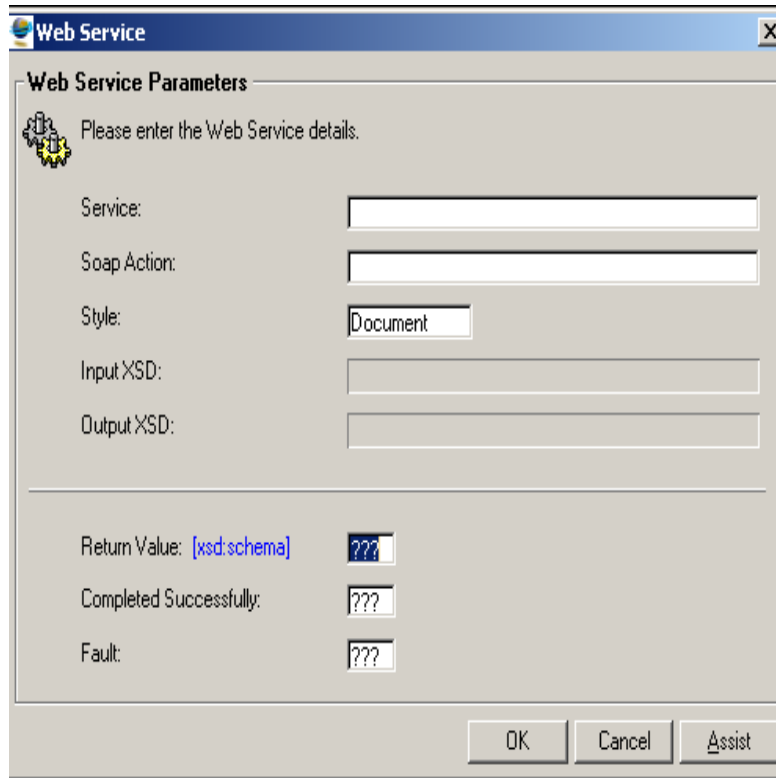
The image shows a Windows-style dialog box titled "Web Service". Inside, there is a section titled "Web Service Parameters" with a gear icon and the text "Please enter the Web Service details." Below this, there are five labeled text input fields: "Service:", "Soap Action:", "Style:" (which has "Document" selected in a dropdown menu), "Input XSD:", and "Output XSD:". At the bottom of the main area, there are three more fields: "Return Value: [xsd:schema]" followed by a dropdown showing "???", "Completed Successfully:" followed by a dropdown showing "???", and "Fault:" followed by a dropdown showing "???". At the very bottom of the dialog box are three buttons: "OK", "Cancel", and "Assist".

Figure 7-2 : The Web Service Dialog Box

The Web Service parameters are:

- Service - Select the Web Service defined as a SOAP type; for example, **Airport Temperature**. You must define the SOAP server in the Server repository before you can create a Web Service as a SOAP type.
- SOAP Action - eDeveloper displays the request identifier specified in the Web Service Provider's Web Service Description Language (WSDL).
- Style - You can select either the Remote Process Call or Document, as described below:

- Remote Process Call - eDeveloper sends arguments to the remote Web Service according to the argument types defined in the calling task. A developer must send matching arguments according to the types defined by the Web Service provider.
- Document - eDeveloper sends one argument as an outgoing XML document and receives one incoming XML document as the returned value of the call. A developer must generate these documents according to accepted XML schemas, as described in the Web Service Provider's WSDL.
- Input XSD - Displays the file location of the Input XML Schema, a local copy of the schema from the WSDL. This is relevant only for Document style.
- Output XSD - Displays the file location of the Output XML Schema, a local copy of the schema from the WSDL. This is relevant only for Document style.
- Operation - Enter the method name as defined in the WSDL of the Web Service Provider; for example, **getTemperature**. This field is for Remote Process Calls.
- Name Space - A unique identifier for the service as defined in the WSDL; for example, **capeconnect:AirportWeather:com. capeclear.weath**. This field is for Remote Process Calls.
- Return Value - Enter a variable to receive the response returned from the Web Service. If you have selected a Web Service operation from the WSDL Call Web Service dialog box, described below, the data type appears next to the Return Value field.
- Completed - Enter a boolean variable that indicates whether the Web Service is completed.
- Fault - Enter a fault string that is returned when the Web Service fails.

You can click Assist to open the WSDL Assist dialog box, which is explained below.

WSDL Assist

The WSDL Call Web Service dialog box lets you update the Web Service fields, described above, by selecting an operation from a WSDL file, as shown below in Figure 7-3.

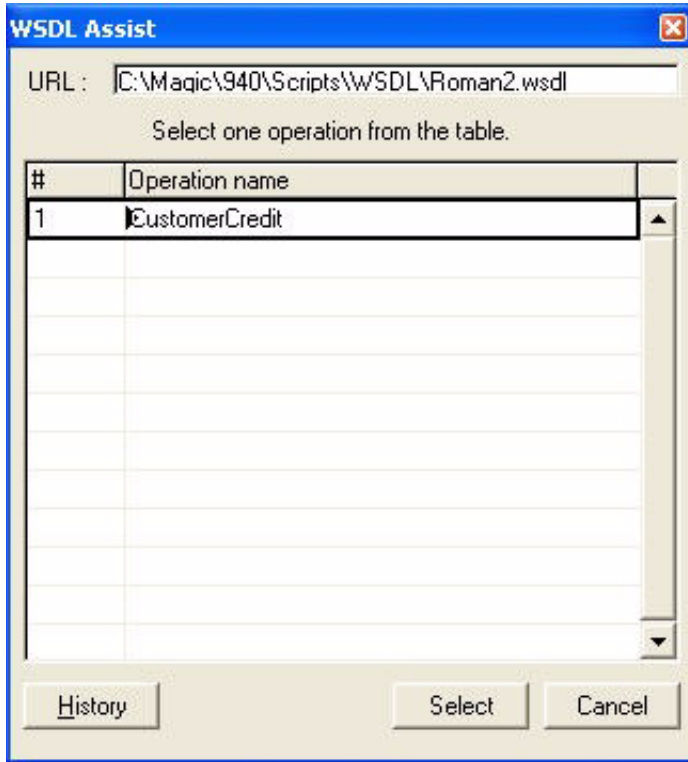


Figure 7-3 : WSDL Call to a Web Service

From the URL field, you can enter a WSDL file or click **F5** to browse. When an eDeveloper client is accessing a web server that requires a user name and password, the URL should be `HTTP://User:Password@[URL]`. You can also use secret names, for example:

HTTP://%user_secretname%:%pass_secretname%@[URL].

The operations in the WSDL file are displayed in the columns below:

- # - eDeveloper automatically generates an internal identifier.

- Name - eDeveloper displays the Operation names entered in the WSDL file. You can modify the name if the argument header is In or In/Out.

You can click History to display a list of called WSDL files.

After you have selected a current WSDL file or have entered the URL for a new WSDL file, choose an operation and click Select to enter the values into the Web Service dialog box.

WSDL Arguments

You can zoom from the Arg (Arguments) field to specify the WSDL argument information listed below:

- Var - Zoom to select the variables or parameters for the argument.
- Exp - Zoom to create an expression that passes the argument when the conditions are met in runtime.
- Description - Displays the WSDL argument description.
- Remote Name - You can specify the remote server.
- WSDL Name Space - You can specify the WSDL Name Space identifier.
- Xsd Type -You can specify the XML data type.
- Mode - Determines whether the WSDL argument is server input, output, or input and then output.
- Header Information - When the Header Information check box is selected, the argument data is sent in the SOAP header. If you do not want to send the data in the SOAP header, you can delete the argument is displayed in the SOAP header by clicking **F3**.

WSDL Assist Usage Considerations

- You can select only one operation. If no operation is selected, eDeveloper updates the Web Service dialog box with the values from the first operation.
- If you are updating values manually in the Call Web Service dialog box or by selecting another Web Service operation in the WSDL Assist dialog box,

eDeveloper displays a confirmation message to override the current Call Web Service values.

Evaluate

Purpose

This operation executes an Action function by evaluating the specified expression that comprises such a function.

The Actions are a special category of functions that perform an input/output-related action every time eDeveloper evaluates them. Each Action returns a logical value in the Return variable to report about success or failure in execution.

Usage

The action functions are:

- Blb2File - saves a Blob object to a file.
- DbDel - deletes a database table.
- DbCopy - copies a database table.
- DbReload - loads a resident table during runtime.
- DDEGet - returns a string value from a DDE server.
- DDEPoke - provides the identifier of the DDE service.
- DDEExec - transfers a command string from eDeveloper to a DDE server.
- Delay - causes a delay in processing.
- File2Req - sends file specifications to the requester.
- FileDLG - accesses a Windows Open File dialog and returns a file name.
- GetParam - accepts values passed as global variables to programs.

- GroupAdd - assigns a user to a user group in the Security file from within an application.
- INIPut - sets a property in the MAGIC.INI file. For more information refer to Chapter 8, Expression Rules
- IOCopy - copies an I/O file.
- IODel - deletes an I/O file.
- IORen - renames an I/O file.
- KbPut - simulates keyboard entry, including short-keys, and to execute System Actions.
- Logon - allows you to logon as a user to the current application.
- MAXMagic - maximizes the eDeveloper Window.
- MINMagic - minimizes the eDeveloper Window.
- MMStop - causes handling of a current running record to stop.
- MnuCheck - displays or hides a check mark in front of a menu entry.
- MnuEnabl - enables or disables a menu entry.
- MnuShow - hides or shows a menu entry.
- ResMagic - restores the eDeveloper window to normal size.
- RightAdd - assigns a Right to a user in the Security file from within an application.
- Rollback - allows rollback of a transaction to a specified nesting level save point.
- SetLang - sets the active language.
- SetParam - sets global variables.
- UDF - calls a User Defined Function.
- UserAdd - adds a user record to the Security file from within an application.
- VARSET - sets a variable to a given value.

Additional information on the action functions can be found in Chapter 8, Expression Rules.

Evaluation Operation Properties

Identification Number of the Expression

The expression number to be evaluated is required. Zoom to the Expression Rules repository to select or define the function you want.

Name

After you select the expression, the first part of its text appears in the Name property for display only.

Range

The Range property is irrelevant to the Evaluate operation and the insertion point skips it.

Ret (for Return Code)

The Ret property is optional. It allows you to specify a variable that will receive the return code indicating whether the action function included in the expression succeeded or failed. At runtime, after execution of the Evaluate operation, the variable specified in the Ret property will receive the value True if the operation was successful, and the value False if the operation failed.

The Ret property will usually receive a Virtual variable. The Virtual variable should be defined as Logical in the Local Variable repository.

Flow

This property is relevant only when you define the Evaluate operation in the Record Main Operation repository of Online tasks. The purpose of the Flow property is to allow you to make the execution of the operation dependent on how the end-user moves the insertion point on the task's main window.

Cnd (for Condition)

This property contains a logical condition that controls whether or not the operation is executed.

Usage Considerations

- The Evaluate operation is significant only if you define an action function in the expression to be evaluated. Otherwise this operation is ineffective.
- Inside the expression you can combine as many action functions as you need. If you do so, the Ret code of the operation will evaluate to True only if all the action functions have succeeded.

Update

Output to assign a value to a variable.

Usage

- To place a particular value in a variable.
- To accumulate totals in reports and Online tasks.
- To update variables in Batch update tasks.
- When copying records between tables.
- To maintain data integrity, through the use of the non-abortable incremental update.

Update Operation Properties

Variable Identifier Letter

The identification letter of the variable is required.

It is the letter of the variable as it appears in the Variable list. This variable is referred to below as the updated variable. It is possible to update variables originating in ancestor tasks.

Name

After you have selected the variable, its name is displayed in this column.

Identification Number of the Expression

The Identification number of the expression is required. It is the number of the expression as it appears in the Expression Rules repository. The result of this expression will be posted to the updated variable.

The expression should be defined for either Normal or Incremental update according to the guidelines explained in the following section on the property How:

Update Expression for Normal Update

The expression value replaces the current value of the updated variable. For example, if you want to keep accumulating the value of B into a sum in the variable A, define $A+B$ as the update expression.

Update Expression for Incremental Update

This method is intended to preserve data integrity when working with linked files. The expression value increments the current value of the updated variable, as explained below. Therefore the update expression must specify only the increment value. For example, to increment the updated variable A by the value in B, you must specify the expression: B, and not the expression: $A+B$.

How: Normal (default)

The available values for the How property are:

- Normal - The updated variable is replaced by the evaluated value of the update expression. This is the normal way to update a variable.
- Incr (for Incremental) - The incremental update is relevant only for Online tasks. For Incremental, eDeveloper evaluates the update expression and

adds this value to or subtracts this value from the updated variable according to the following rules:

1. If the current mode of operation of the task is Create, eDeveloper adds the value of the expression to the updated variable.
2. If the current mode of operation of the task is Modify, eDeveloper subtracts the Old^a value of the update expression from the updated variable, then adds to it the New^b value of the update expression.
3. a) The Old value of the update expression means: the value resulting from the update expression evaluated when the values of its component variables are those present when the record is read.
4. b) The New value of the update expression means: the value resulting from the update expression evaluated when the Update operation is executed.
5. If the current mode of operation of the task is Delete, eDeveloper subtracts the value of the update expression from the updated variable.
6. If the updated variable is a variable selected within a link definition, and if the link criteria are changed during the record interaction, resulting in a new linked record being read, eDeveloper will still maintain the integrity of data as follows: eDeveloper will decrement the Old value of the update expression from the original variable of the original linked record, and eDeveloper will increment the new variable from the newly-linked records by the New value of the update expression.

Example of Incremental Update Usage

Assume you have two files: a receive file, and a linked item file. You want to update your stock count, which is kept in the Stock variable in the linked item file by the amount of stock you receive, which is kept in the Quantity variable in the receive file. Stock is variable A and Quantity is variable B. The Operation is: Update A with Expression B in Incremental mode. If the contents of Stock (A1) is 30 when you start, then:

- In Create mode, if Quantity (B1) is entered as 5, Stock will end up (A2) as 35.
- In Modify mode, if Quantity (B1) is currently 5 and you modify (B2) to equal 7, Stock will be $35 - 5 + 7 = 37$, that is $(A2) - (B1) + (B2) = A3$.
- In Delete mode, if you delete the Receiving transaction record and it still has 7 (B2) in the Quantity variable, stock will be $37 - 7 = 30$, that is $(A3) - (B2) = (A4)$.
- Assume that you did not yet delete the record with Quantity = 7 and that Stock = 37, and then you change the linked Item Record to another one whose Stock = 50. The result will be: Stock of the original record will be $37 - 7 = 30$ and Stock of the new linked record will be $50 + 7 = 57$.

This special behavior of Incremental update saves you having to implement additional procedures to preserve data integrity.

Undo: Yes (default)

The Undo property determines whether the Edit/Cancel F2 command is available to the end-user after the execution of the Update operation.

When Undo is set to Yes, the end-user can Select Edit/Cancel after an Update operation has been executed, to reset the current record dataview to its original state and so cancel the effect of the update.

No. When Undo is set to No, Edit/Cancel is disabled after the Update operation. The end-user cannot undo the changes made to the record dataview at the task level of the updated variable.

- This hard update is generally used in the Record Suffix level, in cases where resetting the updated variable to its original value could damage data integrity. A typical case is when the operation is intended to change data in the record dataview of a parent task. The Undo option set to N will prevent the user's canceling the parent record after the subtask's records have already been accepted.

- Just one Update operation with Undo = N in any one of the subtasks updating the parent record dataview variable is sufficient to prevent the user undoing the update.

Flow

This property is available only when the Update operation is defined in the Record Main Operation repository of an Online task. The purpose of the Flow property is to allow you to make the execution of the operation block dependent on the end-user's interaction with the calling task's main window.

The Flow property contains two columns, each containing a one letter code:

The first column specifies which task "Interaction Mode" conditions the execution of the operation. Basically, Flow modes are related to the end-user's insertion point movements. The default setting for Flow in Update operation is C (for Combine). You can zoom from this first column to see (and optionally set) all the available Flow Modes: Step, Fast, Combined (means Step and Fast), Before and After.

The second column specifies which "Insertion point Direction" movement conditions the execution of the operation. Its default setting for Update operations is C (for Combine). Zoom from this column to see (and optionally set) all the available Insertion point Directions: Forward, Backward and Combine (which means Forward and Backward).

Cnd (for Condition)

The Cnd property contains a logical condition that determines whether the operation execution occurs or is bypassed.

Usage Considerations

- Use the Update operation in the non-interactive levels of a task for all levels except Record Main. Set Undo to N.
- The Update operation on real variables can be used only in the Record level. Record data is not available in any other level, and should not be updated directly.
- The Incremental Update operation is relevant for Online tasks only. In Batch tasks, always use the Normal Update operation.

- When you use the Incremental Update operation, always place it in the Record Suffix Operation repository. This way, eDeveloper executes the Update operation once and only once after the record has been edited.
- Use the Incremental Update operation to accumulate totals in parent task or in variables of linked records. In these cases, set the Undo property to N to prevent end-user cancellation of the parent records after the subtask records have already been accepted.
- In general, variables selected from a linked file are affected by the Update operation in the same way as variables selected from the main file. An exception is the case where an Incremental Update operation is used in a Modify Online task on variables selected from a linked table, where the link expression changes while processing the record. In this case, the original value of the linked variable in the previous record is decreased by the original value of the update expression of the Update operation, and the link variable in the new record is increased by the new value of the expression.

For example, in an Online Order Entry program running in Modify mode, the user may replace the customer number, which is used as a link expression to the customer's file in order to accumulate the orders' total value. In this case when eDeveloper executes the Update operation, eDeveloper subtracts the total for the current (modified) order from the original customer record and adds it to the new customer record, regardless of whether any other changes in the order's total value have been made.

- As a rule, do not use constants in the update expression for an Incremental Update operation. When the task is in Modify mode, such constants are canceled by the process of decrementing and incrementing. However, such an expression will have full effect while the task is in the Create or Delete mode. For example, the constant "1" may be used as the expression of an Incremental Update operation, to keep count of the number of records in a subtask's dataview, where the count is held in a variable selected in a parent task or from a linked file in the current task. As long as the subtask is in Modify mode, no change will occur to the count, but when a record is deleted from the subtask's dataview or a new record is created in it, the count will immediately be updated correctly.

Comparison of Update Assignment with Init Assignment

Both the Update operation in Normal mode and the Init expression of the Select operation can be used to assign a value to a variable. Sometimes they work the same way.

However, there are important differences in their usage:

1. eDeveloper executes the Update operation when it encounters it. If the operation is placed in the Record Main Operation repository, the end-user causes the operation to execute by moving the insertion point, refer to Chapter 5, Application Engine. If the Update operation appears in the Record Suffix Operation repository, eDeveloper executes it during the execution of the other Record Suffix operations. In contrast, an Init expression is always computed and assigned.
2. A Normal Update expression is not recomputed and assigned when its component variables change (procedural operation), while Init expressions are recomputed and assigned as soon as their component variables change (non-procedural operation).
3. An Update expression is evaluated in Create and Modify mode as well as Query mode if the environment setting Allow update in query mode is set to Yes, while Init expressions are assigned to Real variables only if the task runs in Create mode.

Output Form

Purpose

Output Form writes a record to an output text file or to an output device according to the layout specified in the related form. Since a report is an output text file, and its layout is specified in a form, this operation is essential for printing reports.

If the output form is an HTML Merge form, the operation merges data with an independent Internet HTML template file. For more information on the HTML Merge, see Chapter 10, Output Forms.

Usage

- Production of reports
- Exporting user data in text format
- Communication with external Device drivers
- Merge data from an eDeveloper task with a predefined HTML template file containing predefined eDeveloper merge tags.
- Include eDeveloper predefined tags into an HTML template developed independently of the eDeveloper application.
- Implement the eDeveloper Web Online feature, using event handlers activated on input fields on an HTML Merge form.

Output Form Operation Properties

Form Identification Number

The Output Form Identification Number property is required and shows the Identification number of a Form in the Form list. The class of the form you specify here must be greater than 0. The Form you specify is expected to define the format of the output record. It can represent the layout of a text file or an area of a report (Header, Footer, Page Header, Page Footer or Detail). To specify the Output Form Identification Number, zoom to the Form list and select the Form you want. Note that if you need to define a new form, you must go to the Forms repository from Task/Forms or from the Forms toolbar button.

Name

After you have selected the Output Form, its name is displayed in this column.

I/O Number

The I/O Number is required. It specifies the number in the I/O File list, of the device to which you want to write the form. Its default setting is 1, the first

device. For HTML Merge and Web Online Response forms, the I/O file must be either a Requester or a Standard File.

The device you specify must have its Access property set to Write or Append in the I/O File repository. The Access property of an I/O file used by an HTML Merge form must be set to Write.

You can specify a device defined (that is, opened) in an ancestor of the current task.

Zoom to the I/O File list and select the I/O File you want. If you need to define a new I/O File, zoom from the I/O File list into the I/O File repository. For more information on the I/O File repository, refer to Chapter 6, Programs.

Pag (for Page)

The End of Page Behavior property is relevant for reports to a Printer or Console Media only. This property determines eDeveloper's behavior whenever an End of Page state occurs.

The End of Page (EOP) state occurs when the form about to be output is larger than the space remaining on the current page. The page size, number of lines, that eDeveloper uses for evaluating the EOP state is logical, and is not related to the physical page size of any printer. You can use the Printer repository to establish the relationship between the printer's physical and eDeveloper's logical page sizes. This logical page size is determined either by:

- The Rows property of the I/O File associated with the Output Form operation (refer to I/O File repository in Chapter 6, Programs).

If rows are not specified, then by:

- The Lines property specified for the printer associated with the I/O file used for the current Output Form operation, refer to Chapter 2, Settings.

The End of Page state is cleared by the next Output Form operation that causes a page jump. Output Form operations specifying Automatic or Top (see below) clear the EOP state indicator.

Because eDeveloper allows simultaneous output to multiple I/O files and output media, the End of Page state is specific for each device. Therefore, an

End of Page state on one I/O file does not affect any Output Form operations to other I/O files.

The Page property has three possible values:

- Auto (for Automatic) - This is the default. When Auto is specified, eDeveloper jumps to a new page at the End of Page indicator, and then automatically reprints all the header forms having the same class number as the form printed by the current Output Form operation, and which precede it in the Form repository. Header forms appearing after the current one will not be printed at the top of the new page. Auto is the normal choice for report programs.
- Skip - When Skip is specified, before printing the current form to the output file, eDeveloper checks if there is sufficient space left on the current page to accommodate the Form in full. If enough space is available, eDeveloper writes the Form to the output media. Otherwise, it skips the form and does not write it to the output media. Instead, the End of Page state indicator is switched on. You can query this indicator with the EOP function.
- Top - When Top is specified, eDeveloper issues a form feed to the printer before sending the form for printing. This causes the form to be printed on at the top of a new page. This is a way to force page breaks. In this case, the headers are not automatically reprinted.

Note: It is up to you, as a developer, to define what action should be taken based on an End of Page indicator. For example, you can force a page break, using an Output Form operation with Top as its Page setting, and continue with normal output.

When the End of Page indicator occurs, any further output of forms using Skip mode will be ignored. Output with Skip mode will resume when the EOP state is cleared.

Use the Skip mode for applications where you require complete control over output. In such cases, you can condition Output Form operations based on the EOP and Line functions.

Flow

The Flow property is available only when you define the Output Form operation in the Record Main Operation repository of an Online task. The purpose of the Flow property is to allow you to make the execution of the operation dependent on how the end-user moves the insertion point on the task's main window.

Cnd (for Condition)

The Condition property contains a logical condition that controls whether or not the operation is executed.

Usage Considerations

- You can use the Program Generator to automatically generate a data output program.
- The purpose of the Output Form operation is to handle external text files, not eDeveloper DB tables.
- The external text file in the Output Form operation may be re-directed to a console, a disk file, a printer, or a device driver, refer to the I/O File repository in Chapter 6, Programs.
- When you design reports, you should define a separate Output Form operation for each part of a report format, e.g., Headers, Footers, and Detail lines. You can define Output Form operations for printing all parts of a report in the task where the text output file is defined (in its I/O File repository) or in any of its child tasks.
- To reset the EOP condition you must execute an Output Form operation with its Page setting defined as Automatic or Top.

Input Form

Purpose

To read a section from a text file or an input device and to load the contents into variables defined in the related form.

Usage

Importing user data in text format.

Communication with external Device Drivers.

Input Form Operation Properties

Identification Number

The Input Form Identification Number property is required and shows the identification number of a Form in the Form list. The class of the form you specify here should be greater than 0. That is, you should not use the Form number of the Main window of the task. To specify the form into which to read the I/O file, zoom to the Form list and select the form you want. If you need to define a new Form, zoom from the Form list into the Form repository. For more information, refer to Chapter 10, Form Concepts.

Name

After you have selected the Input Form, its name is displayed in this column.

I/O File Identification Number: (default)

This I/O File Identification Number is required. It specifies the number of the I/O file, as it appears in the I/O File list, from which you want to read. Its default setting is 1.

The I/O file you specify must have its Access property set to Read or Direct.

You can specify a file defined (that is, opened) in an ancestor of the current task. If you need to define a new I/O file, zoom from the I/O File list into the I/O File repository. Zoom to the I/O file list and select the file you want.

For more information, refer to Chapter 6, Programs.

Field Delimiting Method > Column

The Field Delimiting Method (Dlm) property defines how the fields of the input record are delimited. The possible values are:

- **Column** - This is the default. When Column is specified, the input fields are located in fixed columns without delimiter characters. The Form used in the operation must match this layout exactly, including the spaces between the fields in the record.

When you specify Column, the Char property that follows is irrelevant and the insertion point skips it.

- **Single** - When Single is specified, the fields in the input record are separated by a single delimiter, added after (to the right side of) each input data item. For example, in the input record.

"123, ABCDE, -23.57", the character comma (,) is used as the delimiter and you will have to specify it in the Char property.

When you specify single, the spacing between fields in the form is not significant.

- **Double** - When Double is specified, the fields in the input record are separated by a double delimiter, one added before and one added after (to the left and right sides of) each input data item. For example, in the input record "123" "ABCDE" "-23.57", the character Quote (") is used as the delimiter and it is specified as such in the Char property.

When you specify Double, the spaces between the fields in the file record and those between the variables in the Form are not relevant.

The Input Form operation with a single delimiter automatically strips leading and trailing double quotes from fields of the input file. For example, the following input

"A","B","C"

when read with the Input Form operation using ',' (comma) as the single delimiter character, will read the values A, B, and C, not "A", "B", "C", into the corresponding eDeveloper variables.

Chr

The Chr property contains the actual character used as the delimiter. This property is available only when you set the Dlm property to Single or Double.

Flow

This property is relevant only when you define the Input Form operation in the Record Main Operation repository of an Online task. The purpose of the Flow property is to allow you to make the execution of the operation dependent on how the end-user moves the insertion point on the task's Main window.

Cnd (for Condition)

The Condition property contains a logical condition that controls whether or not the operation is executed.

Usage Considerations

- You can use the Program Generator to automatically generate a data input program.
- Side Effect: Loading data from an input text file into a variable that is part of an Init or Link expression triggers their non-procedural recomputing.
- Be sure to specify a separate Input Form operation with a separate form corresponding to each different record layout present in the input text file.
- When used in Batch tasks using the 0 Scratch file, you have to define an End Task expression in the Task Properties dialog to terminate the execution of the Batch task either when the input text file has reached its end, using the EOF (End-of-File) function, or when a predefined number of records has been read from the file.
- You can define a Form with multiple lines, to read an equivalent number of physical records from the external input text file using one Input Form

operation. Use this technique when you are copying data from an input text file to create a DB table, and each row of the DB table is based on data spanning several lines of the input text file.

- Each line in a form can have a maximum width of 9999 bytes. Thus, you can use the Input Form operation to read input records containing up to 9999 bytes.
- The maximum Content Size of one Form, that is, the product of lines multiplied by columns, is 32KB.
- The numeric fields of the input text file records may include minus signs, commas and decimal points (edited numeric values).
- For each Input Form operation, eDeveloper will attempt to read from the Input file records whose length equals the width of the form used with the operation. If a line terminator appears before the Form Width number of characters have been read, eDeveloper will terminate the Input Form operation at this point. For more information about line delimiters in Input files, refer to I/O Files in Chapter 6, Programs.

Browse

Purpose

To display a text file in a pop-up window and allow the end-user to scan or edit the text file.

Usage

- To display messages stored in text files.
- To show or edit parts of text files.

Browse Operation Properties

Identification Number of the Browse File Expression

The Identification Number of the Browse File Expression property is required. It is the number of an expression in the Expression Rules repository that at runtime returns a string with the name of the text file to be browsed. This file name can contain a filename, including server and OS path.

Unlike the Output Form and Input Form operations, the I/O file for the Browse operation need not be defined in the I/O File repository.

Name

After you have selected the expression, the first part of its text appears in the Name property, for display only.

Edt: (for Browse File Edit) > Scn

The Edt (Browse File Edit) property specifies whether the user can scan only or modify the text file. The possible values are:

- Scn (for Scan) - This is the default. When Scn is specified, eDeveloper displays the text file for browsing (viewing) only.
- Edt (for Edit) - When Edt is specified, the end-user can edit the file by utilizing the same Screen Editor you use to define Help Screens, refer to Chapter 14, End-User Helps.

If you have specified Edt and the record of the text file does not fit into the Form you defined to receive it, eDeveloper switches the Edit property to Scan and displays an error message. This prevents the end-user from making changes that cannot be saved properly.

Browse File Form Identification Number

The Browse File Form Identification Number property shows the Identification number of a form in the Form list. This Form will be used for display of the browsed file. The Form you specify for a Browse file must be of Class 0. If you leave this property empty, eDeveloper uses the Main Form of the task as default.

To specify the Browse File Form identification number, zoom to the Form list and select the Form you want. If you need to define a new form, zoom from the Form list into the Form repository, refer to Chapter 10, Form Concepts.

Locate

This property is irrelevant for the Browse operation and the insertion point skips it.

Flow

The Flow property is available only when you define the Browse operation in the Record Main Operation repository of an Online task. The purpose of the Flow property is to allow you to make the execution of the operation dependent on how the end-user moves the insertion point on the task's Main window.

Cnd (for Condition)

The Condition property contains a logical condition that controls whether or not the operation is executed.

Usage Considerations

- The external text (I/O) file to be scanned or edited must be a disk file, not redirected to any other device.
- When the Browse operation is activated, the external text file to be scanned or edited must be closed, that is, not in use by any external application, or other user on the network, or a parent task.
- The Browse operation is interactive, by definition. Even if it is used from within a Batch task, the user can still interact with the Form window.

Exit

Purpose

To execute an external program or a Batch/Command file at the Operating System level. eDeveloper suspends the execution of the current task when it encounters an Exit operation. When the external program or Batch/Command file completes, eDeveloper resumes the execution of the task at the next operation.

Usage

Executing processes outside the scope of eDeveloper and then returning to eDeveloper; for example, branching to a Spreadsheet or a Word Processor.

Exit Operation Properties

Identification Number of the Exit Destination Expression

The Identification Number of the Exit Destination Expression property is optional. It is the number of an expression in the Expression Rules repository that a runtime returns a string containing the name of the Operating System object to be executed. If you leave the default value, 0, you can specify the Operating System object directly in the Name column.

Name

If you selected an expression in the previous property, the first part of its text appear in the Name property and the insertion point skips it when you move right.

If you left the expression property empty (value 0), you can type a string containing the name of the Operating System object to be executed, specified without quotes. This string is usually just the program name. The string can be column width or to the maximum length allowed by the specific Operating System. For example, DOS allows 128 characters.

Wait > No

The Wait property specifies if the eDeveloper program waits for the called program to complete before it continues.

- No - This is the default setting. When No is specified, the eDeveloper program will not wait for the called program to complete before continuing.
- Yes - When Yes is specified, the eDeveloper program will wait for the called program to complete before the eDeveloper program continues.

Shw (for Show) > Normal

The Show property determines the appearance of the external program. The possible values of the Show property are Hide, Normal, Maximize, and Minimize.

- Hide - Hide specifies that the external program will run behind the visible windows. The external program will not be visible.
- Normal - Normal is the default setting. Normal specifies that the external program will run in the top, visible window.
- Maximize - Maximize specifies that the external program will run in the top visible window and that its window will be maximized to the full screen.
- Minimize - Minimize specifies that the external program will run minimized, and that only its icon will be visible.

Note: When the external application runs hidden or silent it is advisable to redirect standard output messages of the external program from the screen to either a disk file or a null file.

Ret > (for Return Code)

The Ret property is optional. It allows you to specify a numeric variable that will receive the return code from the Operating System of the executed program. If the returned value is greater or equal zero, the external program was successfully executed by eDeveloper, and the return code should be interpreted according to the specifications of the operating system. If the value returned is -1, eDeveloper could not allow execution of the external

program, usually because the wrong name or wrong path was specified, or insufficient memory was available, or the command processor was not found.

When you specify a Ret property, make sure there is a Select Virtual operation for the receiving variable. The Virtual should be defined as Numeric in the Local Variable repository.

Flow

The Flow property is available only when you define the Exit operation in the Record Main Operation repository of Online tasks. The purpose of the Flow property is to allow you to make the execution of the operation dependent on how the end-user moves the insertion point on the task's Main window.

Cnd (for Condition)

The Cnd property contains a logical condition that determines whether or not the operation is executed.

Raise Event

The Raise Event operation runs a handler that has been defined for a specific event. This operation enables the user to force events through various mechanisms during the flow of the eDeveloper engine. These events will be handled the same way as other events generated by the system.

The interaction between an application and its components often requires handlers from the host application. eDeveloper now lets components raise an event by using the user event's public name as specified in the host application.

Raise Event Properties

The properties for the Raise Event operation are described below.

Event Name

This property lets you define the type of event or trigger that is used in the Raise Event operation. Zoom from the Name column to specify an event type and name from the Event dialog.

The event types are:

- System - Events triggered by defined keystroke combinations.
- Internal - eDeveloper internal actions.
- User - Events defined in the User Events repository.
- Public Event - A selected expression number that represents the public name of a user event. The expression must be a string attribute.

The Expose property must be selected in the User Events repository of the main program in the host application to let programs from other components call a user event handler from the host application. For more information about the Expose property, see User Events on page 416.

Wait

The Wait property has three possible options: Yes, No, and Expression. This property indicates whether the event handler or trigger should be performed immediately or placed at the end of the event queue. The default setting is Yes.

Wait=Yes

When a Raise Event operation is performed with the Wait property set to Yes, it is similar to a Call operation, where the event handler is invoked before executing the command following the Raise Event operation. The status of the task cannot be changed, and the Force Exit property of the event that is raised will be ignored.

The search mechanism also resembles a Call operation. When the operation is performed with a handler that is executed in a parent task, the search for the appropriate event handler will be undertaken with the parent handler's task as the current task. For example, if the parent task has a Task Scope handler on

the event that was invoked by the Raise Event operation, the Task Scope handler will be executed.

Wait=No

When a Raise Event operation is performed with the Wait property set to No, the invoked event is written into the event queue, but is not performed immediately. After the execution of the current handler is completed, and only when the eDeveloper engine reads events from the queue, will it search for the appropriate handler for the raised event, using the search mechanism outlined above.

Wait=Expression

When the Wait property is set to an expression that is True, the handler or trigger for the event will be performed immediately. If it returns False, the invoked event is written into the event queue as described above.

Arguments

This property defines the arguments that will be sent to the event handler once an event handler or trigger is invoked and is active regardless of whether the Wait property is set to Yes or No. The argument field displays the number of arguments passed from the raised event. The user can zoom from this field to bring up the Argument list containing the following columns:

Var (for Variable) - The Var column is used to identify a variable or expression to be passed to the subroutine, with the option to receive back data from the subroutine. This method of passing an argument is called *by-reference*. The other method, *by-value*, is explained below, under Exp.

Zoom to the Variable list to select a variable, or type the letter code of the variable and go to the next variable. The letter code is displayed in this column.

If you want to pass a constant value as an argument, skip the Var column and move to Exp.

The Var column is skipped if an expression is already defined.

Exp - The Exp column is used to pass a constant value to the subroutine by using an expression. This is the by-value method of passing an argument to a

subroutine. Note that you cannot receive data back from the subroutine in this property. The Expression number appears in this column and the Name column shows the first section of the expanded expression text. You can zoom to the Expression Rules repository to select an existing expression or to define the expression you want to pass. The expression may evaluate to any valid eDeveloper data type.

The Exp column is skipped if a variable is already defined as an argument. Remove the variable identifier if you want to use an expression as an argument instead.

Description - The description of the passed variable or expression.

Arguments can be sent to the Raise Event command only if the Wait property is set to Yes.

Flow

The Flow property is available for the Raise Event operation in the Record Main.

Cnd (for Condition)

The Cnd property contains a logical condition that determines whether or not the operation is executed.

Destination Context Name Property

This property lets contexts communicate with each other on an enterprise server by using handled events. You can specify the Destination Context Name expression to raise the event in a specified context. The context name is defined by using the CtxSetName function. Contexts are also able to share resources by using shared values and database files.

This property is available for batch tasks only.

Raise Public Event Runtime Behavior

When the Raise Public Event operation is executed, the expression defined for the operation is evaluated. The evaluated string is regarded as the public name of a user event. The public name is case sensitive.

eDeveloper automatically looks for a corresponding user event handler defined with the user event public name in the runtime tree for the current task of the task tree. eDeveloper searches for the handler from the current task up to the main program of the top application. The corresponding handler that is found will be executed.

No event will be executed when the Expose property is not selected or no corresponding handler is found for the host application.

After the first corresponding handler is found and executed, the Propagation property is selected. If the property is False, the handling of the event is completed. If the property is True, eDeveloper will propagate the user event. The Propagation property does not use the event's public name but the reference to the actual user event as specified in the User Events repository.

Expressions may include literals, operators, variables, and special functions.

In this chapter:

- | |
|---------------------------------------|
| • eDeveloper Expressions Overview |
| • Function Tables |
| • Alphabetical Directory of Functions |

Literals

Numeric values are written without any special identification. For example, 123 and 15.52 are numeric literals.

Literals other than numeric values are treated as alpha strings, and are normally enclosed within single quotes. All alpha strings, except for plain text, include a literal code that identifies the type of the data enclosed within the single quotes. The literal code, however, is itself not enclosed in quotes, and immediately follows the string to which it refers. For example 'abcd' is treated as plain text, while '01/01/92'Date is treated as a date.

eDeveloper translates literal strings into an eDeveloper internal representation while it parses an expression containing the string. At runtime eDeveloper uses the actual internal values for expression evaluation. As internal values, eDeveloper keeps track of any changes that may occur to the values represented by the literals.

For example, the string '1'FILE represents the first entry in the Table repository. If a new entry is inserted before the first entry, causing it to become the second entry, eDeveloper will automatically update the literal string to read '2'FILE. This will preserve the original meaning of the expression.

Another use of literals is to improve portability of the eDeveloper program from one natural language environment to another. For example, it may be desirable for the task mode mnemonic 'MCDQLRUSFOE' to have different meanings in different languages. By specifying the MODE literal as in 'MCDQLRUSFOE'MODE eDeveloper will automatically translate the valid task modes within the string to their appropriate values in the specified target language.

Literals make expressions more robust and less susceptible to unintentional damage due to changes in other parts of an application.

Following is the list of available literal codes:

ACT

Example: 'Exit'ACT

The string is interpreted as an eDeveloper action identifier. Actions are the instructions that eDeveloper received as a result of a user interaction. Actions are generated by translations of key strokes through the Keyboard Mapping repository. The eDeveloper actions can be used in expressions for purposes of testing user activity, and creating macros. Action names can be seen in the settings/keyboard mapping action column. These names should be used in the action string. It is enough to type the first few characters of the action descriptor and eDeveloper will complete the literal. In cases of ambiguity, type more letters.

Date Example: '01/01/97'Date

The string is interpreted as a date. It may participate in arithmetic operations because its internal representation is of a numeric value. '01/01/97'Date+14 is a valid expression which yields the date 15/01/97.

The Century entry in the Environment repository affects results of Date actions. '01/28/97'Date will be interpreted as a date in 1997 if Century contains a year greater than 1892 and less than or equal to 1997, and it will be interpreted as a date in 2097 if Century contains a year greater than 1997 and less than or equal to 2097. eDeveloper expands year values to a full four digit equivalent. Refer to Chapter 2, Settings for a full discussion of Environment settings.

EXP Example: '3'EXP

Declares the string as an expression identifier. Used as input with the ExpCalc function. The EXP literal will allow any relocation of the expression it represents.

FILE Example: '1'FILE

The string is interpreted as the sequence number of a data table in the Table repository. The use of the FILE literal enables eDeveloper to update the table number as its relative position changes within the Table repository. '5'FILE refers to the fifth line in the Table repository at the time of creation of the entry. eDeveloper will automatically maintain the correct number.

FORM

Example: '2'FORM

Declares the string as an identifier of a specific form defined in the forms repository. Used as input in the Main Display property of a task in order to dynamically display a different form. The FORM literal will allow any relocation of the form it represents.

HEB

Example: HEB'xx'

The string is processed as a Hebrew string with a right-to-left orientation in every process it participates in. If assigned to the display, it receives a Hebrew attribute.

KBD

Example: 'F2'KBD

The string is interpreted as a keyboard value. Keyboard values are the raw input to eDeveloper. The keyboard values are translated to actions according to the keyboard mapping repository. The same key may generate several different actions according to context. KBD values may be used in expressions for the purpose of testing user input and creating macros. KBD names can be seen in the assigned key column of the settings/keyboard mapping. These names should be used in the strings KEY.

KEY

Example: '5'KEY

The string is interpreted as the sequence number of an index in the Index repository. The use of the KEY literal enables eDeveloper to update the index number as its relative position changes within the Index repository. '5'KEY refers to the fifth line in the Index repository at the time of creation of the entry. eDeveloper will automatically maintain the correct number.

LOG

Example: 'TRUE'LOG

The interpreted value of the string is either True or False (1,0 respectively). The allowed strings are 'TRUE'LOG and 'FALSE'LOG. Any other string with the LOG literal will be converted to 'FALSE'LOG.

MODE

Example: 'MC'MODE

The string is treated as a task operation mode. The available modes are all the valid entries for the Initial Mode field in the Task Properties repository. When an eDeveloper program is ported to another natural language environment, the MODE literal can be used to preserve meaningful mnemonics in the new language.

Prog

Example: '5'Prog

The string is interpreted as the sequence number of a program in the Program repository. The use of the Prog literal enables eDeveloper to update the program number as its relative position changes within the Program repository. '5'Prog refers to the fifth line in the Program repository at the time of its creation. eDeveloper will automatically maintain the correct number.

Right

Example: 'Right #4'Right

Declares the string as a rights identifier. Used as input with the Rights function, allowing you to query whether a user has the given right.

If the developer does not own a right, its literal will appear.

Time

Example: '14:30:15'Time

The string is interpreted as a time value. It may participate in arithmetic operations since its internal representation is a numeric value.

'14:30:15'Time+5 is a valid expression. Values added to a time value are treated as seconds.

VAR

Example: 'A'VAR

Declares the string as a variable identifier. Used as input with the VarAttr, VarCurr, VarMod, VarName, VarPrev, and VarSet functions. The VAR literal will allow any relocation of the variable it represents due to selection of other variables.

Operators

In eDeveloper expressions, you can use mathematical, logical, and string operators.

Mathematical Operators

+	addition
-	subtraction
*	multiplication
/	division
^	exponentiation
-()	unary negation; changes the sign of a numeric expression

Logical Operators

All expressions involving logical operators return a logical True or False.

=	equal to
<>	different than (not equal to)
<=	less than or equal to
<	less than
>	greater than
>=	greater than or equal to
AND	Returns True only if both operands are True.
CASE	Switches between values.
IF	Evaluates a logical expression and returns one value if True (Then) and another if False (Else).

IN	determines whether a selected value matches any value in a given collection of values.
LIKE	Determines whether a given character string matches a specified pattern.
MOD	modulus; returns the remainder of an integer division. For example, A MOD B returns 3 if A=13 and B=10.
NOT	reverses the logical value of its operand
OR	returns True if either operand is True

String Operator

Use the & symbol to concatenate alpha strings. For example, 'John'&'Smith' yields 'JohnSmith'.

Variables

An expression may contain variables. Variables are real and virtual variables selected in the current task and its ancestors. Variables are identified by an alphabetical code representing their position in the program hierarchy (where A is the first selected variable, B the second selected variable, BA the 27th selected variable, and so on). Variables created in a parent task are listed prior to variables in the current task. The variable identifier can be selected from the Variable list window always open when editing an expression.

When evaluating an expression, eDeveloper replaces the variable's code with its value. Therefore, the variable represents a value. eDeveloper will automatically update the variable code in an expression if the variable was relocated.

If in an expression a variable is required and not the value it represents, the VAR literal should be used to indicate to eDeveloper the proper use of the variable identifier.

Functions

Functions perform specialized operations such as setting conditions, converting and manipulating data, accessing eDeveloper fields, returning a result, or performing certain actions.

The tables following the Dynamic Data Exchange section list eDeveloper functions according to the following subjects: Action, Alpha/Numeric Conversion, Alpha Strings Manipulation, Application Partitioning, Comparison, Database Interface, Date, DB Table Management, Dynamic Data Exchange, Identification and Environment, I/O Functions, Language, Mathematics and Trigonometry, Menu, Numeric Value Manipulation, Specialized Test, Time, User, Variable Value Manipulation, and View functions.

The last section of this chapter beginning on page page 553 is a complete alphabetic reference to all eDeveloper functions.

Dynamic Data Exchange

DDE implementation in eDeveloper uses the DDEML.DLL that is supplied with Windows 95 and Windows NT. It will not work with earlier versions of Windows. This DLL must be present in the Windows System directory.

eDeveloper's DDE implementation is done through eDeveloper functions. Every function initiates a complete DDE conversation, and terminates the conversation before it returns. This means that the DDE exchange is a "cold" exchange and it is always initiated by eDeveloper. eDeveloper will not be aware of changes made by other applications and applied to data eDeveloper obtains via a DDE exchange.

The DDE server application must be online for eDeveloper to communicate with it. Trying to access a DDE server from eDeveloper without the server being online will not cause the server to load, and the operation will fail.

The DDE servers do not get focus when accessed by a client application. Therefore, if the eDeveloper DDE functions cause an error in the server application, the server application will respond by issuing an error dialog, this dialog will not get focus, and the eDeveloper DDE operation will time-out and fail.

The eDeveloper DDE functions are synchronous. This means that eDeveloper will wait to receive the result of the DDE operation before it continues processing.

Note that DDE functions are not portable. They can be used only under Windows. If you attempt to execute DDE functions under a different operating system DDEGet will always return empty strings; DDEPoke and DDEExec will always return False; and DDERR will always return code 15, meaning that a DDE function was executed on an operating system other than Windows.

Buffer Management

A buffer is a BLOB variable containing a stream of binary data. Buffer management functions let you manipulate the position, size, storage, and length of the supported data types in the buffer.

A buffer can be a bundle of data, such as Alpha, Numeric, and Date values. All data types must begin in a specified position with a selected storage type to create a valid structure.

It is best to use the BufSetCnvPrm function before handling buffers. Various programs and components can change the buffer conversion settings. The BufSetCnvPrm function resets the conversion settings to their default values.

Vector Data

The eDeveloper Vector is an array that lets you store and retrieve data from a specified cell index. The Vector attribute is based on the BLOB attribute with an additional cell model property.

The Vector cell must be specified from an eDeveloper field model, as defined in the Models repository. The model can be any field data attribute: Alpha, Numeric, Logical Data, Time, Memo, BLOB, OLE, ActiveX, or Vector.

Vector indexing starts from one. The Vector attribute can only be selected from Virtual and Parameter fields. You cannot directly store vectors in a table.

Recursive vector definitions are not supported. You cannot put a Vector variable on a GUI or Browser form. It is not recommended to store large amounts of data in a vector because the array is stored in the computer's memory.

You can access and modify the vector cells by using the vector functions described in this chapter.

XML Namespaces

You can use the XML functions when the Namespace Awareness mode is activated or not activated. If no prefix is specified for an element, the Namespace Awareness mode is not activated.

In the Namespace Aware mode, you should specify a prefix (namespace alias) for an element. For each I/O file, an association between an alias and URI is kept internally. For an XML file read by eDeveloper, an association between the alias and namespace URI is created for elements with a namespace. The XMLGetAlias function retrieves the alias related to a specific URI. The XMLSetNS function defines the association between an alias and URI.

The prefix can be specified for each element in the element path. In the Namespace Aware mode, each non-prefixed element is compared according to the default namespace URI.

For example, in element path a.al1:b.c.al2:d

- Root element a has no prefix
- Element b has alias, al1
- Element c has no prefix
- Element d has alias, al2

To deactivate the Namespace Aware Mode, do not specify a prefix for any element in the element path.

XML Namespace Examples

Read XML

This section shows how to use the XML functions to read an XML file with a namespace.

XML file:

```
<? xml version="1.0" encoding="UTF-8" ?>
<person ID="123" xmlns="http://server/person">
  <name>John Doe</name>
  <gender>male</gender>
</person>
```

When opening the XML I/O file, the following associations will exist between an alias and an Namespace URI:

Alias	URI	Remarks
	http://server/person	Default namespace

Update var1 XMLGetAlias(0,1,'http://server/person')	Var1 is updated with an empty string.
XMLGet(0,1, 'person.name','')	retrieves the name, "John Doe" (non-namespace aware mode)
XMLGet(0,1, Trim(var1) &':person.name','')	retrieves the name, "John Doe"
XMLGet(0,1, 'person.'&Trim(var1) &':name','')	retrieves the name, "John Doe"
XMLGet(0,1, ':person.name','')	retrieves the name, "John Doe"

Create New XML

This section shows how to use the XML functions to generate an XML document with a default namespace.

```
<? xml version="1.0" encoding="UTF-8" standalone="no" ?>
<person ID="123" xmlns="http://server/person">
  <name>John Doe</name>
```

```
<gender>male</gender>
</person>
```

```
XMLSetNS(0,1, "", 'http://server/person')
XMLInsert(0,1,'person','', '')
XMLInsert(0,1,'person','ID','123')
XMLInsert(0,1,'person.name','', 'John Doe')
XMLInsert(0,1,'person.gender','', 'male')
```

Note: replacing **person** with **:person** should produce the same result.

Modifying an XML document

This section shows how to use the XML functions to modify a namespace.

```
XMLModify(0,1,'person.name','', 'New name') (non-namespace aware mode)
```

or

```
Update var1 XMLGetAlias(0,1,'http://server/person')
XMLModify(0,1,Trim(var1) & ':person.name','', 'New name')
```

Using a Wrong Alias

```
XMLSetNS(0,1,'al2',' http://server/another\_person')
```

When the Wrong Alias is Used

```
XMLModify(0,1,'al2:person.name','', 'New name')
```

this function call fails, displaying the returning error code -4, **Element path not found**.

When an Alias is Not Associated with a Namespace URI

```
XMLModify(0,1,'al3:person.name','', 'New name')
```

If **al3** is not associated with a namespace URI, this function fails, returning error code -4, **Element path not found**.

Summary of XML Functions

eDeveloper XML functions are:

XMLBlobGet	Returns the value of an XML element or an XML attribute according to its element path.
XMLCnt	Returns the number of occurrences of an XML element or an XML attribute according to its path.
XMLDelete	Lets you delete an XML element or attribute according to its path.
XMLExist	Returns a True value if an XML element or an XML attribute can be located by the XML's element path.
XMLFind	Returns the index of an XML element that has a value equal to a specified value or if one of its attribute values is equal to the specified value.
XMLGet	Returns the value of an XML element or an XML attribute according to its element path.
XMLGetAlias	Retrieves the alias associated with a namespace Uniform Resource Identifier (URI) for the root element.
XMLInsert	Lets you insert an XML element or attribute value in a specified path.
XMLModify	Lets you modify an XML element or attribute according to its path.
XMLSetNS	Lets you create an association between an alias and a namespace.
XMLStr	Converts valid XML string into an Alpha string.
XMLVal	Converts a string value into a valid XML string.

XML General Error Codes

XML general error codes are displayed in the table below.

Error Code	Description
-1	Invalid I/O file
-2	Inserted element or attribute alias is not defined
-3	I/O file not opened in Write mode
-4	Element path not found
-5	Attribute not found
-6	Attribute already defined for element
-7	Invalid Before/After flag
-8	Reference element not found
-9	The document contains a root element. Multiple roots not permitted
-10	Invalid auto convert flag
-11	Invalid path, non-valid index
-12	Alias already used
-13	Invalid alias, qualified name
-14	Invalid argument type (for an optional argument)
-20	Invalid XML file (XML parsing failed)

Function Summary

BASIC see Mathematical Operators, page page 515, and Logical Operators, page page 515.

Compound Storage

Function	Description
----------	-------------

Compound Storage

BlobSize	Returns the BLOB size in bytes.
BufGetAlpha	Converts a value stored in a certain position in a buffer to an Alpha string.
BufGetBit	Returns the value of a bit for a specified byte position in a BLOB buffer.
BufGetBlob	Converts a value stored in a specified position in a buffer to an eDeveloper BLOB.
BufGetDate	Converts a value stored in a specified position in a buffer to an eDeveloper Date value.
BufGetLog	Converts a value stored in a specified position in a buffer to an eDeveloper logical value.
BufGetNum	Converts a value stored in a specified position in a buffer to an eDeveloper Numeric value.
BufGetTime	Converts a value stored in a specified position in a buffer to an eDeveloper time value.
BufGetVariant	Retrieves a variant value stored in a specified position in a buffer.
BufGetVector	Converts a value stored in a specified position in a buffer to an eDeveloper vector value.
BufSetAlpha	Converts an Alpha, Memo, or RTF string variable to one of the supported storage types.
BufSetBit	Sets the value of a bit in a byte stored in the BLOB buffer.
BufSetBlob	Converts a BLOB variable into one of the supported storage types at a specified position in the buffer.

Compound Storage

BufSetDate	Converts an eDeveloper date to a binary value in a specified position and storage type in the buffer.
BufSetLog	Converts a logical value into one of the supported storage types at a specified position in the buffer.
BufSetNum	Converts an eDeveloper number into one of the supported storage types at a specified position in the BLOB buffer.
BufSetTime	Converts an eDeveloper Time value into one of the supported storage types at a specified position in the buffer.
BufSetVariant	Inserts an variant value into a specified position in a buffer.
BufSetVector	Converts an eDeveloper Vector value into one of the supported storages at a specified position in the buffer
SetBufCnvParam	Sets parameters that determine the writing and reading conversion values to and from the buffer.
VariantAttr	Retrieves the eDeveloper attribute corresponding to the variant data type.
VariantCreate	Creates a variant according to the specified value of an eDeveloper data attribute.
VariantGet	Retrieves the value of a specified variant data type.
VariantType	Retrieves the variant data type's storage type identifier.
VecCellAttr	Returns the vector's cell attribute.
VecGet	Returns the value of a specified cell.

Compound Storage

VecSet	Updates the value of a selected cell with a given vector.
VecSize	Returns the number of cells for the given vector.

Conversion

Function	Description
ANSI2OEM	Converts data from ANSI to OEM.
BlobToFile	Saves a BLOB object to a file.
DStr	Date-to-string conversion.
DVal	String-to-date conversion.
EuroCnv	Returns the converted value from currency to currency based on the conversion values of the Euro currency in the European Currency table.
File2Blob	Saves a file to a BLOB object.
File2OLE	Moves a file to an OLE variable.
HStr	Converts decimal to hexadecimal.
HVal	Converts hexadecimal to decimal.
Lower	Returns a string in lowercase.
MStr	Converts number to alpha (any format).
MVal	Converts alpha to number (any format).
OEM2ANSI	Converts data from OEM to ANSI.
TStr	Translate a time value to an alpha character.

Conversion

TVal	Alpha to time conversion.
Upper	Returns a string in uppercase.
UTF8FromAnsi	Converts data encoded in ANSI to UTF8.
UTF8ToAnsi	Converts data encoded in UTF8 to an ANSI string. This function uses code pages defined in the CodePage function. If you are not using the CodePage function, the default OS code page is used in a similar way as with Java functions.
Val	Convert an alpha character to a number.

Database

Function	Description
ClrCache	Clears the database of the current task
CurrPosition	Returns the internal position of the current record
DbCache	Returns the database cache hit ratio
DbCopy	Copies an existing data file to a new file.
DbDel	Deletes an eDeveloper table
DbDiscnt	Disconnects the current database connection.
DbERR	Returns and clears a database error message.
DbExist	Checks existence of the Database repository.
DbName	Database table name.

Database	
Function	Description
DbRecs	The number of rows in the Database repository.
DbReload	Loads a resident table during Runtime
DbRound	Database round function for SQL databases.
DbSize	Returns the database table size.
ErrDatabaseName	Returns the name of the database on which the error has occurred
ErrDbmsCode	Returns the DBMS original error code
ErrDbmsMessage	Returns the DBMS original error message
ErrMagicName	Contains the eDeveloper literal of the error that occurred
ErrPosition	Refers to the internal position of the record on which the error occurred
ErrTableName	Returns the physical name of the table on which the error has occurred
InTrans	Evaluates if a transaction is currently open.
MTblGet	Retrieves the content of a memory table as a BLOB variable.
MTblSet	Creates records in a memory table where a BLOB variable is used as the table's content.
Rollback	Allows rollback of a transaction to a specified nesting level save point
TransMode	Provides information about the current active transaction.

Database	
Function	Description
ViewMod	Checks whether records were changed since the last save.

Date and Time	
Function	Description
AddDate	Performs calculation on date
AddTime	Performs calculations on a time variable.
BOM	Returns start date of month specified
BOY	Returns end date of year specified
CDOW	Name of the day (e.g. Sunday) from date
CMonth	Name of the month (e.g. January) from date
Date	System date
Day	Day of month (1-31) from date
DOW	Number of the day of the week (1-7) from date
DStr	Date-to-string conversion
DVal	String-to-date conversion
EOM	Returns date of end of month specified
EOY	Returns date of end of year specified
Hour	Returns the hour portion of the time value.
MDate	Returns the Magic date as input in logon screen
Minute	Returns the minute portion of the time value.
Month	Month of a date expression (1-12)

Date and Time

Function	Description
mTime	Retrieves the time value in milliseconds from midnight to the current time.
mTStr	Converts a time value in milliseconds to a specified Alpha string picture format.
NDOW	Converts the number of the day to the name of the day
NMonth	Converts the number of the month to the name of the month
Second	Returns the second portion of the time value.
Time	Returns the system time.
TStr	Converts the time value to an alpha string.
Year	Returns the year of a date (0000-2999)

Enterprise Server

Function Name	Description
CtxGetAllNames	Returns the names of all open contexts on the server engine where the function is evaluated.
CtxGetId	Retrieves the context identifier by a defined context name. You can define a name for a context identifier by using the CtxSetName function for a specific context.
CtxKill	Removes a context from the active context list.
CtxLstUse	Returns the number of seconds since the last activation of a context.

Enterprise Server

Function Name	Description
CtxNum	Returns the number of active contexts that are currently defined in an enterprise server.
CtxProg	Returns the public name of the top-level program of the context.
CtxSize	Returns the size of the context.
CtxStat	Returns the status of a context.
DiscSrvr	Disconnects the server
File2Req	Sends a file specification to the requester
GetParam	Gets parameters
RqCtxInf	Returns an information string of a given context identified by the context ID.
RqCtxTrm	Terminates a specified context.
RqExe	Requests a MRB to load a new copy
RqHTTPHeader	Sets the required HTTP Header information for the returned result of the HTTP result of a batch program.
RqLoad	Requester load
RqQueDel	Requester queue delete
RqQueLst	Requester queue list
RqQuePri	Requester queue priority
RqRtApp	Requester runtime application information
RqRtApps	Requester runtime applications
RqRtCtx	Returns the information of a given context entry of a given service or server name.
RqRtCtxs	Loads the information of all of the opened contexts of the enterprise server.

Enterprise Server

Function Name	Description
RqRtInf	Requester runtime information
RqRts	Requester runtime
RqRtTrm	Terminates all enterprise servers associated with a requester.
RqRtTrmEx	Terminates all enterprise servers associated with a requester by a graceful timeout.
RqStat	Request status.
RqTrmTimeout	Retrieves the remaining number of seconds before the server termination occurs.
SharedValGet	Retrieves a shared value according to its name. A shared value is a value stored in the memory of the eDeveloper process. Once created, this value can be retrieved by all active contexts and new contexts.
SharedValSet	Creates a shared value, which is a value stored in the memory of the eDeveloper process. Once created, this value can be retrieved by all active contexts and new contexts.
Text	Tests for the server background mode.
WSAttachmentAdd	Attaches a BLOB variable to a web service message.
WSAttachmentGet	Retrieves an attachment by web service.

Environment

Function	Description
----------	-------------

Environment

ANSI2OEM	Converts data from ANSI to OEM.
IniGet	Query Magic.ini file.
IniGetLn	INIT File Get Value.
OEM2ANSI	Converts data from OEM to ANSI
OSEnvGet	This function returns an alpha string containing the value of the variable. If the variable is not set, an empty string is returned.
OSEnvSet	This function sets the value of an operating system environment variable. The duration of this setting is until the eDeveloper process terminates.
ParamsPack	Packs a collection of global values that have been set by the SetParam function into a BLOB variable. The BLOB variable can be transmitted to another engine context or process to recreate the global values by using the ParamsUnPack function.
ParamsUnPack	Unpacks the global values from a BLOB variable created by using the ParamsPack function and sets them for the current engine context.
Pref	Returns the application prefix from the Application repository, including the path.
SetLang	Selects the active language.
SharedValPack	Packs a collection of shared values, set by the SharedValSet function, into a BLOB variable. The BLOB can be transmitted to another process to recreate the global values by using the SharedValUnpack function.

Environment

SharedValUnPack	Creates shared values, retrieved by the SharedValGet function, from a BLOB value created by using the SharedValPack function.
Sys	Returns the name of the application as it appears in the Name column of the Application repository.
Term	Returns the terminal number as specified in the Environment table.

Integration

Function	Description
BlobFromBase64	Decodes a BASE-64 BLOB variable to a regular eDeveloper BLOB.
BlobToBase64	Converts an eDeveloper BLOB to a BLOB variable encoded in the BASE-64 algorithm.
XMLBlobGet	Returns the value of an XML element or an XML attribute according to its element path.
CallDLL	Enables a dynamic call to a DLL
CallDLLF	Call to an external Fastcall function.
CallDLLS	Call to an external Stdcall function.
CallJS	Calls an external Java Script function that can be used for the browser task interface.
CallOBJ	Calls an external method of an object that can be used for a browser task interface.
CallProgURL	Calls a URL designated by an eDeveloper program that resides on an enterprise server.
CallURL	Calls a URL through the Evaluate operation in a browser task.

Integration

Cipher	Encrypts a buffer containing a BLOB.
ClipAdd	Adds a value and its picture to the clipboard for operating systems that support clipboard functionality.
ClipRead	Lets you display the contents copied to the clipboard in CF_Text format.
ClipWrite	The function places the buffer created by the ClipAdd function into the clipboard by using the CF_Text clipboard format.
ClientCertificateAdd	Lets you define a certificate that will be sent for subsequent Call Web Services and HTTP Post and Get calls.
ClientCertificateDiscard	Lets you remove a client-side digital certificate from the list of certificates. Client-side digital certificates verify the identity of the user for highly secure Web applications.
COMError	Retrieves information of the last error that occurred when eDeveloper interacted with a COM object.
COMHandleGet	Retrieves the handle of a loaded COM object. This handle can be stored as a numeric value.
COMHandleSet	Lets you refer to an object, previously loaded from a COM object field, using a handle number returned by the COMHandleGet function.

Integration

COMObjCreate	You can manually create an instance of a COM object based on a Select operation that defines the object's details. The ComObjCreate function can be executed only for an ActiveX or OLE variable that has the Stored Data control property set to Reference.
COMObjRelease	Releases a COM object loaded by the COMObjCreate function or called by the Call COM operation.
DDEBegin	Creates a session.
DDEEnd	Terminates a session.
DDEGet	Returns a string value from a DDE server.
DDEPoke	Transfers a string from eDeveloper to the DDE server.
DDERR	Retrieves the last error that occurred during an eDeveloper DDE conversation.
DDExec	Transfers a command string from eDeveloper to the DDE.
DeCipher	Converts an encrypted buffer to a buffer containing an Alpha string or a BLOB.
DragSetCrsr	Determines whether the cursor file is defined as Copy mode or as None mode.
DragSetData	Determines the data content and format for a control that is not defined for automatic data handling. This function also lets you assign the data content to a different data format.
DropFormat	Checks that the defined data format is supported for the Allow Dropping property.

Integration

DropGetData	Retrieves the data from the drag and drop operation by the defined format.
DropMouseX	Retrieves the Mouse Cursor X coordinate relative to the current form at the time the dragged data value is dropped on a form or control.
DropMouseY	Retrieves the Mouse Cursor Y coordinate relative to the current form at the time the data value is dropped onto a form or control.
HTTPGet	Retrieves the returned HTML result of an HTTP request as a BLOB object.
HTTPLastHeader	Retrieves the value of an HTTP header entry from the entry name. The function queries the HTTP header information received from the last HTTP retrieval from the HTTPGet or HTTPPost functions.
HTTPPost	Posts information via an HTTP message and returns an HTML/XML result of the HTTP request as a BLOB object.
LDAPError	Returns the last error message from the LDAP server. The function scope is per context.
LDAPGet	Retrieves the user information stored in a Lightweight Directory Access Protocol (LDAP) operating system directory.
MailBoxSet	Switches to another mailbox connected to the IMAP mail server.
MailConnect	Opens a connection to a mail server.
MailDisconnect	Closes a connection to an email server.

Integration

MailError	Translates a given mail error code that has been returned from one of the functions described above to a readable error message.
MailFileSave	Saves a message attachment to a file on a disk.
MailLastRC	Retrieves the most recent error message that occurred when using any of the mail functions.
MailMsgBCC	Retrieves the BCC string of the selected mail message.
MailMsgCC	Returns a comma-delimited string of all the CC addresses.
MailMsgDate	Retrieves the date and time information of the selected mail message.
MailMsgDel	Deletes a message from the server mailbox.
MailMsgFile	Returns the file name of the specific attachment of the message.
MailMsgFiles	Returns the number of attachments of the message.
MailMsgFrom	Returns the address from which the message was sent.
MailMsgHeader	Retrieves the header information of the selected mail message.
MailMsgId	Returns the email message identifier.
MailMsgReplyTo	Retrieves the <i>reply to</i> string of the selected mail message.
MailMsgSubj	Returns the subject string of the message.
MailMsgText	Returns the body text string of the message.

Integration

MailMsgTo	Returns a comma-delimited string of all the main addresses to which the message is sent.
MailSend	Used to send an email
UDF	Calls a user defined function.
WSAttachmentAdd	Attaches a BLOB variable to a web service message.
WSAttachmentGet	Retrieves an attachment received by a web service.
XMLBlobGet	Returns the value of an XML element or an XML attribute according to its element path.
XMLCnt	Returns the number of occurrences of an XML element or an XML attribute according to its path.
XMLDelete	Lets you delete an XML element or attribute according to its path. The function deletes the value and tags of the element or attribute.
XMLExist	Returns a True value if an XML element or XML attribute exists according to its element path.
XMLFind	Returns the index of an XML element that has a value equal to a specified value or if one of its attribute values is equal to the specified value.
XMLGet	Returns the value of an XML element or an XML attribute according to its element path.

Integration

XMLGetAlias	Retrieves the alias associated with a namespace Uniform Resource Identifier from the namespaces internal table. A namespace uniquely identifies an element type and attribute name. A URI is an alphanumeric character string identifying an Internet Resource.
XMLGetEncoding	Retrieves the encoding of an XML document.
XMLInsert	Lets you insert an XML element or attribute value in a specified path.
XMLModify	Lets you modify an XML element or attribute according to its path.
XMLSetEncoding	Sets the encoding of an XML document that was opened for Write access. The encoding affects both the encoding attribute in the document header, such as encoding=UTF 8, and the encoding of the actual XML document.
XMLSetNS	Defines the namespace-alias Uniform Resource Identifier (URI). A namespace uniquely identifies an element type and attribute name. A URI is an alphanumeric character string identifying an Internet Resource.
XMLStr	Converts valid XML data into an Alpha string.
XMLVal	Converts a string value into a valid XML string.

Interface	
Function	Description
CHeight	Returns the position on the y-axis that represents the height of a specified control or of the last control
CLeft	Returns the position on the x-axis relative to the window of a specified control or of the last parked control
CLeftMDI	Returns the position on the x-axis of a specified control or of the last parked control relative to the eDeveloper
ClickCX	Returns the X position of the last click, relative to a control
ClickCY	Returns the Y position of the last click, relative to a control
ClickWX	Returns the X position of the last click, relative to the window
ClickWY	Returns the Y position of the last click, relative to the window
CTop	Returns the position on the y-axis relative to the window of a specified control or of the last parked control.
CTopMDI	Returns the position on the y-axis of a specified control or of the last parked control relative to the eDeveloper MDI
CtrlGoto	Lets you park on a defined control.
CtrlHWND	Returns the Window handle of a control.
CtrlName	Returns the control name of the last clicked control
CurRow	Returns the number of the current parked row within a table control

Interface	
Function	Description
CWidth	Returns the position on the x-axis that represents the width of a specified control or of the last parked control
HitZOrdr	Returns Z-Order number of a control
LastPark	Returns the name of the control on which the user last parked in the specified area
MAXMagic	Maximizes the eDeveloper window.
MINMagic	Minimizes the eDeveloper window.
ResMagic	Restores the eDeveloper window to normal size.
SetCrsr	Set Cursor Shape.
WINBox	Returns window dimensions
WINHWND	Returns Windows window handle

I/O	
Function	Description
Blb2File	Saves a BLOB object to a file.
EOF	End-of-file signal in I/O file.
EOP	End-of-page signal in I/O file
File2Blb	Saves a file to a BLOB object.
File2OLE	Moves a file to an OLE variable.
FileDLG	Accesses a Windows Open File dialog.
FileListGet	Returns a list of the file names in a directory based on a set of file-name filters.
IOCopy	Copies file.

I/O

IOCurr	Returns the position of an IO file in the IO File table.
IODel	Delete disk file
IOExist	Check existence of a disk file
IORen	Rename file
IOSize	Size of a disk file
Line	Current line in output file
Page	Current page in output file

Java

Function	Description
EJBCreate	Obtains a new instance of an Enterprise Java Bean.
JCall	Calls an instance method.
JCallStatic	Calls a class method.
CodePage	Sets the code page that would be used when converting Java characters and strings to an eDeveloper Alpha type, and from an eDeveloper Alpha type to Java characters.
JCreate	Obtains a new instance of a Java class.
JException	Returns a pseudo-reference to the last exception of the current context.
JExceptionOccurred	Informs you that the last J* or EJB* function threw an exception.
JExceptionText	Returns a text image from the last exception and an optional backtrace of the stack. This function refers to the last exception thrown during the last j* or ejb* function.

Java

JExplore	Describes a class.
JGet	Retrieves the value of an instance variable.
JGetStatic	Queries a class variable.
JInstanceOf	Simulates the Java's operator instance.
JSet	Updates an instance variable.
JSetStatic	Updates a class variable.

Miscellaneous

Function	Description
CndRange	Sets a conditional range for the value of a variable.
EuroCnv	Returns the converted value from currency to currency based on the conversion values of the euro currency in the European Currency table.
EuroDel	Deletes an existing currency from the European Currency table.
EuroGet	Accesses the Base Currency value in Deployment mode.
EuroSet	Modifies the Base CURrency value in Deployment mode.
EuroUpd	Modifies the European Currency table.
EvalStr	Evaluates dynamic expressions that may be constructed at runtime.

Miscellaneous

EvalStrInfo	Checks a string representing an eDeveloper expression that can be evaluated using the EvalStr function and can retrieve information about the expression.
ExpCalc	Executes a function during the evaluation of an expression.
NULL	Sets a variable to a given value.
WinHelp	Opens a specified Help file and performs a selected command.

Numeric Values Manipulation

Function	Description
*	Multiplication
+	Addition
-	Subtraction
/	Division
ABS	Absolute value.
ACOS	Arc cosine
ASIN	Arc sine
ATAN	Arc tangent
ChkDgt	Check digit
CHR	Number ASCII character
COS	Cosine

Numeric Values Manipulation

EXP	Exponential
Fix	Extracts a portion of a number without rounding
HStr	Converts decimal to hexadecimal
LOG	Natural logarithm
MAX	Compares values and returns the largest one
MIN	Compares values and returns the smallest one
MOD	Returns the remainder of an integer division
MStr	Converts number to alpha (any format)
RAND	Random number generator
Range	Checks whether a number falls within a specified range
Round	Extracts a portion of a number with rounding
SIN	Sine
Str	Translates numeric value to an alpha string
TAN	Tangent

Security

Function Name	Description
GroupAdd	Assigns a user to a user group in the Security file from within an application.

Security

LMChkIn	Checks in a user instance of a specified feature license.
LMChkOut	Checks out a use instance of a specified feature license.
LMUVStr	Returns the vendor string of the user license that was checked out by a user.
LMVStr	Returns the vendor string of the license that is checked out by the eDeveloper engine.
Logon	Entry to a current application.
PPD	Programmable Protection Device Code.
RightAdd	Assigns a right to a user in the Security file from within an application.
Rights	Queries whether the user owns a specific right.
User	User information from the User ID repository.
UserAdd	Add a user record into the Security file from within an application.
UserDel	Lets the supervisor delete a user identification in a security file from within an eDeveloper application.

String

Function Name	Description
&	Use to concatenate alpha strings.
ANSI2OEM	Converts data from ANSI to OEM
ASC	ASCII value of a string character.
Astr	Applies a selected format to a string value.
CRC	Calculate redundancy check.

String	
Function Name	Description
Del	Delete characters from string
DStr	Date-to-string conversion
Fill	Repeat string
Flip	Invert string
HVal	Hexadecimal to decimal conversion
Ins	Insert string in another string
InStr	Search for first occurrence of string in another string
Left	Extract substring from left
Len	String length
Logical	Converts a visual representation to a logical representation.
Lower	Returns string in lowercase
LTrim	Remove leading blanks
MID	Extract substring
MIsTrans	Returns the translation of a string.
MTVal	Converts a time value in milliseconds from an Alpha string to a numeric value.
MVal	Alpha string to number conversion.
OEM2ANSI	Converts data from OEM to ANSI
Rep	Replaces substring within string
RepStr	Replaces all occurrences of a defined substring with another substring in a given source string.
Right	Extracts substring from right
RTrim	Removes trailing blanks
SoundX	Compares homonyms

String	
Function Name	Description
StrToken	Returns a token from a delimited string
StrTokenCnt	Returns the number of existing delimited tokens in a given string.
StrTokenIdx	Returns the token index in a delimited Alpha string.
Translate	Translates all logical names and nested logical names to their actual values. Secret names are not translated.
Trim	Removes leading and trailing blanks
TVal	Alpha to time conversion
Upper	Returns string in uppercase
Val	Converts an alpha character to a number
Visual	Converts a logical representation to a Visual representation
WebRef	Converts a text string to an input field on an HTML web page

Task	
Function	Description
CallProg	Calls up a program and passes arguments to it.
Counter	Iteration counter.
Delay	Freezes all activity for a second.
Flow	The current flow mode of a task.
FlwMtr	Appends a message to the Activity Message list.
GetLang	Gets the active language

Task

GetParam	Retrieves values passed as parameters to programs.
HandledCtrl	Returns the name of the control from which the current handler was executed.
Idle	Check system inactivity time.
InTrans	Evaluates if a transaction is currently open.
IsComponent	Checks if the executed program or handler is from a component or a host application.
IsFirstRecordCycle	The Record Main conversion from previous eDeveloper versions to eDeveloper Version 9.4 requires an indication of a conversion procedure when executing the Record Prefix in a task.
KbGet	Last entry capture.
KbPut	Key entry.
Level	Task execution level.
Lock	Returns an evaluated expression that locks a table row or task
LoopCounter	Returns the current count of the block loop cycle.
Menu	Menu path.
MMClear	Clears marked records.
MMCount	Provides the number of marked table rows.
MMCurr	Provides the current row of the number of marked rows in the counting process.

Task

MMStop	Stops the multi-mark handler.
MnuCheck	Displays or hides a checkmark in front of a menu entry.
MnuEnabl	Enables or disables a menu entry.
MnuName	This function sets the menu entry text of a selected menu.
MnuShow	Hides or shows a menu entry.
Prog	Task path.
ProglIdx	Returns the current index number of the program selected in the Program repository.
Rollback	Allows rollback of a transaction to a specified nesting level save point.
RunMode	Returns a numeric code corresponding to the runtime engine mode.
SetLang	Selects the active language.
SetParam	Sets parameters.
SharedValGet	Retrieves a shared value according to its name. A shared value is a value stored in the memory of the eDeveloper process. Once created, this value can be retrieved by all active contexts and new contexts.
SharedValSet	Creates a shared value, which is a value stored in the memory of the eDeveloper process. Once created, this value can be retrieved by all active contexts and new contexts.
Stat	Task operation.

Task

TDepth	Returns the current task depth.
THIS	Directs a variable-related function or a task generation function to the variable or task from which an event was triggered.
TransMode	Provides information about the current active transaction.
TreeLevel	Retrieves the current level of the selected node in the data tree.
TreeNodeGoto	Parks on a tree node that is identified by the tree node identifier.
TreeValue	Retrieves the node identification determined by the current level of the selected node.
Unlock	Returns an evaluated expression that unlocks a table row or task that is locked.
ViewMod	Checks whether records were changed since the last save.

Variables

Function	Description
EditGet	Retrieves the control value in the edit mode.
EditSet	Sets the edited value of the control that invoked the last handler.
IsDefault	Tests if the value of a variable is equal to its default value.
ISNULL	Checks for the existence of a NULL value in a variable.
VarAttr	Returns a column's attribute.
VarCurr	Variable identifier.

Variables

VarCurrN	Returns the current value of a variable according to the variable's name.
VarDbName	Queries a selected dataview to retrieve the physical definition of each variable.
VarIndex	Returns the index of a variable according to the variable's name.
VarInp	Identifies the last variable where the input has occurred.
VarMod	Variable modification check.
VarName	Provides a variable's origin and description.
VarPic	Returns a string value that represents the picture of the selected field.
VarPrev	Retrieves the original value of a variable, based on a dynamic value representing a variable index within the Variable list.
VarSet	Sets a variable to a given value

Alphabetical Directory of Functions

A LIKE A This function will determine whether or not a given character string matches a specified pattern. The function will return a logical value. A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must match the characters specified in the character string; wildcard characters, however, can be matched with arbitrary fragments of the character string. Wildcard characters include:

'*' - any string of zero or more characters.

'?' - any single character

It is possible to search for character strings that include one or more of the wildcard characters. To search for a wildcard character as a regular character, the '\ ' character must be added before the wildcard character.

Syntax: 'string' LIKE 'pattern' (A LIKE A)
Parameter: Parameter string
Returns: True if the character string matches the specified pattern, and False if it does not.
Note: When you perform string comparisons with LIKE, all characters in the string's pattern are significant including leading and trailing blank spaces.

ABS

Absolute Value

Returns the absolute value of a real number, without regard to sign.

Syntax: ABS(numeric)
Parameter: *numeric:* A number.
Returns: Absolute value of number.
Example: Where x=20 and y=30,
ABS(x-y)
returns 10

ACOS

Arc Cosine

Returns the arc cosine value of a number, in radians.

Syntax: ACOS (numeric)
Parameter: *numeric:* A number.
Returns: Number (radians).
Example: Where x=1 and y=0.5,
ACOS(x-y)
returns 1.04719755
See also: COS, SIN, ASIN, TAN, ATAN

AddDate

Performs calculation on a date variable

Constructs a date out of an input date and 3 values to be added to that *date: Years, months, and days*. The resulting date is always a valid date.

Syntax: AddDate (date,years,months,days)
Parameters: *date*: A date.
years: The number of years to add to *date*. May be zero.
months: The number of months to add to *date*. May be zero.
days: The number of days to add to *date*. May be zero.
Returns: Date value.
Example: AddDate('01/01/1992'Date,1,2,2)
returns 03/03/1993
See also: DVal

AddTime

Performs calculations on a time variable.
Constructs an input time and three values that can be added to that input time value - hours, minutes, or seconds. The resulting time is the original input plus the number of hours, minutes, or seconds added to it.

Syntax: AddTime (time,hours,minutes,seconds)
Parameters: *time*: The time value.
hours: The number of hours that can be added to the time input.
minutes: The number of minutes that can be added to the time input.
seconds: The number of seconds that can be added to the time input.
Returns: The Time value.
Example: AddTime('12:00:00Time1,2,3) returns 13:02:03
See also: TVal, AddDate

ANSI2OEM

Converts an ANSI character set to OEM.

Syntax: ANSI2OEM (string)
Parameter: *String*: An OEM character string.
Returns: Parameter *string* returns a translation to ANSI

ASC

ASCII Value of String Character
Evaluates the left most character of an alphanumeric string and returns its value in the ASCII character set.

Syntax: ASC('string')

Parameter: *string*: An alpha character or string of characters.

Returns: ASCII code.

Example: The expressions ASC('N') and ASC('New') each return the same result, 78

See also: CHR

ASIN

Arc Sine
Returns the trigonometric arc sine of a number, in radians.

Syntax: ASIN(number)

Parameter: *number*: A number.

Returns: Arc sine of a number (radians).

Example: Where x=1 and y=0.5,
ASIN(x-y)
returns 0.52359878

See also: SIN, COS, ACOS, TAN, ATAN

Astr

Applies a selected format to a string value.

Syntax: Astr(source string, format string)

Parameters: *source string* - The string value to be formatted.
format string - The string value that determines the format of the source string value.

The characters below are placeholders or modify a corresponding character:

- X - Any character
- H - Any character
- # - Any character
- U - Corresponding character is switched to uppercase
- L - Corresponding character is switched to lowercase

The A character, used for Auto Skip in an Edit field, is ignored.

Numbers define the repetition of the preceding format character. If a format character does not precede the

number, it will repeat the number of times the X format character appears.

For example: The format of 2@3U5 is the same as XX@@@UUUUU.

The backslash (\) character lets you define an operator as a fixed character.

For example: Astr('abcd','2@3A\U\5\A') returns 'ab@@@U5A'

If the total length of the character placeholders provided in the format string is smaller than the source string length, the remaining characters are truncated.

Returns: The modified source string value determined by the format string.

ATAN

Arc Tangent

Returns the arc tangent of a number in radians.

Syntax: ATAN (number)

Parameter: *number*: A number

Returns: Arc tangent of a number (radians)

Example: Where x=2 and y=1,
ATAN(x-y)
returns 0.78539816

See also: TAN, COS, ACOS, SIN, ASIN

Blb2File

Save BLOB to File

Blb2File saves a BLOB object to a file.

Syntax: Blb2File (BLOB variable, file name)

Parameters: *BLOB variable*: Variable.

file name: Alpha file name.

Returns: Logical True if succeeded.

Example: Blb2File(BE, 'C:\bfile.blb')
Moves a BLOB variable BE to a file named bfile.blb on the C drive.

Blob2Req Sends the data of a BLOB variable to the requester

Syntax: Blob2Req(*data*)

Parameters: The data as a BLOB value sent to the requester.

Returns: True if the data is a valid BLOB value that is not a Null and the engine is executed as an enterprise server. If the value is not a BLOB, is a Null, or the engine is not executed as an enterprise server, this function returns False.

Example: Blob2Req(B) sends the data of BLOB Variable B to the requester.

Note: Before you send data to the requester, you can define the HTTP Header information by using the RQHTTPHeader function.

See also: File2Req

BlobSize Returns the BLOB size in bytes.

Syntax: BlobSize(variable)

Parameters: *variable* - A BLOB value containing the selected BLOB.

Returns: The BLOB size in bytes. If the BLOB is a Null, the function returns 0.

BlobFromBase64 Decodes a BASE-64 BLOB variable to a regular eDeveloper BLOB.

Syntax: BlobFromBase64(BLOB)

Parameter: A BLOB value encoded in BASE-64.

Returns: A BLOB value containing the decoded data.

BlobToBase64 Converts an eDeveloper BLOB to a BLOB variable encoded in the BASE-64 algorithm.

Syntax: BlobToBase64(BLOB)

Parameter: A BLOB.

Returns: Returns a BLOB variable containing the binary value encoded in BASE-64.

BOM

Beginning of Month

Returns the date of the start of the month specified in the parameter.

Syntax: BOM(date)

Parameter: a *date* or a *date* expression.

Returns: Value of type Date.

Example: BOM ('05/10/93'Date)
returns '05/01/93'

See also: BOY, EOM, EOY, AddDate, CDOW, and all Date functions.

BOY

Beginning of Year

Returns the date of the start of the year specified in the parameter.

Syntax: BOY(date)

Parameter: *date*: A date or a date expression.

Returns: Value of type Date.

Example: BOY ('10/05/93'Date)
returns '01/01/93'

See also: BOM, EOM, EOY, AddDate, CDOW, and all Date functions.

BufGetAlpha

Converts a value stored in a certain position in a buffer to an Alpha string.

Syntax: BufGetAlpha(variable reference, position, storage, length, value as pointer)

Parameters: *variable reference* - The reference to the BLOB variable containing the buffer. For example, 'A'VAR
position - The numeric value defining the starting position of the conversion method.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page page 560.
length - A numeric value representing the storage length.
value as pointer - A logical value indicating if the Alpha value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the Alpha value as a pointer.

- Returns:** An Alpha string when the conversion succeeds. An empty string when the conversion fails.
- The function fails when the:
- Position parameter has a negative value.
 - Position and length parameters are not part of the BLOB's content.
 - Storage parameter does not contain a valid number.
 - Length parameter does not contain a valid value.

Note: The available storage types are displayed below in the Magic Storage Type table.

#	Attribute	Storage ID	Storage Name	Length
1	Alpha	1	String - non-Null terminated string	<32K
2	Alpha	2	LString - String with a short integer containing the string length	<32K
3	Alpha	3	ZString - Null terminated string	<32K
4	Numeric	1	Signed Integer	1,2,4
5	Numeric	2	Unsigned Integer	1,2,4
6	Numeric	3	IEEE Float	4,8
7	Numeric	4	Float MS-Basic	4,8
8	Numeric	5	Float Decimal	4,8
9	Numeric	6	Packed Decimal	
10	Numeric	7	Numeric	
11	Numeric	8	Numeric Character	
12	Numeric	10	C-ISAM Decimal	
13	Numeric	11	Extended Float	
14	Logical	1	Number containing 0,1	

#	Attribute	Storage ID	Storage Name	Length
15	Logical	2	Dbase containing T,F	
16	Date	1	Integer - days from 1/1/1	4
17	Date	2	Integer - days from 1/1/1901	4
18	Date	3	YYMD	4
19	Time	1	Integer	4
20	Time	2	HMSH	4
21	BLOB	1	4 bytes of storage length + the buffer (16 bytes)	
22	BLOB	2	The buffer (16 bytes)	

See Also: SetBufCnvParam

BufGetBit

Returns the value of a bit for a specified byte position in a BLOB buffer.

Syntax: BufGetBit(variable reference, position, bit number)

Parameters: *variable reference* - A BLOB reference containing a byte value. For example, 'A'VAR

position - A number defining the position of the byte in the BLOB variable.

bit number - A value from 1 to 8 representing a bit number in the byte.

Returns: True when the bit is on. False when the bit is off. If the bit number is not valid, the function returns Null.

Example: BufSetNum('A'VAR,1,3,2,1) sets the binary value of the first byte to 00000011 by using the default low-hi conversion parameter.

BufGetBit('A'VAR,1,1) returns False

BufGetBit('A'VAR,1,7) returns True

BufGetBit('A'VAR,1,8) returns True

BufGetBlob

Converts a value stored in a specified position in a buffer to an eDeveloper BLOB.

Syntax: BufGetBlob(variable reference, position, storage, length, value as pointer, vector cell storage)

Parameters: *variable reference* - The reference to the BLOB variable containing the buffer. For example, 'A'VAR
position - The numeric value defining the starting position of the conversion method.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page 560.
length - A numeric value representing the storage type length.
value as pointer - A logical value indicating if the Alpha value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the Alpha value as a pointer.
vector cell storage - If a Vector value is written to the buffer, you need to specify the cell's storage type. For more information about the Vector data attribute, see Chapter 3, Data Items and Chapter 3, Models.

Returns: A BLOB string when the conversion succeeds. An empty string when the conversion fails.
The function fails when the:

- Variable reference does not refer to a valid variable identification.
- Variable reference does not refer to the BLOB variable.
- The position parameter has a negative value.
- The position and length parameters are not part of the BLOB's content.
- The storage parameter does not contain a valid number.

See Also: SetBufCnvParam

BufGetDate Converts a value stored in a specified position in a buffer to a Magic Date value.

Syntax: BufGetDate(variable reference, position, storage)

Parameters: *variable reference* - The reference to the BLOB variable containing the buffer. For example, 'A'VAR
position - The numeric value defining the starting position of the conversion method.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page page 560.

Returns: A Date value when the conversion succeeds. 0 when the conversion fails.

The function fails when the:

- Position parameter has a negative value.
- Position and length parameters are not part of the BLOB's content.
- Storage parameter does not contain a valid number.
- Conversion errors occur.

See Also: SetBufCnvParam

BufGetLog Converts a value stored in a specified position in a buffer to an eDeveloper logical value.

Syntax: BufGetLog(variable reference, position, storage)

Parameters: *variable reference* - The reference to the BLOB variable containing the buffer. For example, 'A'VAR
position - The numeric value defining the starting position of the conversion method.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page page 560.

Returns: A logical value when the conversion succeeds. A Null when the conversion fails.

The function fails when the:

- Position parameter has a negative value.
- Position and length parameters are not part of the BLOB's content.
- Storage parameter does not contain a valid number.
- Value is not valid.

See Also: SetBufCnvParam

BufGetNum

Converts a value stored in a specified position in a buffer to an eDeveloper Numeric value.

Syntax: BufGetNum(variable reference, position, storage, length)

Parameters: *variable reference* - The reference to the BLOB variable containing the buffer. For example, 'A'VAR
position - The numeric value defining the starting position of the conversion method.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page page 560.
length - A numeric value representing the storage type length.

Returns: A numeric value when the conversion succeeds. 0 when the conversion fails.

The function fails when the:

- Position parameter has a negative value.
- Position and length parameters are not part of the BLOB's content.
- Storage parameter does not contain a valid number.
- Length parameter does not contain a valid value.
- Conversion causes an overflow.

See Also: SetBufCnvParam

BufGetTime Converts a value stored in a specified position in a buffer to an eDeveloper time value.

Syntax: BufGetTime(variable reference, position, storage)

Parameters: *variable reference* - The reference to the BLOB variable containing the buffer. For example, 'A'VAR
position - The numeric value defining the starting position of the conversion method.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page page 560.

Returns: A Time value when the conversion succeeds. 0 when the conversion fails.
The function fails when the:

- Position parameter has a negative value.
- Position and length parameters are not part of the BLOB's content.
- Storage parameter does not contain a valid number.
- Conversion errors occur.

See Also: SetBufCnvParam
Retrieves a variant value stored in a specified position in a buffer.

BufGetVariant Retrieves a variant value stored in a specified position in a buffer.

Syntax: BufGetVariant(variable reference, position, value as pointer)

Parameters: *variable reference* - The reference to the BLOB variable modified by the conversion operation. For example, 'A'VAR
position - The numeric value defining where the conversion result is placed in the BLOB value.
value as pointer - A logical value indicating if the variant value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the variant value as a pointer.

Returns: A variant when the function succeeds. A null value is returned when the function fails.

See Also: SetBufCnvParam

BufGetVector Converts a value stored in a specified position in a buffer to an eDeveloper vector value.

Syntax: BufGetVector (blob, position, vector attribute, no. of elements, value as pointer, cell storage, cell length)

Parameters: *blob* - A BLOB value that contains the value for vector.
position - The numeric value defining the starting position of the conversion method.
vector attribute - A number defining the vector attribute type. Valid vector attributes are displayed in the Vector Type table.

Value	Attribute
1	Alpha
2	Numeric
3	Logical
4	Date
5	Time
6	BLOB

no. of elements - A numeric value representing the length of the vector.

value as pointer - A logical value indicating if the Vector value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the Vector value as a pointer.

cell storage - The storage type of the vector cells that will be used when the external vector is read from the buffer. This parameter can be any of the appropriate storage types that relate to the cell attribute.

cell length - The storage length type of the vector cells that will be used when the eDeveloper vector is created from the external vector. This parameter can be any of the appropriate lengths that relate to the storage type.

For valid storage numbers and lengths, refer to the eDeveloper Storage Type table on page 560.

Returns: A Vector. Null is returned if the functions fails.

The function fails when the:

- Position is a negative value.
- Position and length are not part of the contents of the BLOB.
- The storage parameter does not contain a valid number.
- The length parameter does not contain a valid value.

See Also: SetBufCnvParam

BufSetAlpha Converts an Alpha, Memo, or RTF string variable to one of the supported storage types.

Syntax: BufSetAlpha(variable reference, position, value, storage, length, value as pointer)

Parameters: variable reference - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR

position - The numeric value defining where the conversion result will be placed in the BLOB value.

value - The Alpha value that will be converted.

storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the eDeveloper Storage Type table on page 560.

length - A numeric value representing the length of the Alpha value.

value as pointer - A logical value indicating if the Alpha value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the Alpha value as a pointer.

Returns: True when the conversion succeeds. False when the conversion fails.

The function fails when the:

- Variable reference does not refer to a valid variable identi-

fication.

- Variable reference does not refer to a BLOB variable.
- Position parameter is a negative value.
- Storage parameter does not contain a valid number.

Example: BufSetAlpha('A'VAR,1,'filler',1,6,'FALSE'LOG) inserts *filler* from the first byte to the sixth byte.

See Also: SetBufCnvParam

BufSetBit

Sets the value of a bit in a byte stored in the BLOB buffer.

Syntax: BufSetBit (variable reference, position, bit number, logical value)

Parameters: *variable reference* - The reference to a BLOB variable that is modified by the conversion operation. For example, 'A'VAR

position - A number defining the position of the byte in the BLOB variable.

bit number - A value from 1 to 8 representing the bit number in the byte.

logical value - A logical value to set the specified bit.

Returns: True when the bit number is set. False when the bit number is not valid.

BufSetBlob

Converts a BLOB variable into one of the supported storage types at a specified position in the buffer.

Syntax: BufSetBlob(variable reference, position, value, storage, value as pointer)

Parameters: *variable reference* - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR

position - The numeric value defining where the conversion result will be placed in the BLOB value.

value - The BLOB value that will be converted.

storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the eDeveloper Storage Type table on page page 560.

value as pointer - A logical value indicating if the Alpha value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the Alpha value as a pointer.

Returns: True when the conversion succeeds. False when the conversion fails.

The function fails when the:

- Variable reference does not refer to a valid variable identifier.
- Variable reference does not refer to a BLOB variable.
- Position parameter has a negative value.
- Storage parameter does not contain a valid number.

See Also: SetBufCnvParam

BufSetDate Converts a Magic date to a binary value in a specified position and storage type in the buffer.

Syntax: BufSetDate(variable reference, position, value, storage)

Parameters: *variable reference* - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR

position - The numeric value defining where the conversion result will be placed in the BLOB value.

value - The numeric value that will be converted.

storage - A numeric value representing a Magic valid storage type. For a list of storage types, see the Magic Storage Type table on page page 560.

Returns: True when the conversion succeeds. False when the conversion fails.

The function fails when the:

- Variable reference does not refer to a valid variable identification.

- Variable reference does not refer to a BLOB variable.
- Position parameter has a negative value.
- Storage parameter does not contain a valid number.

Example: BufSetDate ('A'VAR,1,Date(),1) converts a current date into BLOB variable A starting from the first byte.

See Also: SetBufCnvParam

BufSetLog

Converts a logical value into one of the supported storage types at a specified position in the buffer.

Syntax: BufSetLog(variable reference, position, value, storage)
variable reference - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR
position - The numeric value defining where the conversion result will be placed in the BLOB value.
value - The numeric value that will be converted.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page 560.

Returns: True when the conversion succeeds. False when the conversion fails.
 The function fails when the:

- Variable reference does not refer to a valid variable identifier.
- Variable reference does not refer to a BLOB variable.
- Position parameter has a negative value.
- Storage parameter does not contain a valid number.

See Also: SetBufCnvParam

BufSetNum

Converts an eDeveloper number into one of the supported storage types at a specified position in the BLOB buffer.

Syntax: BufSetNum(variable reference, position, value, storage, length)

Parameters: *variable reference* - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR
position - The numeric value defining where the conversion result will be placed in the BLOB value.
value - The numeric value that will be converted.
storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page 560.
length - A numeric value representing the storage length.

Returns: True when the conversion succeeds. False when the conversion fails.

The function fails when the:

- Variable reference does not refer to a valid variable identification.
- Variable reference does not refer to a BLOB variable.
- Position parameter has a negative value.
- Storage parameter does not contain a valid number.
- Length parameter does not contain a valid value.
- Conversion causes an overflow.

See Also: SetBufCnvParam

BufSetTime Converts an eDeveloper Time value into one of the supported storage types at a specified position in the buffer.

Syntax: BufSetTime(variable reference, position, value, storage)

Parameters: *variable reference* - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR
position - The numeric value defining where the conversion result will be placed in the BLOB value.
value - The time value that will be converted.

storage - A numeric value representing a valid storage type in eDeveloper. For a list of storage types, see the Magic Storage Type table on page 560.

Returns: True when the conversion succeeds. False when the conversion fails.

The function fails when the:

- Variable reference does not refer to a valid variable identifier.
- Variable reference does not refer to a BLOB variable.
- Position parameter has a negative value.
- Storage parameter does not contain a valid number.

See Also: SetBufCnvParam

BufSetVariant Inserts an variant value into a specified position in a buffer.

Syntax: BufSetVariant(variable reference, position, value, value as pointer)

Parameters: *variable reference* - The reference to the BLOB variable modified by the conversion operation. For example, 'A'VAR
position - The numeric value defining where the conversion result is placed in the BLOB value.

value - The variant vlaue to be written in the buffer.

value as pointer - A logical value indicating if the variant value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the variant value as a pointer.

Returns: True when the function succeeds. False is returned when the function fails.

See Also: SetBufCnvParam

BufSetVector Converts an eDeveloper Vector value into one of the supported storages at a specified position in the buffer.

Syntax: BufSetVector (*variable reference, position, value, cell storage, cell length, value as pointer*)

Parameters: *variable reference* - The reference to the BLOB variable that is modified by the conversion operation. For example, 'A'VAR

position - The numeric value defining where the conversion result will be placed in the BLOB value.

value - The vector value that will be converted.

cell storage - The storage type of the vector cells that will be used when the external vector is read from the buffer. This parameter can be any of the appropriate storage types that relate to the cell attribute.

cell length - The storage length type of the vector cells that will be used when the eDeveloper vector is created from the external vector. This parameter can be any of the appropriate lengths that relate to the storage type.

value as pointer - A logical value indicating if the Vector value should be written directly to the buffer or as a pointer to the data. If you enter True, eDeveloper writes the Vector value as a pointer.

Returns: True or False indicating the success or failure of the function.

The function fails when the:

- Variable reference does not refer to a valid variable identification.
- Variable reference does not refer to a BLOB variable.
- Position is a negative value.
- Storage parameter does not contain a valid number.

See Also: SetBufCnvParam

CallDLL

Call to DLL

Enables a dynamic and direct call to a DLL from within eDeveloper.

Syntax: CallDLL (modulename.functionname,argument type string,arg1,arg2,...)

Parameters: *modulename.functionname* - The module and function names from the DLL.

argument type string - A string in which each character represents the type of argument. The last character represents the type of the return value of the function. The Argument types are:

- 1 - Char
- 2 - Short
- 4 - Long
- F - Float
- 8 - Double
- D - Double pointer
- E - Float pointer
- L - Long pointer
- A - Null terminated string pointer
- V - Void pointer
- T - A pointer to a buffer
- O - Void

arg1,arg2,... - The function arguments from the DLL.

Returns: The return value of the function in the DLL.

Example: `CallDLL('mydll.lmath','1L44',Action,A,B)` where the C function is: `long lmath(char action, long *a, longb)` gives the return value of the `lmath` function in the DLL.
`CallDLL ('mydll.add_str','AAA',A,B)` where the C function is: `char*add_str(char*a,char*b)` gives the return value of the `add_str` function in the DLL.

CallDLLF

Call to an external Fastcall function.

Enables a dynamic and direct call to a DLL from within eDeveloper to an external Fastcall function.

Syntax: `CallDLLF (modulename.functionname,argument type string,arg1,arg2,...)`

Parameters: same as `CallDLL`

Returns: same as `CallDLL`

Example: same as `CallDLL`

CallDLLS

Call to STDCALL

Enables a dynamic and direct call to a DLL from within eDeveloper to an external Stdcall function.

Syntax: CallDLLS (modulename.functionname,argument type string,arg1,arg2,...)

Parameters: same as CallDLL

Returns: same as CallDLL

Example: same as CallDLLF

CallJS

Call Java Script

Allows you to call an external Java Script function that is embedded externally within the HTML template being used for a browser-task interface.

Syntax: CallJS (name, parameters)

Parameters: *name*- The name of the Java Script function.

parameters- A comma-delimited sequence of parameters that will be passed to the Java Script function

Returns: The return value of the Java Script function.

Example: Calling a Java Script function that has a single character and is multiplied by a given factor: CallJS ('my_function', 'A',3) - This function will return a string of 'AAA'.

Note: You can also call a function and pass parameters in a single string.

For example: CallJS('my_function("A",3)')

CallOBJ

Call Object

Allows you to call the external object of an object (Active -X) that may be embedded externally within the HTML template being used for a browser-task interface.

Syntax: CallOBJ (object, method, parameters)

Parameters: *object*: The identifier of the object as placed in the HTML template.

method: The name of the object's function.

parameters: A comma-delimited sequence of parameters that will be passed to the object's method.

Returns: The return value of the object's method.

Example: Calling a movie-clip player by passing it the name of the clip to play:

CallOBJ('my player','\clips\my_clips.mpg')

This function will return a TRUE value if the player succeeded in uploading and playing the clip, False if it failed.

CallProg

Allows you to call a program and pass arguments to it.

Syntax: CallProg (program number, argument-1...argument-n)

Parameters: program number, argument -1...argument-n

Returns: The value returned by the called program.

Examples: 'Your discount is' & Str (CallProg ('23'prog,'" A, FG, 'Preferred', 23),'2P0')
where program 23 returns a numeric value.

Note: NULL values cannot be passed as parameters of the CallProg function.

CallProgURL

Calls a URL designated by an eDeveloper program that resides on an enterprise server.

Syntax: CallProgURL(application name, program's public name, destination, argument1, argument2...)

Parameters: *application name* - A string representing the name of the application to be called.

program name - A string representing the program name to be executed.

destination - A string representing the name of the frame or window for the program display.

arguments - A series of arguments that must be passed to the called program.

Returns: True

Example: CallProgURL('My App','My Prog','"A,5) issues a URL that calls the specified program, My Prog, from the specified application, My App, and passes to it the required set of

arguments. The result of the program execution is displayed in the defined destination frame or window. For this example, the blank Destination string opens a new window.

CallURL

Calls a URL through the Evaluate operation in a browser task.

Syntax: CallURL(URL, Destination)

Parameters: *URL* - A string representing the URL to open.

Destination - A string representing the name of the frame or window that displays the URL.

Returns: True

Example: CallURL('http://www.magicsoftware.com','Magic') opens the Magic Web site in the frame named Magic.

Note: Defining the destination as an empty string opens the requested URL in a new window.

CASE

Switch Between Values

Switches between various values according to a controlling expression.

Eliminates the need for nested IF expressions.

Syntax: CASE(*controlling expression*, *case1*, *value 1*, *case 2*, *value2*,..., *default value*)

Parameters: *controlling expression*: The expression by which to switch between the various values.

case x: One of the possible values of the controlling expression. When the controlling expression equals the value of *case x*, *value x* is returned.

value x: The value to be returned when the controlling expression equals *case x*.

default value: The value that is returned when the *controlling expression* does not equal *case x*.

Returns: The value that is returned when the controlling expression equals the case value. A default value is returned if the controlling expression does not equal a case value.

Example: CASE(A,1,'Red',2,'Green',3,'Blue','Black')
When A=1, the function returns 'Red'.
When A=2, the function returns 'Green'.

When A=3, the function returns 'Blue'.
When A is other than 1,2,3..., the function returns 'Black'.

CDOW

Character Day of Week

Returns the name of the day (e.g., Sunday, Monday) from a *date* or a *date expression*.

Syntax: CDOW(date)

Parameter: *date*: A date or a date expression.

Returns: Day of week in character string.

Examples: CDOW('01/28/92'Date)
returns 'Tuesday'

Where x contains, in effect, the date 01/28/92 (Tuesday),
then CDOW(x+1)
returns 'Wednesday'

See also: Day, DOW, NDOW

Note: The use of Date following '01/28/92' identifies the string as a date literal string, and should not be confused with the Date() function.

You can control the length of the day name by manipulating the length of the alpha column that will hold it. For example, if you update a 3-character alpha column with the result of a CDOW operation, eDeveloper displays the first three letters of the day name (e.g., 'Mon', 'Tue').

CHeight

Control Height

Returns the position on the y-axis that represents the height of a specified control or the last parked control.

Syntax: CHeight (control name, generation)

Parameters: *control name*: The name of the control. If you specify a null string (''), the last parked control will be used.

generation: A number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, etc.

Returns: Position in units of measurement.

Note: . Only parkable controls set the values for the function. A parkable control is a control with data or a control that may return an action. The relevant controls are: Check Box, Combo Box, Edit, Image Control, List Box, OLE Control, Push Button, Radio Button, Rich Edit, Slider, and Tab control.

ChkDgt

Generates a check digit. A check digit is a digit added to a number, either at the end or the beginning, that validates the authenticity of the number. A simple algorithm is applied to the other digits of the number which yields the check digit. By running the algorithm and comparing the check digit you get from the algorithm with the check digit encoded with the number, you can verify that you have correctly read all of the digits and that they make a valid combination. Check digits are used for credit cards and identification numbers.

Syntax: ChkDgt(string,numeric)

Parameters: *string*: An alpha string representing the number for which the check digit is calculated. The number should be converted to alpha type before this function is applied.
numeric: A numeric value representing Modulus 10 or Modulus 11, simple algorithms used to validate the number on a credit card.

The following numeric values are:

0 - Modulus 10

1 - Modulus 11

Returns: A number

Examples: ChkDgt('6789',0) returns 2

ChkDgt(x,0), where x contains the alpha string '6789', returns 2

ChkDgt('6789',1) returns 1

See also: CRC

CHR

Number to ASCII Character Conversion

Converts a number to a corresponding character in the ASCII character set.

Syntax: `CHR(numeric)`

Parameter: *numeric*: A number.

Returns: Alpha.

Examples: `CHR(78)`
 returns 'N' (ASCII code 78)
 Where the variable X contains 77,
 `CHR(X+1)`
 returns 'N'

See also: `ASC`

Note: Use this function to display characters that have no keyboard equivalent in a form class>0 and not in a Windows display. For example, `CHR(26)` displays the right arrow character.

Cipher

Encrypts a buffer containing a BLOB.

Syntax: `Cipher(Cipher ID, Buffer, Key [, Mode, IV])`

Parameters: *Cipher ID* - A number representing the selected cryptographic algorithm.
Buffer - The BLOB that will be encrypted.
Key - A BLOB containing the encryption key.
Mode - An Alpha string containing the selected mode for the encryption method selected by the Cipher ID. If a mode is not specified for a method that requires a mode, the CBC is used as the default. See the Mode table below.
IV - A BLOB containing an initialization vector. This parameter is optional.

Returns: A BLOB containing the encrypted buffer.
 If the Cipher ID is not valid, or the key length does not match the Cipher ID, the function returns a Null value.

Note: The supported encryption methods and modes are:

Algorithmic Name	Cipher Code	Supported Modes and IV Length	Key Length	Symmetry
BLOWFISH	1	ECB - NA CBC - 8 CFB - 8 OFB - 8	Minimum: 1 Maximum: 56 Recommend: 16	Symmetric
CAST	2	ECB - NA CBC - 8 CFB - 8 OFB - 8	Minimum: 5 Maximum: 16 Recommend: 8	Symmetric
DES	3	ECB - NA CBC - 8 CFB - 8 OFB - 8	Number of Keys: 1 Supported: 8 Recommend: 8	Symmetric
IDEA	4	ECB - NA CFB - 8 OFB - 8	Minimum: 1 Maximum: 16 Recommend: 16	Symmetric
RC2	5	ECB - NA CBC - 8 CFB - 8 OFB - 8	Minimum: 5 Maximum: 16 Recommend: 8	Symmetric
RC4	6	Not Applicable	Minimum: 1 Maximum: NR Recommend: 16	Symmetric
RC5	7	ECB - NA CBC - 8 CFB - 8 OFB - 8	Minimum: 1 Supported: 255 Recommend: 16	Symmetric

Algorithmic Name	Cipher Code	Supported Modes and IV Length	Key Length	Symmetry
DES3	8	ECB3 - NA CBC3 - 8	Number of Keys: 2 Max: 16 or 24 Recommend: 24	Symmetric
RSA	9	Not Applicable	Minimum: 48 Maximum: 2048 Recommend: 128	Asymmetric

See also: DeCipher, EncryptionError

CLeft

Control Left

Returns the position on the x-axis relative to the window of a specified control or the last parked control.

Syntax: CLeft(control name, generation)

Parameters: *control name*: The name of the control. If you specify a null string (""), the last parked control will be used.

generation: A number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, etc.

Returns: Position in units of measurement

Note: Only parkable controls set the values for the function. A parkable control is a control with data or a control that may return an action.

The relevant controls are: Check Box, Combo Box, Edit, Image control, List Box, OLE control, Push Button, Radio Button, Rich Edit, Slider, and Tab control.

CLeftMDI

Control Left MDI

Returns the position of a specified control or the last parked control on the x-axis relative to the eDeveloper MDI.

Syntax: CLeftMDI(control name, generation)

Parameter: *control name*: the name of the control. If you specify a null string ("), the last parked control will be used.

generation: a number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, etc.

Returns: Position in units of measurement

Note: Only parkable controls set the values for the function. A parkable control is a control with data or a control that may return an action.

The relevant controls are: Check Box, Combo Box, Edit, Image control, List Box, OLE control, Push Button, Radio Button, Rich Edit, Slider, and Tab Control.

ClickCX

Click Control X

Gives the X location of the last click, relative to the control within which the click occurred. The location is always expressed in the unit of measurement that is in effect.

Syntax: ClickCX ()

Parameter: None.

Returns: Numeric value

Example: ClickCX ()

Returns the X coordination of where the user clicks the mouse relative to a control.

Note: When checked after clicking on a tab control, the ClickCX function returns the layer number of the clicked tab.

See also: ClickCY, ClickWX, ClickWY

ClickCY

Click Control Y

Gives the Y location of the last click, relative to the control within which the click occurred. The location is always expressed in the unit of measurement that is in effect.

Syntax: ClickCY ()
Parameter: None
Returns: Numeric value.
Example: ClickCY ()
Returns the Y coordinate of where the user clicks the mouse relative to a control.
Note: When checked after clicking on a tab control, the Click CY() function returns the layer number of the clicked tab.
See also: ClickCX, ClickWX, ClickWY

ClickWX

Click Window X
Gives the X location of the last click, relative to the window. The location is always expressed in the unit of measurement in effect.
Syntax: ClickWX()
Parameter: None.
Returns: Numeric value
Example: ClickWX ()
Returns the X coordinate of where the user clicks the mouse relative to a window.
See also: ClickCX, ClickCY, ClickCY

ClickWY

ClickWY()
Gives the Y location of the last click, relative to the window. The location is always expressed in the unit of measurement in effect.
Syntax: ClickWY()
Parameters: None
Returns: numeric value
Example: ClickWY()
Returns the Y coordinate of where the user clicks the mouse relative to a window.
See also: ClickCX, ClickCY, ClickWX

ClientCertificateAdd

Lets you define a certificate that will be sent for subsequent Call Web Services and HTTP Post and Get calls.

Syntax: ClientCertificateAdd (*PKCS12-certificate*, *password* [, *SSLCACertificates*])

Parameters: *Certificate URL* - The certificate's URL.

PKCS12-Certificate - Public Key Cryptography Standard #12 is an industry format used for the transport, backup, and restoration of a certificate and its associated public and private key. This parameter must be an alphanumeric value.

Password - The Password can be a Null value when the certificate is exported without a password.

SSLCACertificates - An intermediate Certificate Authority, for example Certificates Service of Microsoft (CertSrv). For the target server to process the Call Web Service or HTTP Post or Get functions, the client can pass the SSLCACertificates of the intermediate Certificate Authorities. Certificate authorities are separated by the semi-colon character (;). This parameter is optional.

Returns: True if a valid and authorized pKCS12 certificate was added or False when the:

- Same certificate was added more than once.
- Certificate does not exist.
- Certificate is not in PKCS12 format.
- Password is incorrect.

Example: ClientCertificateAdd (my.pfx, %MY_PASS%, %ROOT_CERTS%) where:

- My.pfx is the PKCS12 certificate.
- %MY_PASS% is the logical name for the user's password,
- %ROOT_CERTS% is the logical name for the intermediate Certificate Authority. This parameter is optional.

Note: All client certificates will remain active for the current context, sending subsequent calls, until the certificate is removed from the current context by using the

ClientCertificateDiscard function. You can call the ClientCertificateAdd function several times to add multiple client certificates. You cannot add the same client certificate more than once. Certificates can be requested with an option to make the private key non-exportable, which means that PKCS12 certificates can not be imported to eDeveloper.

See also: HTTPGet, HTTPPost functions, and the Call Web Service operation.

ClientCertificateDiscard

Lets you remove a client-side digital certificate from the list of certificates. Client-side digital certificates verify the identity of the user for highly secure Web applications.

Syntax: ClientCertificateDiscard (*PKCS12 certificate*)

Parameters: *PKCS12 certificate* - The certificate's URL. This parameter must be an alphanumeric value.

Returns: True when the certificate is discarded or False when the certificate is not discarded. The certificate is discarded only when the URL matches the certificate name previously added to the context by using the ClientCertificateAdd function or the Client Certificate application property.

Example: ClientCertificateDiscard (my.pfx) where my.pfx is the discarded client-based digital certificate.

Note: The certificate specified in the environment settings will be reissued when switching from runtime to toolkit, even when the certificate was discarded during runtime by using this function.

ClipAdd

Adds a value and its picture to the clipboard for operating systems that support clipboard functionality.

Syntax: ClipAdd (Value, Picture, [Value, Picture])

Parameters: This function receives a variable number of parameter pairs. Each pair contains a value and its picture that are used when placing the value on the clipboard.

Value - Any legal eDeveloper value except a BLOB storage type.

Picture - The corresponding picture for the value.

Returns: Boolean - For operating systems that support clipboard functionality, the value returned is True. For operating systems that do not support clipboard functionality, the value return is False.

Note: This function places the values into a buffer for the clipboard. Between the values a Tab character is inserted, and a new line is placed at the end of the string.

ClipRead Lets you display the contents copied to the clipboard in CF_Text format.

Syntax: ClipRead()

Parameters: None

Returns: ClipRead returns a BLOB value containing the contents of the clipboard in CF_Text format. If the operation cannot be performed (for example, an empty clipboard, incompatible formats, or an incompatible operating system), the function returns a NULL value.

ClipWrite The function places the buffer created by the ClipAdd function into the clipboard by using the CF_Text clipboard format.

Syntax: ClipWrite()

Parameters: None

Returns: If the buffer is successfully placed on the clipboard, the function returns a value of True.

ClrCache Clear Cache
Clears the data cache of the current task, which enables eDeveloper to read records directly from the database. This function can be used only in online tasks.

Syntax: ClrCache()

Parameter: None.

Returns: True/False

CMonth

Character Month

Returns the name of the month (e.g., January, February) from a *date* or a *date expression*.

Syntax: CMonth(date)

Parameter: *date*: A date or a date expression.

Returns: Alpha string

Example: CMonth('01/28/97'Date)
returns 'January'

If X contains, in effect, the date 01/28/97, then
CMonth(X+30)
returns 'February'

See also: Month, NMonth

Note: The use of Date following '01/28/92' identifies the string as a date literal string, and should not be confused with the Date() function.

You can control the length of the month name by manipulating the length of the alpha column that will hold it. For example, if you update a 3- character column with the results of a CMonth operation eDeveloper will display the first three letters of the month name (e.g., 'Jan', 'Feb').

CndRange

Allows for a conditional range for the value of a variable.

Syntax: . CndRange (condition, value)

Parameters: *condition* - Boolean expression to be evaluated during runtime.

value - to return if the condition is True.

Returns: Any value

Example: CndRange (`TRUE'L, 10)
The function will return the value 10
CndRange (`FALSE'L, 20)
The function will not execute.

CodePage Sets the code page to convert incoming and outgoing data to eDeveloper.

When eDeveloper receives data, it is converted from the specified code page to eDeveloper's internal representation. When eDeveloper sends data, it is converted from eDeveloper's internal representation to the specified code page. The code page is effective immediately for the current context only, until the next code-page call made by the current context. eDeveloper's default code page is the default code page of the operating system. This function is relevant for Java functions, XML functions, and web services.

Syntax: CodePage(*code page*)

Parameter: *code page* - The numeric value that represents the code page to be used. A zero value resets the code page back to the default of the operating system.

Returns: A logical value. If the function is successful in locating and setting the code page, a 'TRUE'LOG value is returned.

Example: CodePage(1252) sets the code page to Windows ANSI

Note: The Code Page Options table is located on 1107.

COMError Retrieves information of the last error that occurred when eDeveloper interacted with a COM object.

Syntax: COMError(*info type*)

Parameters: *info type* - The numeric value that specifies the information to be retrieved. The numeric values are:

- 1 - The error description.
- 2 - The COM object in which the error occurred.
- 3 - The index number in which the error occurred.
- 4 - The help file that describes the error.
- 5 - The context number of the error help topic.
- 0 - The HRESULT value that represents the actual error number. The HRESULT is retrieved in hexadecimal representation.

Returns: An Alpha string as defined by the *info type* parameter. Numeric values, such as an argument index, are returned as

Alpha strings. A blank is returned when there is no error or when the info type parameter is set to an invalid value.

COMHandleGetRetrieves the handle of a loaded COM object. This handle can be stored as a numeric value.

Syntax: COMHandleGet(object)

Parameters: *object* - A BLOB value representing a COM object field.

Returns: A positive numeric value representing the object's handle. A negative number is returned when the function fails.

The negative numbers that the function can return are:

-1 - The attribute is not an ActiveX or OLE variable.

-3 - The object is not loaded by the defined variable.

Note: Handling objects using the COMHandleGet and COMHandleSet functions should be done with caution. Accidentally assigning wrong handles to wrong objects may result in unexpected behavior.

COMHandleSetLets you refer to an object, previously loaded from a COM object field, using a handle number returned by the COMHandleGet function.

Syntax: COMHandleSet(variable reference, handle)

Parameters: *variable reference* - The value representing a variable index in the Variable list.

handle - A numeric value of a handle that was retrieved using the COMHandleGet function.

Returns: 0 when the function is successful. A negative number is returned when the function fails.

The negative numbers that the function can return are:

-1 - The attribute is not an ActiveX or OLE variable.

-3 - The object is not loaded by the defined variable.

-4 - The handle provided by the variable is not a valid handle.

Note: Handling objects using the COMHandleGet and COMHandleSet functions should be done with caution.

Accidentally assigning wrong handles to wrong objects may result in unexpected behavior.

COMObjCreate You can manually create an instance of a COM object based on a Select operation that defines the object's details. The ComObjCreate function can be executed only for an ActiveX or OLE variable that has the Stored Data control property set to Reference.

Syntax: COMObjCreate(variable)

Parameters: *variable* - A value representing a variable index within the Variable list.

Returns: The handle number of the loaded object. A negative number is returned when the function fails.

The negative numbers that the function can return are:

-1 - The attribute is not an Active-X or OLE variable.

-3 - The object has already been loaded by the defined variable.

-4 - Undetermined failure.

Note: Any COM object loaded manually must be released manually. The eDeveloper engine does not automatically release objects that were loaded by this function. Use the COMObjRelease function to release manually loaded objects.

COMObjRelease

Releases a COM object loaded by the COMObjCreate function or through automatic instantiation.

Syntax: COMObjRelease(variable)

Parameters: *variable* - The value representing a variable index within the Variable list.

Returns: 0 when the function is successful. A negative number is returned when the function fails.

The negative numbers that the function can return are:

-1 -The attribute is not an ActiveX or OLE variable.

-3 - The object is not loaded by the defined variable.

COS

Cosine

Returns the cosine of an angle, where the angle is expressed in radians.

Syntax: COS(numeric)

Parameter: *numeric*: A number that represents the angle in radians.

Returns: Number

Example: COS(0.7854)
returns 0.70711

See also: ACOS, SIN, ASIN, TAN, ATAN

Counter

Batch Iteration Counter

Counts the number of iterations at a record level performed by a Batch task.

Syntax: Counter(generation)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

Returns: Number

Example: Placed as an End Task condition with End Check=Yes, the expression Counter(0)=5 ends the current task (task 0) after five iterations on the record level.

CRC

Calculate Redundancy Check

Performs a cyclic redundancy check value on an alpha string.

Syntax: CRC(string,numeric)

Parameters: *string*: An alpha string to which the CRC is applied.

numeric: A number that represents the CRC algorithm. In this version of eDeveloper use 0 to apply CRC-16.

Returns: 2-byte string containing the CRC value.

Examples: CRC is used primarily for verification of a data stream. Since the function's algorithm uses all the elements of the input

string for the calculation, the result can serve as a reliable measure for insuring that the string isn't altered.

A common application for CRC is communication between two computers. In order to verify that the data transmitted has actually arrived intact, calculate a CRC value from the data prior to transmission. Then append the CRC word (2 bytes) to the information, and transmit. On the receiving side, strip off the CRC information, and calculate a CRC value of the remaining data. If the CRC calculated from the received data and the CRC received with the data match, then there were no faults detected during transmission. If the two CRC values do not match, then the data probably was corrupted during transmission.

Send `ABC&CRC('ABC',0)`

Call it *string*

At the receiving end, ask whether
`CRC(Left(string,3),0)=Right(string,2)`

See also: ChkDgt

CTop

Control Top

Returns the position on the y-axis relative to the window of a specified control or the last parked control.

Syntax: CTop(control name, generation)

Parameter: *control name*: the name of the control. If you specify a null string ("), the last parked control will be used.

generation: a number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, etc.

Returns: Position in units of measurement

Note: Only parkable controls set the values for the function. A parkable control is a control with data or a control that may return an action.

The relevant controls are: Check Box, Combo Box, Edit, Image, List Box, OLE, Push Button, Radio Button, Rich Edit, Slider, and Tab controls.

CTopMDI

Control Top MDI

Returns the position of a specified control or the last parked control on the y-axis relative to the eDeveloper MDI.

Syntax: CTopMDI(control name, generation)

Parameter: *control name*: the name of the control. If you specify a null string ("), the last parked control will be used.

generation: a number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, etc.

Returns: Position in units of measurement

Note: Only parkable controls set the values for the function. A parkable control is a control with data or a control that may return an action.

The relevant controls are: Check Box, Combo Box, Edit, Image, List Box, OLE, Push Button, Radio Button, Rich Edit, Slider, and Tab controls.

CtrlGoto

This function lets you park on a defined control.

Syntax: CtrlGoto('control name', row number, generation)

Parameters: *control name*: A string value that represents the name of the control on which you wish to park.

row number: A numeric value that represents the number of the row in a table control where the control is located. The Zero value will be considered as the current row. This parameter is ignored if the control only appears once in table.

generation: A number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, and so on.

Returns: The function returns a True value if it is successful in finding the control by name and row in the window of the given task. The function returns a False value if the selected control name is not found, the control is invisible, the specified row does not exist, the referenced task has no window, or the referenced task does not exist.

Example: CtrlGoto('Customer_Name',3,0) switches the focus to the control of *Customer_Name* in row 3 of the table in the window of the current task.

Note: The focus that switches to the control occurs when the application becomes idle after the evaluation of the function.

CtrlHWND

Window Handle for Controls

Provides the window handle of a control. You can use this handle within a user-defined function in an external DLL to determine the look and behavior of the control or to get additional information about the control styles or content.

You can use the function with these controls: Radio Button, List Box, OLE, Push Button, Combo Box, Slider, RTF Edit, RTF Text, and Check Box.

Syntax: CtrlHWND (control name)

Parameter: *control name* as specified in the Form editor

Returns: The window handle as a long numeric value of 4 bytes. This value can be passed by value to either a User-defined Function (UDF) or Procedure (UDP).

Example: CtrlHWND('Olecontrol') where Olecontrol is the name of a control on the Form, and returns the control's window handle.

CtrlName

Control Name

Returns the name of the last clicked control provided in the Control Name property.

Syntax: CtrlName()

Parameter: None.

Returns: The name of the last clicked control.

CtxClose

Gracefully removes a context from the Active Context list. When closing an active context, the engine properly closes all running programs, and clears the context from the Active Context list.

Syntax: CtxClose (*context entry or identifier*)

Parameter: *context entry or identifier* - A number representing the context. This number can be either the context entry number in the current context list or the actual context identifier.

Returns: True when the specified context is removed. If the context is not found, the function fails and returns False.

Note: Only pending contexts can be removed.

The RqCTxTrm function behaves like the CtxClose function. Before evaluating the CtxClose function, you should load the contexts list by evaluating the CtxNum function.

See also: CtxKill

CtxGetAllNames

Returns the names of all open contexts on the server engine where the function is evaluated.

Syntax: CtxGetAllNames()

Parameters: None

Returns: A string vector containing all the context names. For contexts that were not explicitly set with a context name, the context identification number is returned.

Note: The function returns an empty vector when executed on a non-server engine.

CtxGetId

Retrieves the context identifier by a defined context name. You can define a name for a context identifier by using the CtxSetName function for a specific context.

Syntax: CtxGetId(*context name*)

Parameters: context name - A string value representing the context name.

Returns: A numeric value of a context identifier corresponding to a specified name.

Example: CtxGetId('Dispatcher') returns the numeric identifier for a context named Dispatcher.

Note: If the context name parameter is empty, the returned context identifier is for the current context. If the specified context name is not found, a value of 0 is returned.

CtxKill

Abruptly removes a context from the Active Context list. The purpose of this function is to end active contexts waiting for the completion of certain procedure like Block loops, Exit operations, or calling DLLs.

Syntax: CtxKill (context *entry* or *identifier*)

Parameters: *context entry or identifier* - A number representing the context. This number can be either the context entry number in the current context list or the actual context identifier.

Returns: True when the specified context is abruptly removed. If the context is not found or your trying to remove the current context, the function fails and returns False.

Note: A running context can be removed at any time regardless of its current activity, including synchronous external operations such as the Call UDP operation.

The CtxKill function abruptly terminates all running threads of the specified context. Terminating a running thread can result in the improper release of resources, such as database connections and Instantiated Java or COM objects, which is similar to abruptly terminating the engine process.

When importing applications from previous versions of eDeveloper, the CtxKill function will be replaced with the current CtxKill function and not with CtxClose.

Before evaluating the CtxKill function, you should load the contexts list by evaluating the CtxNum function.

See also: CtxClose

CtxLstUse

Context Last Use Time

Returns the number of seconds since the last activation of a context.

Syntax: CtxLstUse(*context entry* or *identifier*)

Parameters: *context entry or identifier* - A number representing the context. This number can be either the context entry

number in the current context list or the actual context identifier.

- Returns:** Active returns 0.
Pending returns the number of seconds since the last activation.
Terminated returns -1.
Invalid returns -2.
- Note:** You must first evaluate the CtxNum function before using the CtxLstUse function.

CtxNum

Context Number

Returns the number of active contexts that are defined in an enterprise server.

- Syntax:** CtxNum()
- Parameters:** None
- Returns:** A numeric value indicating the number of active contexts in an enterprise server.
- Note:** You must first evaluate this function before using the CtxLstUse, CtxProg, CtxSize, and CtxStat functions.

CtxProg

Context Program

Returns the Public Name of the top-level program of the context.

- Syntax:** CtxProg(*context entry or identifier*)
- Parameters:** *context entry or identifier* - A number representing the context. This number can be either the context entry number in the current context list or the actual context identifier.
- Returns:** The Public Name of the top-level program of the context.
- Note:** You must first evaluate the CtxNum function before using the CtxProg function.

CtxSetName

The function lets you set the name of the current context. The name is used to post a message to the context.

Syntax: CtxSetName (name)
Parameters: *name* - A string defining the context name.
Returns: True when the operation is successful.
The function fails when:

- A null value was passed.
- An empty string was passed.
- Another context has the same name.

Note: The name length cannot be more than 128 characters.

CtxSize

Context Size

Returns the size of the context.

Syntax: CtxSize(*context entry or identifier*)
Parameters: *context entry or identifier* - A number representing the context. This number can be either the context entry number in the current context list or the actual context identifier.
Returns: The number of the program. For terminated or invalid contexts 0 will be returned.
Note: You must first evaluate the CtxNum function before using the CtxSize function.

CtxStat

Context Status

Returns the status of a context in the context table.

Syntax: CtxStat(*context entry or identifier*)
Parameter: *context entry or identifier* - A number representing the context. This number can be either the context entry number in the current context list or the actual context identifier.
Returns: A string, indicating the status of the context. Possible values are:
E for Executing when the current entry is currently executing.

P for Pending when the current entry is waiting for an event.
T for Terminated when the current entry has terminated.
I for Invalid when the entry number is invalid.

Note: You must first evaluate the CtxNum function before using the CtxStat function.

CurROW

Returns the number of the current parked row inside a Table control.

Syntax: CurRow(generation)

Parameters: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, and so on.

Returns: If a Table control exists in the task specified by the generation, the function will return the number of the parked row in that table.

If no table exists, the function returns the value of 1.

If the specified generation does not exist, the function returns the value of 0.

If the specified generation is a batch task, the function returns the value of 0.

If there is no table, the function returns zero.

Example: CurRow(0) returns the number of the parked row of the table control in the current task.

CurrPosition Current Position

Returns the internal position of the current record of the Main Table of the current task.

Syntax: CurrPosition(generation)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

Returns: BLOB expression

CWidth

Control Width

Returns the position on the x-axis that represents the width of a specified control or the last parked control.

Syntax: CWidth(control name, generation)

Parameters: *control name*: the name of the control. If you specify a null string ("), the last parked control will be used.

generation: a number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 the immediate ancestor, etc.

Returns: Position in units of measurement.

Date

System Date

Returns the system date.

Syntax: Date()

Parameter: None

Returns: Date

Examples: If the system date is 01/28/97,
Date()
returns 28/01/97
Date()+5
returns 02/02/97

See also: MDate, DStr, DVal

Note: eDeveloper automatically converts the date to the date format set in Environment (default), or to the picture of the variable that holds it. If the system date is set to the European format, you can display or print the system date in American format by changing the relevant parameter in Environment, or by defining a suitable picture for the variable that holds the date.

Day

Day of Month

Returns the day portion of a date, i.e., a number 1-31.

Syntax: Day(date)

Parameter: *date*: A date or a date expression.

Returns: Number

Example: Day('01/28/96'Date)
returns 28

The following expression displays the word "Overdue" if the system date is later than the 15th of the month:
IF(Day(Date())>15,'Overdue','')

See also: CDOW, DOW, NDOW

DbCache

Database Cache

Monitors the hit ratio, and returns an integer that specifies the cache hit ratio. The hit ratio is defined as the ratio between the number of times a record was found in a cache divided by the total number of times the cache was searched. The cache efficiency is determined by the hit ratio.

Syntax: DbCache (number,generation)

Parameters: *number*: A number that represents the table's sequence number in the Table repository. This parameter is mandatory.

generation: A number that represents the execution level of the task.

0 = the current task, 1 = parent task, etc.

Returns: The function will return an integer between 0-100 that specifies the hit ratio.

Example: DbCache ('1' FILE, 0) will return the hit ratio for the first table defined in the Table repository in the current task.
Note: the FILE literal can be used.

For more information, see the section on The eDeveloper Cache in Chapter 20, Utilities.

DbCopy

Database Copy

Allows you to copy an existing data file and its definition and data to a new file. DbCopy supports both ISAM and SQL files.

Syntax: DbCopy (*number, source file specification, destination file specification*)

Parameters: *number*: A number that represents the file's sequence in the Table repository.

source file specification: An optional string that represents the file specification name.

destination file specification: A string that represents the destination file specification name.

Returns: Logical indicating success or failure.

The function returns False in the following cases:

- The source file is currently open
- The source file is a resident file
- The source file does not exist - displayed error: "Table does not exist: source file specification"
- The destination file already exists - displayed error: "Target table exists. Create failed for table: destination file specification"
- The source file is a view - displayed error: "Can not drop/copy a view, view: source file specification"
- Other errors - displayed error: "Failed to copy, table: source file specification"

Notes: When using eDeveloper for iSeries, you can use the DbCopy functions in SQL gateway by entering the database name in the Owner table property. From the DBMS repository, you must enter the **NAMING= *SQL** keyword in as a parameter for AS400. For more information, see the Defining DBMS Parameters in the *iSeries Guide*.

DbDel

Database Table Deletion

Deletes an eDeveloper database that appears in the Table repository, and returns a logical True or False indicating success or failure of the operation. Removes all rows from a table that is residing as a parameter. Subsequent opening of this table reloads the rows from its database.

Syntax: DbDel(number,tablespec)

Parameters: *number*: A number that represents the table's sequence number in the Table repository. This parameter is mandatory.

tablespec: This parameter is used as an alternative to number. If not needed, enter ''. If needed, enter an alpha string that represents the table specification. The string may

contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: Boolean (True, False)

Examples: DbDel(1,'') or
DbDel('1'FILE,'')
attempts to delete the first table of the Table repository and returns True if the operation is successful.
DbDel (1,'FSTFIL.DAT')
attempts to delete the first table of the Table repository, using the OS name 'FSTFIL' in the current directory.

Note: Using the literal FILE will insure that the correct table will be deleted even if the table's position in the Table repository has changed. The table sequence number and path behave exactly as defined in the Table repository. Refer to Chapter 4, Tables.

See also: DbRecs, DbExist, DbSize

The DbDel function, when given a table that is resident as a parameter, will remove all rows from the resident table. Subsequent opening of this table will reload them from the database.

DbDiscnt

Database disconnect

Receives a database as a parameter and disconnects the current connection of that database. The DbDiscnt function is relevant only for SQL databases.

Disconnecting is allowed only if there is no cursor open on the database that you are trying to disconnect from. If there is a cursor open and the user issues a DbDiscnt command, an error message will be displayed.

DbDiscnt allows the developer to disconnect the current database connection in order to disconnect a user from a database connection no longer needed in the application, or to disconnect a user from a database connection in order to connect another user to the same database connection.

Syntax: DbDiscnt(string)

Parameter: *string* - An alpha string or alpha string expression containing the database name.

Returns: Boolean (True, False)
Returns True when the disconnect operation has succeeded, and False when the disconnect operation has not succeeded or when there is no connection to disconnect.

Example: Use DbDiscont('MSSQL'), where MSSQL is the database name defined in the Database repository.

DbERR

Database Error Message

Returns the first database error message that causes an internal processing error or chain of errors. This function enables the retrieval and display of a database error code and message text for display to end-users. Invoking the DbERR function also clears the error message.

Syntax: DbERR(string)

Parameter. *string*: An alpha string or alpha string expression containing the name of the database.

Returns: For SQL: A string containing the error message.
For ISAM: A number representing the error code.

Example: Use DbERR with the Evaluate operation to clear the last error, if one exists, immediately before the operation that needs to be monitored for errors.

Note: The error number or text returned depends on the underlying DBMS used for data storage.

DbExist

Verify Table Existence on Disk

Checks whether a specified data table exists on disk, and returns a logical True or False accordingly.

Syntax: DbExist(number,tablespec)

Parameters: *number*: A number that represents the table's sequence number in the Table repository.

tablespec: This parameter is used as an alternative to *number*. If not needed, enter ". If needed, enter an alpha string that represents the table specification. The string may contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: Boolean (True, False)

Examples: DbExist(1,'') or
DbExist('1'FILE,'')
return True if the table exists on disk.
DbExist (1,'FSTFIL.DAT')
attempts to check the first line of the Table repository, using
the OS name 'FSTFIL.DAT' in the current directory.
Note: Using the literal FILE will insure that the correct table
will be found even if the table's position in the Table
repository has changed. The table sequence number and
path behave exactly as defined in the Table repository. For
more information refer to Model Repositories in Chapter 4,
Tables.

DbExist with a table number parameter of 0 (zero), or greater than the
number of tables in the Table repository, will always return False.

See also: DbDel, DbRecs, DbSize

DbName

Database table name
Identifies a database table by name from the Table repository.

Syntax: DbName(number)

Parameter: *number*: The row entry in a Table repository.

Returns: DB table name from row(number) in the Table repository.

Examples: DbName(3) or
DbName('3'FILE) each return
the name of the table shown in the 3rd row of the Table
repository.

Note: The use of the FILE literal will assure that the correct entry
will be chosen even if the table's position in the Table
repository has changed. The table sequence number and
path behave exactly as defined in the Table repository. Refer
to the Model Repository in the Chapter 4, Tables.

DbRecs

Table Rows Count
Returns the number of rows in a database table.

Syntax: DbRecs(number,tablespec)

Parameters: *number*: A number that represents the table's sequence number in the Table repository.
tablespec: This parameter is used as an alternative to *number*. If not needed, enter "". If needed, enter an alpha string that represents the table specification. The string may contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: Number (0 if empty).

Example: DbRecs(1,") or
 DbRecs('1'FILE,")
 returns the number of rows contained in the first table of the Table repository.
 DbRecs(1,'FSTFIL.DAT')
 returns the number of rows in the FSTFIL.DAT table in the current directory.

Note: Using the literal FILE will insure that the correct table will be counted even if the table's position in the Table repository has changed. The table sequence number and path behave exactly as defined in the data dictionary. Refer to the Model Repository in Chapter 4, Tables.

See also: DbDel, DEBEXIST, DbSize

DbReload

Database Load

Reloads a resident table during runtime.

Returns a logical value indicating the success or failure of the load.

With a specific path as the second parameter, DbReload deletes the old resident table path and loads the new path of the resident table.

Syntax: DbReload (number,tablespec)

Parameters: *number*: A number that represents the table's sequence number in the Table repository.

tablespec: This parameter is used as an alternative to *number*.

If not needed, enter "". If needed, enter the complete alpha string that represents the table specification.
 The string may contain a path.

If the path is not indicated, eDeveloper assumes the current directory.

Returns: True for a successful load
False for a failed load

Example: DbReload(3,"")

Note: DbReload loads only the resident tables of the current context.

This function can only be used if the Load Resident Tables setting (Settings/Environment/Preferences) is set to Yes.

DbRound

Database round function for SQL databases.

Returns a numeric expression, rounded to the specified number of decimal places. The DbRound function rounds a number according to the behavior of the SQL database. The returned value is different than the value returned from the Round function.

Syntax: DbRound(numeric_expression,length)

Parameters: *numeric_expression* - Is an expression of a numeric variable or constant.

length - Is the number of decimal places to which the numeric expression is rounded. It's required that length be a numeric value. When the length is a positive number, the numeric expression is rounded to the number of decimal places specified by the length. When the length is a negative number, the numeric expression is round to the left side of the decimal point.

Returns: Returns a numeric value.

Remarks: DbRound always returns a numeric value. If the length is negative and larger than the number of digits to the left of the decimal point, the DbRound function returns 0.

Examples:.

Example	Result
DbRound(657.4545,2)	657.45
DbRound(657.4552,2)	657.46
DbRound(657.37,-4)	0
DbRound(657.37,-1)	660
DbRound(657.37,-2)	700
DbRound(657.37,-3)	1000

DbSize

Table Size

Returns the size of a database table.

Syntax: DbSize(number,tablespec)

Parameters: *number*: A number that represents the table's sequence number in the Table repository.

tablespec: This parameter is used as an alternative to *number*. If not needed, enter ". If needed, enter an alpha string that represents the table specification. The string may contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: Number (0 if no table exists).

Examples: DbSize(1,") or
DbExist('1'FILE,")
return the number of bytes of data contained in the first table of the Table repository. The actual size of the table on disc may be larger.

DbSize (1,'FSTFIL.DAT') returns the number of bytes contained in the FSTFIL.DAT table in the current directory.

Note: Using the literal FILE will insure that the correct table will be used even if the table's position in the Table repository has changed. The table sequence number and path behave exactly as defined in the Table repository. Refer to Chapter 4, Tables.

See also: DbDel, DbExist, DbRecs

DDEBegin

DDE Begin
Creates a session.

Syntax: *DDEBegin (service, topic)*

Parameters: *service*: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord, for MS Word for Windows, or Excel, for MS Excel.
topic: String, depending on the service.

Returns: True if the DDE server is already connected or a new connection has been established, or False for failure initializing the DDE or connecting to the DDE.

Example: DDEBegin('Excel','c:\docs\budget.xls')
Initiates a DDE session control by the user, independent of the DDE process. The DDE session remains open until the user selects the DDEEnd function.

See also: DDEEnd, DDEGet, DDEPoke, DDERR, DDExec

DDEEnd

DDE End
Terminates a session.

Syntax: *DDEEND (service, topic)*

Parameters: *service*: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord, for MS Word for Windows, or Excel, for MS Excel.
topic: String, depending on the service.

Returns: True for successful completion, or False for failure if the service and topic were not started by the DDEBegin function.

Example: DDEEnd('Excel','c:\docs\budget.xls')

See also: DDEBegin, DDEGet, DDEPoke, DDERR, DDExec

DDEGet

DDE Get
Get a string of characters from a DDE server

DDEGet returns a string value of size *length* from the specified DDE server. Each DDE server application provides access to its services via a

combination of the three identifiers: *service*, *topic*, and *item*. Taken together, the three identifiers provide a unique identification of the service.

Syntax: DDEGet (service,topic,item,len)

Parameters: *service*: The main identifier of the DDE service. Usually this is an application name such as WinWord for MS Word for Windows, or Excel for MS Excel.

topic: The area within the server application with which you want to exchange information. A topic may be a document name in MS Word for Windows or a spreadsheet in MS Excel. Server applications usually provide a system topic as standard practice. The system topic provides information about the application and topics that may be accessed by DDE. The system topic services can be requested through the item parameter. For example, the MS Excel system topic has a system item that returns the items that are available in the system topic. The format of the information returned by the system topic depends on the server application.

item: Further defines the exact data item for the exchange. Together with service and topic it points to a unique item. For example, a paragraph in a Word for Windows document may be read by DDE if the topic specifies the document name, and the item points to a bookmark in that document that marks the required paragraph.

len: The maximum length of the information that will be returned by the function. If the information returned is less than the maximum length, its trailer will be padded with blanks.

Returns: A character string of size *len*. If the DDEGet fails, the string will be empty. The string returned is provided by the DDE server application according to the *service*, *topic*, and *item* parameters of the function.

Examples: DDEGet('WinWord', 'c:\docs\ddetest.doc', 'toMagic',2000) reads a paragraph from an MS Word for Windows document that is marked with a bookmark in Word where 'c:\docs\ddetest.doc' is the document name; 'toMagic' is the bookmark name, and 2000 is the maximum size in bytes of the paragraph that

will be returned by the function.

Note that the single quotes are required on the first three columns, all string columns.

DDEGet('Excel','c:\docs\budget.xls','R19C1:R22C7', 2000)
reads a range of cells from an MS Excel spreadsheet where:
'c:\docs\budget.doc' is the spreadsheet name,
'R19C1:R22C7' is the cell range, and
2000 is the maximum size in bytes that will be returned by
the function.

eDeveloper requires the single quotes shown in these
examples to identify parameters as strings.

See also: DDEBegin, DDEEnd, DDEPoke, DDERR, DDExec

DDEPoke

DDE Poke

DDEPoke transfers a string from eDeveloper to the DDE server specified by the function's parameters. Each DDE server application provides access to its services via a combination of three identifiers: *service*, *topic*, and *item*. Taken together, the three identifiers provide a unique identification of the service. The string transferred by eDeveloper will be inserted to the server application at the location identified by *service*, *topic*, and *item*.

Syntax: DDEPoke (service,topic,item,data)

Parameters: *service*: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord for MS Word for Windows, or Excel for MS Excel.

topic: Provides a definition of the area within the server application with which you wish to exchange information. A topic may be a document name in MS Word for Windows or a spreadsheet in MS Excel.

item: Further defines the exact data item for the exchange. Together with service and topic it points to a unique item. For example, a paragraph in a Word for Windows document may be inserted by DDE if the topic specifies the document name, and the item points to an empty bookmark in that document that marks the required paragraph.

data: The data string that will be passed by the function to the server application.

Returns: True for successful completion, or
False for failure to poke the data to the server application.

Note: The information transferred to the server application may have to follow some formatting rules dictated by the server application. Refer to the server application documentation for details.

Examples: DDEPoke ('WinWord', 'c:\docs\ddetest.doc', 'fromMagic', 'This paragraph was planted by eDeveloper using DDE')
Will write a paragraph to an MS Word for Windows document, where the paragraph has been marked with a bookmark in Word, and where
'c:\docs\ddetest.doc' is the document name,
'fromMagic' is the bookmark name, and
'This paragraph was planted by eDeveloper using DDE' is the data that will be transferred by the function to the server application.

DDEPoke ('Excel', 'c:\docs\budget.xls', 'R1C1', '100')
Will write the number 100 into an MS Excel spreadsheet cell, where: 'c:\docs\budget.doc' is the spreadsheet name, and
'R1C1' is the first cell of the spreadsheet that will receive the information passed by the function.

eDeveloper requires the single quotes shown in this example, to identify columns as strings.

See also: DDEBegin, DDEEnd, DDEGet, DDERR, DDExec

DDERR

DDE Error

DDERR retrieves the last error that occurred during an eDeveloper DDE conversation. A subsequent call to DDERR clears the previously reported error code and resets the return value. This function retrieves useful information for debugging purposes.

Syntax: DDERR ().

Parameters: None

Returns: A numeric value ranging between 0 and 15, with the following meanings:

- 0 No error in the last DDE operation, or a reset return value if this is the second consecutive call to the function
- 1 Failure to initialize the DDEML system
- 2 Failure to connect to the server (could be a wrong or missing *service* parameter in the preceding DDE call)
- 3 Server was busy during the DDE call and could not service the call
- 4 Server did not process the DDE service required (could be a wrong parameter, an invalid combination of parameters, or an invalid required service combination)
- 5 The last DDEGet failed (server could not provide the required *item*)
- 6 No *item* was specified on a GET or POKE request
- 7 No *command* was specified on an EXEC request
- 14 Unknown type of error
- 15 Attempt to execute a DDE function on an operating system other than Windows

See also: DDEBegin, DDEEnd, DDEGet, DDEexec, DDEPoke

DDEexec

DDE Execute

DDEexec transfers a command string from eDeveloper to the DDE server specified by the function's parameters. Each DDE server application provides access to its services via a combination of three identifiers: *service*, *topic*, and *item*. Taken together, the three identifiers provide a unique identification of the service. The command string transferred by eDeveloper will be transferred to the server application, which in turn will

try to execute it. The command string must follow strict DDE format rules. The command contents must be a valid server application's command. If the command contents are not a valid server application command, the server application will fail.

Syntax: DDEExec (*service,topic,item,command*)

Parameters: *service*: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord, for MS Word for Windows, or Excel, for MS Excel.

topic: when used with DDEExec, the topic will usually be 'System', as the server application, represented by the System topic, rather than a data object within the server application, is responsible for the exchange.

item: Further defines the exact data item for the exchange. When used with DDEExec, it will usually remain empty.

command: The command string that will be passed by the function to the server application that will then execute it. DDE commands must be contained within square brackets [].

Several commands may be included within one command string, each in its own brackets, separated by blanks.

For example:

[command1]

[command2(parameter1)]

[command3(parameter1, parameter2, parameter3)]

[command1] [command3(parameter1, parameter2, parameter3)]

The command string must contain a valid server application command. Refer to the server application documentation for details about command syntax.

Returns: True for successful completion, or False for failure to execute the command at the server application.

Examples: DDEExec ('WinWord', 'System', '',
'[FileOpen "c:\docs\ddetest.doc"]')
Opens an MS Word for Windows document where:
'[FileOpen "c:\docs\ddetest.doc"]'

is the command transferred by the function to the server application for execution.

```
DDExec ('Excel', 'System', "",  
[run("MACROS.XLM!FormatCells")])
```

Executes the MS Excel macro (FormatCells).

```
DDExec ('Excel','System','',  
[Open("c:\docs\test.xls")])
```

Executes the MS Excel system.

eDeveloper requires the single quotes shown in these examples, to identify columns as strings.

MS Excel requires the double quotes shown in these examples.

See also: DDEBegin, DDEEnd, DDEGet, DDEPoke, DDERR

DeCipher

Converts an encrypted buffer to a buffer containing an Alpha string or a BLOB.

Syntax: DeCipher(Cipher ID, Buffer, Key [, Mode, IV])

Parameters: *ID* - A number representing the selected cryptographic algorithm.

Buffer - A BLOB with the encrypted buffer.

Key - A BLOB string containing the encryption key.

Mode - An Alpha string containing the selected mode for the encryption method selected by the Cipher ID. If a mode is not selected for a method that requires a mode, CBC is used as the default mode. See the Mode table below.

IV - A BLOB string containing an initialization vector. This parameter is optional.

Returns: A BLOB containing the decrypted buffer.

If the Cipher ID is not valid, or the key length does not match the Cipher ID, the function returns a Null value.

Note: See the information about the Cipher function for the supported encryption methods.

See also: Cipher, EncryptionError

Del

Delete Characters

Deletes characters from an alpha string.

Syntax: Del(string,start,length)

Parameters: *string*: An alpha string or an alpha string expression.

start: The position of the first character to be deleted.

length: The number of characters to be deleted, beginning with position *start* and proceeding rightward.

Returns: The source string with the specified segment deleted.

Examples: Del('ABCD',2,1)

deletes the second letter of the string, and returns 'ACD'.

If X contains a character string of a length greater than or equal to 2, the following expression removes either the first or second character, or leaves the string intact, according to the value that appears in field Y (negative, positive or zero).

IF (Y<0,Del(X,1,1),IF(Y>0,Del(X,2,1),X))

See also: Ins, Fill, Rep

Delay

Freeze all activity for *n* tenths of a second

The Delay function stops the execution of the current process for a specified period of time. The delay is terminated when the time specified for it has passed, or with the first user keystroke.

Syntax: Delay(10th of seconds)

Parameter: *number*: Number of tenths of a second to delay the current process.

Returns: True

Example: Delay(10)

will cause a 1.0 second delay, and then the calling process will resume. If a key is pressed while the delay function is active, the delay will be immediately terminated.

DiscSrvr

Disconnect Server

Disconnects a server no longer required by eDeveloper. This function releases unnecessary server connections.

Syntax: DiscSrvr(string)

Parameter: *string*: A string defining a server in the Server repository

Returns: Boolean (True, False)
True means that the disconnect operation from the specified server has succeeded.
False means that the disconnect operation has failed, for one of the following reasons:
The server name does not exist in the Server repository
No connection existed to the server
eDeveloper still requires the server connection
The disconnect operation failed.

DOW

Day of Week
Returns the number of the day of the week from a date, where Sunday is 1, Monday is 2, etc.

Syntax: DOW(date)

Parameter: *date*: A date

Returns: Number

Examples: DOW('01/28/96'&Date) (representing a Wednesday) returns 4.

The following expression displays a message if the system date is a Sunday:

IF(DOW(Date())=1,'Never on Sunday','')

See also: CDOW, Day, NDOW

DragSetCrsr

The DragSetCrsr function determines whether the cursor file is defined as Copy mode or as None mode.

Syntax: DragSetCrsr(mode, cur file)

Parameters: *mode* - A number that represents the cursor mode:

1 - The cursor is set for Copy mode. The Copy mode image appears when a dragged data value can be dropped on a form, control, or external application.

2 - The cursor is set for None mode. The None mode image appears when a dragged data value cannot be dropped on a form, control, or external application.

cur file - The cursor file name and path (*.cur).

Returns: True when the cursor file is located and loaded successfully.

Note: If the cursor mode number is not supported, the mode will be considered copy mode.

DragSetData The DragSetData function determines the data content and format of a control that is not defined for automatic data handling. This function also lets you assign the data content to a different data format.

Syntax: DragSetData(*data content*, *data format*, *user defined format*)

Parameters: *data content* - A string value that represents the actual data as part of the drag operation data.

data format - A numeric value that represents the format of the data stored by the operating system. The data formats are:

0 - User-defined format

1 -Text

2 -OEM text

3 -Richtext

4 -HTML

5 - Hyperlink

6 - File name and path

user defined format - An optional string parameter that names a user-defined format. This parameter is required when the data format parameter is set to 0 but is not relevant for other data formats.

Returns: True only when evaluated within the scope of an active drag begin event.

Example: DragSetData('ABC', 1) results in displaying the 'ABC' string as the text format Drag operation data.

DragSetData('DEF', 0, 'My format') results in setting the 'DEF' string as the user-based format drag operation data.

Note: The DragSetData function can be performed several times to define data for different types of formats. Each data format can only be set once with data in a single drag

operation. If the function is evaluated twice for the same data format within the same drag operation, only the last evaluation will be applied.

DropFormat The DropFormat function is used to check whether a defined data format is supported by a drop operation.

Syntax: DropFormat(data format, user defined format)

Parameters: *data format* - The numeric value for the data format stored in the operating system. The data formats are:

0 - User-defined format

1 - Text

2 - OEM text

3 - Rich text

4 - HTML

5 - Hyperlink

6 - File name and path

user defined format - This parameter is required when the data format parameter is set to 0 but is not relevant for other data formats.

Returns: True if the data format is supported.

Example: DropFormat(0, 'My format') returns True if the user-defined format is supported by the drag operation data.

DropFormat(6) returns True if the File format is supported by the drag operation data.

Note: To handle drop data of user-defined formats, you must first set the names of the expected user-defined formats in the Drop Data supported user formats environment setting on the External tab of the Environment Settings dialog.

DropGetData The DropGetData function retrieves the data from the drag and drop operation using the defined format.

Syntax: DropGetData(data format, user defined format)

Parameters: *data format* - A numeric value for the data format stored in the operating system. The data formats are:

- 0 - User-defined
- 1 - Text
- 2 - OEM text
- 3 - Rich text
- 4 - HTML
- 5 - Hyperlink
- 6 - File name and path

user defined format - An optional string parameter that specifies a user-defined format by its name. This parameter is required when the data format parameter is set to 0 but is not relevant for other data formats.

Returns: A string value of the retrieved data. If the format is not supported, the function returns a Null.

Example: DropGetData(0, 'My format') returns the string value of the user-defined format data.

DropGetData(1) returns the string value of the text format data.

DropGetData(4) returns the string value of the HTML format data.

DropGetData(6) returns the full path name of the dragged file. If multiple files are marked, dragged, and dropped, the function returns the full path name of each file, delimited by a comma.

Note: To handle the drop data of user-defined formats, you must first set the names of the expected user-defined formats in the Drop Data supported user formats environment setting on the External tab of the Environment Settings dialog.

DropMouseX The DropMouseX function retrieves the Mouse Cursor X coordinate of the current form when the dragged data value is dropped.

Syntax: DropMouseX()

Parameters: No parameters required.

Returns: The numeric value of the Mouse Cursor X coordinate

Note: When a drop occurs outside the form's client area, such as the form title, the X coordinate will be returned as a negative value.

DropMouseY

The DropMouseY function retrieves the Mouse Cursor Y coordinate of the current form when the data value is dropped.

Syntax: DropMouseY()

Returns: The numeric value of the Mouse Cursor Y coordinate.

Note: When a drop occurs outside the form's client area, such as the form title, the Y coordinate will be returned as a negative value.

DStr

Date to Picture String Conversion

Converts a date or a date expression to a character string, according to a format.

Syntax: DStr(date, picture)

Parameters: *date*: A date

picture: The format of the resulting character string. For a full description of the pictures, refer the Pictures section in Chapter 3, Data Items.

Returns: Alpha string

Examples: When the Date Mode is set to American,
DStr ('01/28/96'Date,'WWWWWWWWW, MMMMMMMM DD, YYYY')returns 'Tuesday, January 28, 1996'
When the Date Mode is set to European, DStr('28/01/96'Date,'DD/MM/YY') returns 28/01/96

Note: You must use the Date Mode environment setting to set the date format, such as:

- American -- MM/DD/YY
- European -- DD/MM/YY
- Scandivanian -- YY/MM/DD
- Buddhist -- DD/MM/YY (YY is the current non-Buddhist date + 36)

The date format that you selected must be reflected in the Date Literal string.

For example, If you have selected American for the Date Mode, you must enter the Date Literal string for this function as '01/28/96'Date.

If the Date Literal string is inconsistent with the Date Mode value, a Bad Date error appears on the status bar when the Syntax Checker is evoked.

The word, Date, following '01/28/92' identifies the string as a Date Literal string, and should not be confused with the Date() function.

See also: DVal

DVal

Date String to Numeric Value

Converts a date entered or stored as a character string to a numeric value. The numeric value represents the number of days elapsed since the day before the first day of the 1st century (01/01/0001) until the date that is being converted.

Syntax: DVal(datestring,picture)

Parameter: *datestring*: A character string or an alpha expression that can be interpreted as a date (e.g., '01/01/97', 'Jan 1, 1997').

picture: The format for *datestring*. This parameter is required for eDeveloper to read and interpret the character string or expression.

Returns: Number

Examples: DVal('01/01/97','MM/DD/YY') and
DVal('Jan 1, 1997','MMM D, YYYY')
each returns 729025

where variable A contains 'January 1, 1997'

DVal(A,'MMMMMMMMMM DD, YYYY') where variable A contains 'January 1,1997' also returns 729025

DVal('01/01/01','MM/DD/YY') and DVal('Jan 1, 2001','MMM D, YYYY') each returns '01/01/2001'Date.

where variable A contains 'January 12, 2002'
DVal(A,'MMMMMMMMMM DD, YYYY') also returns '12/01/2002'Date.

See also: DStr

Note: If *picture* does not correspond to the format of *datestring*, the function returns 0.

EditGet

Control value

Returns the control value in edit mode.

Syntax: EditGet()

Returns: The edited value of the field from which the last handler was invoked.

Note: The EditGet function returns the edited value of the last control from which the logic (handler) was invoked.
The EditGet function returns the value of the edited control according to the variable's attribute.
In controls other than Edit control, which cannot be edited, the value will be the current value of the control or variable.
If the EditGet functions is evaluated in the flow when the Magic engine is not parked on any control the function returns NULL.
If the handler invoked on a control is set on a Force Record Exit event the EditGet will return NULL.

EditSet

Sets the edited value of the control that invoked the last handler.

Syntax: EditSet (value)

Parameter: value - the value entered in the control.

Returns: A boolean value that indicates the success (True) or failure (False) of the function.

Note: The value provided in this function must match the attribute of the variable associated with the control. If the attributes are not identical, the function will return a False value.
For controls other than the Edit control, which cannot be edited, the set value is the variable associated with the

control.

If the EditSet function is evaluated in the flow when the Magic engine is not parked on any control, the function will return a False value.

If the handler invoked on a control is set to a Force Control Exit event, the EditSet function will set the value of the variable associated with the control.

If the handler invoked on a control is set on a Force Record Exit event, the EditSet function will return a False value.

EJBCreate Obtains a new instance of an Enterprise Java Bean.

Syntax: EJBCreate('<JNDI name>'[, 'jndi properties string'])

Parameters: <JNDI name> - Must be alpha-numeric
jndi properties - Must be alpha-numeric

Returns: A pseudo-reference to a new instance. The return value must be a BLOB variable.

Example: EJBCreate('JNDI_ejb1')

EJBExplore Provides a description of an Enterprise Java Bean.

Syntax: EJBExplore('<JNDI name>' [, 'jndi properties string'])

Parameters: <JNDI name> - Must be alpha-numeric
jndi properties - Must be alpha-numeric

Returns: An XML description of the EJB. The return value must be an alpha-numeric variable.

EncryptionError Returns the last error message for the Cipher and DeCipher functions. The function scope is per context.

Syntax: EncryptionError()

Parameters: None

Returns: The last error message for the Cipher and DeCipher functions.

EOF

End of File

Reports whether the specified I/O file is at the end of information.

Whenever an I/O file reaches the end of information, an EOF flag is raised.

This flag is queried by the EOF function.

Syntax: EOF(generation,file)

Parameters: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

file: A number that represents the sequence number of the I/O file in the current task.

Returns: Logical

Example: EOF(0,1)

returns True after processing the last record of the first I/O file in the current task.

Note: EOF is used for input I/O files. It should be used as an end condition in tasks that read information from operating system text (I/O) files. The following expression when used as an end condition in a task reading from a single I/O file will cause task termination as soon as the last piece of information is read from the file: EOF (0,1).

See also: EOP, Line, Page

EOM

End of month

Returns the date of the end of the month specified in the parameter.

Syntax: EOM(date)

Parameter: *date*: A date or a date expression.

Returns: Value of type Date.

Example: EOM ('05/05/97'Date)
returns '05/31/97'.

See also: BOY, BOM, EOY, AddDate, CDOW, and all Date functions.

EOP

End of Page

Reports whether the line counter of the current page in the specified I/O file is greater than the Lines parameter specified for it. If the lines parameter contains 0, eDeveloper uses the Lines parameter specified for

the printer associated with the I/O file. EOP is raised if Form lines is specified and eDeveloper line count is greater than Form lines.

Syntax: EOP(generation,file)

Parameter: *generation:* A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

file: A number that represents the sequence number of the output file in the current task.

Returns: Logical True if the specified I/O file is at the EOP, False if not.

Example: EOP(0,1)
returns True at the end of a page, while processing the first output file in the current task.

See also: EOF, Line, Page

EOY

End of Year

Returns the date of the end of the year specified in the parameter.

Syntax: EOY(date)

Parameter: *date:* A date or a date expression.

Returns: Value of type Date.

Example: EOY ('11/17/97'Date)
returns '12/31/97'

See also: BOM, BOY, EOM, AddDate, CDOW, and all Date functions.

ErrDatabaseName

Error in Database - Name is not specified.

Returns the name of the database where the error occurred.

Syntax: ErrDatabaseName ()

Parameter: None.

Returns: Name of the database where the error occurred.

ErrDbmsCode Error in Database - Code is not specified.

Returns the DBMS original error code.

Syntax: ErrDbmsCode ()
Parameter: None
Returns: The DBMS original error code.
Note: ErrDbmsCode can be used for fatal and unexpected errors, assuming that the developer is familiar with DBMS internal error codes.

ErrDbmsMessage

Error in Database - Message is not specified.

Returns the original DBMS error message.

Syntax: ErrDbmsMessage ()
Parameter: None.
Returns: Original DBMS error message.
Note: Can be used for fatal and unexpected errors. eDeveloper gets the internal message from DBMS and displays it.

ErrMagicName Database Error - eDeveloper Literal is not specified.

Contains the eDeveloper literal for the error that occurred.

Syntax: ErrMagicName ()
Parameter: None
Returns: An alpha string that contains the eDeveloper literal for the error that occurred.
Note: This function is needed when using a handler of type 'any'.

ErrPosition

Database Error - Position is not specified.

Refers to the position of the record where the error occurred.

Syntax: ErrPosition ()
Parameter: None
Returns: BLOB expression with the internal position of the record where the error occurred.

Note: If outside the scope of an error, the function returns a blank. The function can be used as an expression to display the record where an error occurred.

ErrTableName Database Error - Physical Table Name is not specified.

Returns the physical name of the table on which the error has occurred.

Syntax: ErrTableName ()

Parameter: None

Returns: Physical name of the table on which the error has occurred.

Note: If the error handler is defined for a specific table, calling this function is unnecessary. If not, it can be used to determine if the error occurred on the Main table or one of the linked tables. When the function will be irrelevant, the return value will be blank.

EuroCnv European Currency Conversion

Returns the converted value from currency to currency based on the conversion values of the euro in the Currency table.

Syntax: EuroCnv(From,To,Value)

Parameters: *From:* The currency that the user converts from(Alpha).
To: The currency that the user converts to (Alpha).
Value: The value of the currency to be converted (Numeric).

Returns: The converted value in the To currency.

Example: EuroCnv('ITL','DEM',A)
Converts the value A from Italian lira to German DM.
EuroCnv('ITL', EURO. 50*B)
Converts 50 times the value B from Italian lira to euro.
EuroCnv(A,B,C)
Converts the value C from Currency A to Currency B.

EuroDel European Currency Code Delete

Allows the developer to remove the Currency code from the European Currency Conversion Table.

Syntax: EuroDel (Selected Currency)
Parameter: Selected Currency - A string value that represents a currency code listed in the European Currency Conversion Table (ECCT).
Returns: A True value is returned if the currency entry is deleted.
Example: . EuroDel('FRF') removes the French Franc currency from the ECCT.
Note: The EuroDel function can be used with the Evaluate operation only in runtime. EuroDel (A) is not sufficient, specify EuroDel (A).

EuroGet

European Currency Table Access
Lets the user to access the currency list in deployment mode.

Syntax: EuroGet ()
Parameter: None
Returns: The Base Currency string is returned.
Example: EuroGet () returns EURO for a default setting of the Base Currency property.

EuroSet

European Currency Table - Base Currency Set
Modifies the Base Currency in deployment (runtime) mode.

Syntax: EuroSet (Selected Currency)
Parameter: *Selected Currency* - A string value that represents a currency code.
Returns: A True value is returned if the Base Currency is modified.
Example: EuroSet ('FRF'=0) sets the Base Currency parameter to French Francs.
Note:
1. Changing the Base Currency property of an application affects the database rows that are based on the currency value.
2. EuroSet can be used with the Evaluate operation.

EuroUpd

Updates an entry to the Currency table.

Syntax: EuroUpd (Code Name, Name, Precision, Euro Rate)

Parameters: *Code Name:* Currency Code name (maximum 4 characters)
Name: Currency description name. (maximum 20 characters)
Precision: The precision of results of calculations based on this entry. Valid values are 0,1,2.
Euro Rate: The exchange rate of the currency to 1 euro.

Returns: A True value is returned if the entry is updated.

Note: 1. An update to a non-existent currency will append the currency to the Currency table.
 2. The Conversion table does not allow duplicate code names.

EvalStr

Evaluate String

This function lets you evaluate dynamic expressions that may be constructed at runtime.

Syntax: EvalStr(string, default value)

Parameters: string: A string that evaluates to a valid eDeveloper expression.
 default value: The function returns the default value of the first parameter that is found to be an invalid expression. The default value of the function must match the expected attribute of the function according to the context of where the function is placed in the expression.
 For example, in the expression 1 + EvalStr(string, default value), the expected attribute of the default function is Numeric.

Returns: The evaluated value of the expression given by the first parameter. If this expression is invalid, the default value is returned.

Example: EvalStr(A,0) , where A='3 + 2', returns the value of 5.
 EvalStr(A,0), where A='3 +', returns the value of 0.

Note: If the expression represents a single quote character (for a string or a literal), you should provide two single quotes for each single quote.

For example:

EvalStr('DbDel("1"FILE,"MYFILE.DAT").orEvalStr("'A" & "B"',) = 'AB'If strings are not wrapped by quotes, they are regarded as variable references:EvalStr('A & B') = variable A & variable B.Although using the variable letter codes is legal, it is best to avoid referring to variables by their codes because their location may vary.If you wish to refer to variables, it is better to refer to them by their names using the VarCurrN and VarIndex functions. For example:EvalStr('VarCurrN("NAME") andVarCurrN('LAST_NAME')'0)

EvalStrInfo

Checks a string representing an eDeveloper expression that can be evaluated using the EvalStr function and can retrieve information about the expression.

Syntax: EvalStrInfo(*expression string,option*)

Parameters: *expression string* – A string representing an expression that can be evaluated using the EvalStr function.

option – A numeric value that sets the type of information that can be retrieved.

The options are:

- 1 – Attribute
- 2 - Expression Parser Error
- 3 - Result Expression

Returns: A string with the information according to the selected option.

Options: Attribute (option=1)

If the string represents a valid expression. the function returns the resolved attribute of the expression:

A – The expression returns an Alpha value.

N – The expression returns a Numeric value.

L – The expression returns a Logical value.

B – The expression returns a BLOB-based value. This includes BLOB, Vector, OLE, and Active-X data values.

- M – The expression returns a Memo attribute.
- D – The expression returns a Date attribute.
- T – The expression returns a Time attribute.
- * – The expression returns an undetermined attribute.
- Error – The expression is invalid.

Note: Date and Time attributes are returned only if the expression is a single function that is date or time-related, such as Time(), or a simple variable letter combination of a Date or Time variable, such as 'A'. For more complex expressions with Date or Time values, the expression will be identified as a Numeric attribute.

Also M is returned only if the expression is a simple variable letter combination of a Memo variable. For more complex expressions with Memo values, the expression will be identified as an Alpha attribute.

For example: EvalStrInfo('123',1) returns N

If the string represents an invalid expression, the function returns an error string.

For example: EvalStrInfo(` 123+` ,1) returns Error

EXP

Exponential
Calculates the exponential value of x.

- Syntax:** EXP(x)
- Parameter:** x: a numeric value
- Returns:** Number, the exponential function e^x of the function's argument.
- Example:** EXP(5)
returns e^5
- See also:** LOG

ExpCalc

Executes a function during the evaluation of an expression.

- Syntax:** ExpCalc (*Expression Specification*)

Parameter: *Expression Specification* - the expression number to execute represented by the EXP literal, such as '2'exp as a pointer to the second function in the expression list of the current task.

Returns: The evaluation value of the specified expression.

Examples: ExpCalc('2'EXP)
Executes expression #2.
ExpCalc(BA)
Executes expression position #2.

Note: The expression specification may be one of the following:
EXP literal (For example, '2' exp as a pointer to the second function in the expression list in the current task).
Constant numeric

File2Blb

File2Blb moves a file into a BLOB variable. This allows the developer to send files in memo requests.

Syntax: File2Blb (File Name)

Parameters: *File Name* (Alpha) - File Name.

Returns: BLOB - the file packed as a BLOB.

Example: File2Blb('C:\att1.rtf')
Combines an BLOB variable with an att1.rtf file.

File2OLE

Enables programmers to convert a file into an object BLOB in eDeveloper, either by linking or embedding.

Syntax: *File2OLE (file name, linked)*

Parameters: *file name*: The name of the file to be converted.

linked: True indicates that the result will be a link to the file;
False indicates that the file will be embedded.

Returns: The function will return a BLOB. If the BLOB is Null, the function was not able to perform the conversion.

Example: File2OLE('C:\att1.doc','TRUE'Logical)
Combines an OLE2 BLOB variable with a word file name att1.doc.

File2Req

File to requester

File2Req sends the contents of a BLOB to the requester.

Syntax: File2Req (*file name*)

Parameters: *file name* (Alpha) - The file name and path

Returns: True if the file is found and the engine is executed as an enterprise server. If not, this function returns False.

Note: Before you send data to the requester, you can define HTTP Header information by using the RqHTTPHeader function.

See also: Blob2Req

FileDLG

Accesses a Windows Open File dialog

Syntax: FileDLG(string1, string2)

Parameters: *string 1:* A file type

String 2: Groups of files

Returns: An Alpha string containing the file name and path.

FileListGet

Returns a list of the file names in a directory based on a set of file-name filters.

Syntax: FileListGet(*directory name*, *filter list*, *sub dir search*)

Parameters: *directory name* – A string representing a directory. The string can contain logical names.

filter list – A string representing a list of file filters, including wild card characters separated by the '|' character. For example: '*.xml|*.xsd'. If the value of this argument is an empty string, all files in the directory will be listed.

sub dir search – A boolean parameter. True means that the directory and all subdirectories are searched. False means that only the specified directory is searched.

Returns: A vector that contains alpha strings. Each cell has a string that is the file's name and relative path, for example: 'dir1\po.xml'. If no files are found, a Null is returned.

Example: FileListGet('c:\temp','*.xml|*.xsd','True'Log) returns a vector containing alpha strings. Each entry in the vector

contains a file name that is found in the 'C:\temp\directory' and its subdirectories, for example: 'dir1\po.xml'.

Fill

Fill a string

Repeats an alpha string or expression n times.

Syntax: Fill(string,times)

Parameters: *string*: An alpha string or expression.

times: The number of times the string will be repeated.

Returns: Alpha String

Example: Fill('*',5) creates a string of five asterisks '*****'

Note: This function accepts a maximum of 32k.

See also: Ins, Del, Rep

Fix

Extract-Truncate Number

Extracts a specified part of a number, real or integer.

Syntax: Fix(number,whole,decimal)

Parameters: *number*: The number subjected to the operation.

whole: The number of digits to be extracted from the integer part of *number*. eDeveloper counts the digits from the decimal separator leftward.

decimal: The number of digits to be extracted from the decimal part of *number*. eDeveloper counts the digits from the decimal separator right wards.

Returns: Number

Examples: Fix(12345.6789,3,3)
returns 345.678

The statement

IF(Fix(M,2,0)=99,N+1,N)

checks the number M; if the last two digits of its whole part are 99, it increments N.

See also: Round

Flip

Flip String to Mirror Image

Reverses an alpha string or the result of an alpha expression to its mirror image.

Syntax: Flip(string)

Parameter: *string*: An alpha string or alpha string expression.

Returns: Alpha string

Example: Flip('Good') returns 'dooG'

Flow

Check Flow Mode

Checks the flow mode of an operation.

Syntax: Flow(string)

Parameter: *string*: An alpha string that represents the flow mode:

N - next, step mode

F - fast mode forward

P - previous, step mode

R - reverse, fast mode

S - Select exit with select operation

C - cancel, exit without selection

Returns: Boolean (True) if the mode specified as input is the current operation mode.

Example: Flow('N') returns True if the operation is performed in Step mode.

See also: ViewMod, VarMod

FlwMtr

Appends a message string to the Activity Message list, and stops execution before the next operation in the eDeveloper engine flow.

Syntax: FlwMtr(message string, break expression)

Parameters: . *message string*: String that is to be appended to the Activity Message list.

Break expression: A boolean (True) expression that stops execution before the next operation in the eDeveloper engine flow.

Returns: Boolean - True or False values determine whether the next operation in the eDeveloper engine flow will be executed.

FlwMtrVars Instructs the flow monitor to append the Variables dataview to the current line of the Flow Monitor.

Syntax: FlwMtrVars()

Parameters: None

Returns: True when the engine reports to a flow monitor. False when the engine does not report to a flow monitor.

Note: Vector variables are appended in an expanded mode, displaying the content of their cells.

GetLang Get Starting Language
Displays the current selected language string. This is taken from the Starting Language setting in the Environment dialog.

Syntax: GetLang()

Parameters: None

Returns: String or empty

Example: GetLang()

See also: SetLang

GetParam Get Parameters
Retrieves values passed as parameters to programs using a hyperlink or Call Remote command and retrieves values set to global parameters using the SetParam function.

Syntax: GetParam (parameter name)

Parameter: *parameter name*: A string representing the name of a global parameter. If the parameter was set by the SetParam function, the name should be the same as that used in the SetParam function.

If the parameter was received by passed arguments from a Call Remote command, the name should be MGARG##, where ## represents the sequential number of the required parameter. The name MGARG0 returns the number of passed arguments.

If the parameter was received by passed arguments using a hyperlink, the name can either be MGARG##, as in the Call

Remote command, or the given name of the passed argument, as specified in the hyperlink or Submit Form operation.

Returns: Data according to the data type that was defined in the parameter.

Example: `GetParam('MyArgument')`
Returns the value of the global parameter 'MyArgument' set by the `SetParam` function or passed by a hyperlink.

`GetParam('MGARG0')`

Returns the number of passed arguments in a Call Remote command or hyperlink.

`GetParam('MGARG1')`

Returns the value of the first argument.

`GetParam ('MGARG2')`

Returns the value of the second argument.

`GetParam('MGARG#')`

Returns the value of the argument number (#).

Note: Unexpected results may occur if the parameter and variable data types are incompatible.

The function returns a NULL where the parameter name is not defined.

In an eDeveloper program call by a hyperlink, the following occurs:

The program may receive HTTP environment variables as global parameters that can be accessed by the `GetParam` function.

Sequential retrieving of arguments using the 'MGARG##' convention can be only used if the Arguments value is specified in the HTML form or hyperlink, and contains the comma delimited argument names.

See also: `SetParam`

GroupAdd

Adds a user to a group. Assigns a user to a user group in the Security File from within an application.

Syntax: `GroupAdd(user,group)`

Parameters: *user*: the userid of the user to be assigned to the user group given by the second parameter.
group: the group to which the user will be assigned.

Returns: Logical True or False according to the success or failure of the operation.

Example: GroupAdd('Accountant', 'Accounting') will assign the user Accountant to the user group called Accounting. The user Accountant will inherit all of the rights of the group Accounting.

Note: Only users with SUPERVISOR rights can use the function. Attempts by other users to use this function will be ignored.

HandledCtrl

This function returns the name of the control from which the current handler has been invoked.

Syntax: HandledCtrl()

Returns: String - The name of the control from which the current handler has been invoked.

Note: In a handler that was not invoked from a specific control, such as a Task, Record level handlers, or handlers in a batch task, the function returns a blank string. This function returns the name of controls that invoked the event, even if they are not parked; for example, in cases of mouseover and mouseout events.

HitZOrdr

Z-Order Level

This function returns the Z-Order number of a control, and has no parameters. This function is Window-specific.

Syntax: HitZOrdr()

Parameters: None

Returns: An integer between 1 and n that represents the Z-Order of a control in a form that has been encountered by the Control Hit event.

Example: HitZOrdr()

Hour

Hour of Time

Returns a number that represents the hours part of a time value or a time expression.

Syntax: Hour(time)

Parameter: *time*: A time value or a time expression.

Returns: Number (01-99)

Examples: Hour('12:00:00' Time)
returns 12

Where the variable A contains the time value 14:00:00,
the expression Hour(A)+2
returns 16

See also: Minute, Second

HStr

Decimal to Hexadecimal Conversion

Returns the hexadecimal (base 16) value of a decimal (base 10) number.

Syntax: HStr(numeric)

Parameter: *numeric*: A decimal number (a base 10 number) or a numeric expression that represents a decimal (base 10) number.

Returns: Alpha string (representing a hexadecimal number)

Examples: HStr(15) returns 'F'
HStr(16) returns '10'

See also: HVal

Note: Results left justified.

HTTPGet

Retrieves the returned HTML result of an HTTP request as a BLOB object.

Syntax: HTTPGet(URL, header line 1, header line 2, ...)

Parameters: URL – A string that represents an HTTP address and lets you retrieve information. When an eDeveloper client is accessing a web server that requires a user name and password, the parameter should be
HTTP://User:Password@[URL]

You can also use secret names, for example:

HTTP://%user_secretname%:%pass_secretname%@[URL]

Header lines – A string that provides additional request header information. You may specify as many header information strings as you need.

Returns: BLOB data – The actual file retrieved by using the URL. If the function fails to make the connection, a NULL value is returned.

Note:

1. The HTTPGet function can connect through a proxy server. The proxy server can be defined in the Magic.ini file or in the HTTP Proxy environment setting.
2. The retrieval phase of the function can be defined to stop after a defined-elapsed time. This HTTP timeout can be defined in the Magic.ini file.
3. When an HTML file or any textual data is retrieved, you can query the returned text stream using all the available string manipulation functions.
4. Use the File2Blb function to store the retrieved file on your disk.

Example: HTTPGet('http://www.magicsoftware.com/index.html',
'if-Modified-Since: Tue 7 Aug 12:00:00 2001',
'User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)')

This example requests the file specified by the first parameter. The following parameters are samples of additional request header information. The first parameter requests the page only if it was modified after the provided date. The second parameter provides the server with the user agent type that this request simulates.

HTTPLastHeader

Retrieves the value of an HTTP header entry from the entry name. The function queries the HTTP header information received from the last HTTP retrieval from the HTTPGet or HTTPPost functions.

Syntax: HTTPLastHeader(header title)

Parameter: header title - A string value that represents a header title

type. For example, Content-Type, Expires, or Content-Length.

Returns: A string value that represents the defined header entry. If the header title is a blank string, the return value is the entire header.

Example: After performing an HTTPGet operation that returns an HTML page, the HTTPLastHeader('Content-Type') returns 'text/html'.

When passing a blank string as the header title, the HTTPLastHeader("") returns:

```
'HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 13 July 2000 05:46:53 GMT
Content-Length: 2291
Content-Type: text/html
Cache-control: private'
```

Note: If the HTTPPost or HTTPGet functions were not used, the HTTPLastHeader function returns a blank string. If the Magic broker cannot locate the header title, the function also returns a blank string.

HTTPPost

Posts information via an HTTP request and returns an HTML or XML result of the HTTP request as a BLOB object.

Syntax: HTTPPost(*URL, body, header line 1, header line 2, ...*)

Parameters: URL - A string that represents an HTTP address and lets you activate and post information. When an eDeveloper client is accessing a web server that requires a user name and password, the parameter should be
HTTP://User:Password@[URL]

You can also use secret names, for example:

HTTP://%user_secretname%:%pass_secretname%@[URL]

Body – A BLOB field containing textual information that is posted to the URL, such as var1=100&VAR2=200.

Header lines – A string providing additional request header information. You can specify as many header information strings as required.

Returns: A BLOB data object with the information result from the posted HTTP message retrieved by the URL. If the function fails to make a connection, a NULL value is returned.

Note: The HTTPPost function can connect through a proxy server. You can define your proxy server in the Magic.ini file or in the HTTP Proxy environment setting on the External tab of the Environment dialog. The retrieval phase of the function can be set to stop by defining the HTTP timeout in the Magic.ini file.

When an HTML file or any textual data is retrieved, you can query the returned text stream by using the available string manipulation functions.

Use the File2Bib function to store the retrieved file to your disk.

Example: HTTPPOST('http://localhost/Service',A,'User-Agent "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)" ')
 where 'http://localhost/Service' is an example of a url. A is an example of a BLOB value that contains a collection of variables, and 'User-Agent "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)" ' is an example of a header string.

HVal

Hexadecimal to Decimal Conversion

Returns the decimal (base 10) value of a hexadecimal (base 16) number.

Syntax: HVal(string)

Parameter: *string*: An alpha string that represents a hexadecimal (base 16) number.

Returns: Number

Examples: HVal('FF') returns 255
 HVal('10') returns 16

See also: HStr

Idle

Check System Inactivity Time

Returns the time elapsed from the last keystroke received by eDeveloper, in tenths of a second. The Idle time depends on the keyboard idle seconds parameter in the Environment. The Idle function will only return a value if the number of seconds specified by the Environment keyboard idle

seconds parameter have elapsed. The result of the Idle function is cumulative in keyboard idle tenth-second units.

Syntax: Idle()

Parameter: None

Returns: 0, if less than keyboard Idle tenth-seconds have elapsed since last keystroke. A cumulative number in keyboard Idle tenth-second units if more than keyboard Idle tenth-seconds have elapsed since the last keystroke.

Example: With the Environment keyboard Idle seconds set to 5, then Idle() returns
0 after .1 second;
0 after .2 seconds;
0 after .4 seconds;
5 after .5 seconds;
5 after .8 seconds;
10 after 1.0 seconds.

IF

If-Then-Else

Evaluates a logical expression and returns one value if True (Then) and another if False (Else).

Syntax: IF(boolean,valuetrue,valuefalse)

Parameters: *boolean*: A Boolean expression.

valuetrue: The value returned if the condition evaluates to True.

valuefalse: The value returned if the condition evaluates to False.

Returns: Value.

Example: IF(A=1,'Blue','Green')
returns Blue if A=1 (True) and
returns Green if A not=1 (FALSE)

IN

Determines whether a selected value matches any value in a given collection of values. This function is available from iSeries OS/400 V5R2.

Syntax: IN(match value, value1, value2...)

Parameters: *match value* - The value that is matched with the following values.
values - Each parameter represents a value to be matched with the match value parameter.

Returns: True when the match value is matched with one of the following values. False is returned when the match value cannot be matched.

Example: IN('cat','bird','cat','dog') returns True.
 IN('Java','UNIX','Microsoft','Web Service') returns False.
 IN(5,1,2,3,5) returns True.
 If A is 01/01/2003,
 IN(A, '01/01/2003'd,'03/04/03'd) returns True.

Note: This function is not available for the iSeries SQL gateway.

INIGet

Query the Magic.ini file
 Returns an environment value from the Magic.ini file.

Syntax: INIGet(inientry)

Parameter: *inientry*: An alpha string indicating the MAGIC.INI file section and parameter to read '[section] parameter'

Returns: An alpha string indicating the Magic.INI file section and parameter.

Example: INIGet('owner')
 returns the current owner value
 INIGet('[MYAPPL]LAST_CUST')
 returns an application dependent value

Note: If no MAGIC.INI section is specified, the '[MAGIC_ENV]' section is used by default.
 The *inientry* parameter is case-insensitive.
 This function could be used to access eDeveloper-specific columns, or to retrieve application-dependent global data. If used to retrieve updatable application data, you should ensure that each user has their own MAGIC.INI file. If you need to share global data and counters among multiple users, you should use standard DB tables.

For performance reasons, you may prefer to set the Resident MAGIC.INI setting in the Environment dialog to Yes. Refer to the discussions of the MAGIC.INI file and the command line in Chapter 2, Settings.

See also: INIPut

INIGetLn

Allows the user to select a specific value from a line in a section in the MAGIC.INI file.

Retrieves a value from a line section in the MAGIC.INI file.

Syntax: INIGetLn (Section Name, Line Number)

Parameters: *Section Name* (Alpha) - The name of the requested section.
Line Number (Numeric) - The requested line section.

Returns: Alpha - the MAGIC.INI requested section line string.

Remarks: Returns empty string when line<1.
Returns empty string when it comes to the first empty line.
Calling arguments will be returned as the first lines of the Magic section.

Example: INIGetLn ('[MAGIC_ENV]',3)
Returns the value of line #3 from the MAGIC_ENV section of the MAGIC.INI file.

Note: It is impossible to define a space as a delimiter.

INIPut

INIPut Update Environment Value

Updates an environment value in the MAGIC.INI file.

Syntax: INIPut(*inientry*, *force write*)

Parameter: *inientry*: An alpha string containing multiple entries, each indicating a MAGIC.INI section and parameter, and a value to update - '[section]parameter=value'.

force write: Boolean value. In a multi-threaded eDeveloper enterprise server, each runtime context of every thread keeps a separate copy of the MAGIC.INI. When this parameter is False, the new INI modification will only be written into the local MAGIC.INI copy. In a multi-threaded server, the change will only be reflected for the running thread.

When this parameter is set to True, then the new INI modification will be written into the physical INI. Any new loaded thread will have the new modifications in MAGIC.INI copy.

Returns: True if successful.

Example: INIPut('Owner=John, Century=1920','FALSE'LOG)
sets the current Owner and century value.

See also: INIGet

Note: If no MAGIC.INI section is specified, the '[MAGIC_ENV]' is used by default.

The *inientry* parameter is case-insensitive. If an entry is not found, a new one is inserted.

This function could be used to set eDeveloper-specific columns, or to update application-dependent global data. If used to update application data, you should ensure that each user has their own MAGIC.INI file. If you need to share global data and counters among multiple users, you should use standard DB tables.

For performance reasons, you may prefer to set the 'Resident MAGIC.INI' flag in the Environment table to 'Yes', in which case MAGIC.INI is updated only when exiting eDeveloper. (Refer to the discussions of the MAGIC.INI File section in the Chapter 2, Settings).

Updating a filter keyword is sent to the broker in the current session only in the case when the local engine has not yet been registered to the broker, such as when Ininput is called from the Record Prefix of a Main program of an application opened during startup. Otherwise, Ininput for a filter keyword will be effective in the next session.

Ins

Insert String

Inserts one string into another.

Syntax: Ins(target,source,position,length)

Parameters: *target*: An alpha string that represents the target string.
source: An alpha string that represents the source string.

position: A number that represents the starting position in the alpha string target.
length: A number that represents the number of characters from source that will be inserted into the alpha string target.

Returns: Alpha string.

Example: `Ins('abcde','xxx',3,1)`
returns 'abxcde'

See also: Del, Rep, Fill

InStr

In-String Search

Returns a number that represents the first position of a sub-string within an alpha string or an alpha expression.

Syntax: `InStr(string,substring)`

Parameters: *string*: An alpha string or alpha expression.
substring: An alpha string that will be the search argument in *string*

Returns: Number. 0 if not found.

Examples: `InStr('abcd','b')`
returns 2
`InStr('ABCDEF','DE')`
returns 4

InTrans

Open Transaction Test

Evaluates if a transaction is currently open.

Syntax: `InTrans ()`

Parameters: None.

Returns: Boolean

IOCopy

File Copy (On Disk)

Copies a file.

Syntax: `IOCopy(origin,target)`

Parameters: *origin*: An alpha string that represents the file specification of the existing file to be copied.

target: An alpha string that represents the name of the new file.

Returns: True if successful.

Example: IOCopy('MAGIC.FIL','MAGIC.SAV')
Copies MAGIC.FIL to a new file called MAGIC.SAV.

See also: IODel, IOExist, IORen, IOSize

Note: In each parameter the string may include a path. If the path is omitted, eDeveloper assumes the current directory.

IOCurr

Current IO

Returns a number representing the position of an IO file in the IO File repository. This is the IO file that is currently being printed to.

Syntax: IOCurr()

Parameters: None.

Returns: Returns a number representing an entry in the IO File repository.

Example: IOCurr()

Note: This function may only be used when an IO is actually in process. This means that this function can be used for Event expressions or within the Event tasks that are spawned by the Page Header and Page Footer actions. This function returns the IO number only when you have assigned page header or page footer forms to the graphic printer. Otherwise the function will return 0.

IODel

File Erase (On Disk)

Deletes a file from the disk, and returns a Boolean (True, False) indicating success or failure.

Syntax: IODel)

Parameter: *tablespec*: An alpha string that represents the file specification. The string may contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: Success returns True; failure returns False.

Example: IODel('c:\MAGIC\MAGIC.FIL')
Deletes MAGIC.FIL from the MAGIC on Drive C.

See also: IOCopy, IOExist, IORen, IOSize

Note: It is advisable to use this function instead of performing a user exit to the operating system.

IOExist

Verify Existence of File on Disk

Checks whether a specified file exists on a drive, and returns a Boolean (True, False).

Syntax: IOExist

Parameter: *tablespec*: An alpha string that represents the file specification. The string may contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: Boolean (True, False)

Example: IOExist('c:\magic\magic.ini')
returns True if the MAGIC.INI file exists in the C:\MAGIC directory.

See also: IOCopy, IODel, IORen, IOSize

IORen

Rename a File (On Disk)

Renames a file.

Syntax: IORen(origin,target)

Parameters: *origin*: An alpha string that represents the file specification of the file to be renamed.

target: An alpha string that represents the new name.

Returns: Success returns True; failure returns False

Example: IORen('MAGIC.FIL','MAGIC.SAV')
result: MAGIC.FIL is renamed MAGIC.SAV

See also: IOCopy, IOExist, IODel, IOSize

Note: In each parameter the string may include a path. If the path is omitted, eDeveloper assumes the current directory.

If the rename cannot be performed (e.g., the target file is on a different drive), the file will be copied to the target drive under the new name.

IOSize

Query File Size (on disk)

Returns the size of a file

Syntax: IOSize

Parameter: *tablespec*: An alpha string that represents the file specification. The string may contain a path. If the path is not indicated, eDeveloper assumes the current directory.

Returns: A numeric value indication the byte size of the I/O file.

Example: IOSize('c:\magic\magic.ini') returns the number of bytes contained in the C:\MAGIC\MAGIC.INI file.

See also: IOCopy, IODel, IOExist, IORen

IsComponent Checks if the executed program or handler is from a component or a host application.

Syntax: IsComponent()

Parameters: None

Returns: True when the executed program or handler is from a component. Returns False when the executed program or handler is from a host application.

IsDefault Tests if a variable's value is equal to its default value.

Syntax: IsDefault (variable)

Parameter: *variable*: value representing a variable index in the variable list

Returns: Boolean value indicating if the variable's value is (True) or is not (False) equal to the default value.

Example: IsDefault ('C'VAR)
Compares the C variable's value to the default value set in the Column Properties dialog.

IsFirstRecordCycle

This function lets you identify the first record cycle in the task. This information is useful for running logic that on the one hand is based on the already-fetched data, after the Task Prefix, and on the other hand for logic that is run only once, on entering the task.

Syntax: IsFirstRecordCycle (*task generation*)

Parameters: *task generation* – A number representing the task's hierarchy in the task tree. 0 represents the current task, 1 is

the immediate ancestor, and so on.

Returns: True when the current record cycle is the first one or when the task is executed in the Task Prefix level. This function returns False when the task is executed in the Task Suffix or Group level handlers.

Note: The various Refresh events like View Refresh, Screen Refresh, Display Refresh, or when eDeveloper refreshes the dataview after User Range or User Sort, do not affect this function. When this function returns False, it can return True only by returning to the task.

ISNULL

NULL Value Identification

Tests for the presence of a null value in a variable.

Syntax: ISNULL(variable)

Parameter: *variable*: A variable as defined in the view record. This function expects an actual variable and not a variable literal.

Returns: True if the variable contains a NULL value or False if the variable does not contain a NULL value.

Example: ISNULL(BE)
returns True if the variable designated by BE contains a NULL value.

See also: NULL

JCall

Calls an instance method.

Syntax: JCall (pseudo-reference, '[class.]method', 'signature', parameters)

Returns: The value of the method

Example: Select Virtual Ref2 (BLOB)
Update A JCreate('pkg.CLS_A', '(I)V',1000)
Update B JCall(A, 'increment', '(I)Lpkg.CLS_B;',200)
The method increment of class 'pkg.CLS_A' will be activated with 200 as the argument (I), and will return a new pseudo-reference to class pkg.CLS_B (Lpkg.CLS_B;).
You can access a **Java object** name by using the getName method, as shown below:

JCall(<Java ref var>,'getName','()Ljava/langString;')

JCallStatic

Calls a class method.

Syntax: JCallStatic('<class+method-name>', '<method-signature>', parameters)

Returns: The return value of the method.

Example: Select Virtual Ref2(BLOB)
Update B JCallStatic('pkg.A.increment_count',
'(I)Lpkg.CLS_B; ', 200)
The method 'increment_count' of class 'pkg.CLS_A' will be activated with 200 as the argument (I) and will return a new pseudo-reference to class pkg.CLS_B'space'(Lpkg.CLS_B;).

JCreate

Obtains a new instance of a Java class.

Syntax: JCreate('<class name>', '<constructor signature>', parameters)

Returns: A pseudo-reference to a new instance

Example: Selected Virtual Ref1 (BLOB)Update A JCreate('pkg.CLS_A',
'(I)V', 1000)

JException

Returns a pseudo-reference to the last exception of the current context.

Syntax: JException ()

Returns: A pseudo-reference to the last exception thrown during the last j* or ejb* function. Each call clears the exception when activated.

Example: Update B JException()

JExceptionOccurred

Informs you that the last J* or EJB* function threw an exception.

Syntax: JExceptionOccurred()

Returns: True only if the last function threw an exception.

Example: Verify JExceptionText() condition: JExceptionOccurred()

JExceptionText

Returns a text image from the last exception and an optional backtrace of the stack. This function refers to the last exception thrown during the last j* or ejb* function.

Syntax: JExceptionText(brief / [complete])
Returns: An Alpha string
Example: JExceptionText('T'LOG) - getMessage + stack trace
JExceptionText('F'LOG) - getMessage only

JExplore

Describes a class.

Syntax: JExplore ('<class name>')
Returns: An XML description of the class
Example: JExplore('pkg.CLS_A')

JGet

Retrieves the value of an instance variable.

Syntax: JGet(pseudo-reference, 'variable-name', 'variable-signature')
Returns: The variable's value
Example: JGet (A, 'a', 'I')

JGetStatic

Queries a class variable.

Syntax: JGetStatic('class name.variable name', 'variable-signature')
Returns: The variable's value
Example: JGetStatic('pkg.CLS_A.a2', 'I')

JInstanceOf

Simulates the Java's operator instanceof.

Syntax: JInstanceOf(pseufo-reference, class name)
Returns: A 'TRUE'LOG value if the object is not a NULL and can be cast to the class without generating an exception.
Example: JINSTANCEOF(A, 'javax.jms.Message').
You can use this function to check the Java reference relates to a specific class, as shown below:

JInstanceOf(<Java ref
var>,'javax.security.auth.kerberos.KerberosPrincipal')

Note: The parameter must be a BLOB.
The return-value must be alphanumeric.

JSet

Updates an instance variable.

Syntax: JSet(pseudo-reference, 'variable-name', 'variable-signature',
value)

Returns: True when the update is successful. A False value is
returned when the pseudo-reference is not part of the
current context, the variable name is not found, or the last
parameter could not be converted according to the
signature.

Example: JSet(A, 'a', 'I', B)

JSetStatic

Updates a class variable.

Syntax: JSetStatic('class name.variable name', '<variable-
signature>', value)

Returns: True when the update is successful. A False value is
returned when the variable name was not found, or if the
last parameter could not be converted according to the
signature.

Example: JSetStatic('pkg.CLS_A.a2', 'I', B)

KbGet

Returns the last pressed key or the last action performed, as defined in
the Keyboard Mapping table, in Alpha format.

Syntax: KbGet(numeric)

Parameter: *numeric* - 1 returns the Action. 0 returns the function key.

Returns: An Alpha string

Example: KbGet(0)
returns the Alpha string 'F2', if the F2 key is pressed.

KbPut

Key Entry Simulation

Simulates invoking a system action or other keyboard activity from within eDeveloper.

Syntax: KbPut(string)

Parameter: *string*: An alpha string, including a concatenation of alpha, keyboard (KBD), and action (ACT) literals.

Returns: 'TRUE' if successful

Example: KbPut('F2'KBD&'Exit'ACT&'abcd')
simulates F2; performs the Exit action as defined in the Action column of the Keyboard Mapping repository; and inputs the 'abcd' string.

Note: The KBD and ACT literals are actually shortcut notations for the strings '<keyboard value>' and '[action value]'. Because of this, multiple action and keyboard values may be concatenated. In any case, you should always use literals, as this will guarantee portability of your application to other languages.

The values specified are actually 'stuffed' at the end of an internal keyboard buffer, and are acted upon only when eDeveloper actually requires input. Because of this, you must use this function with care.

As this function actually performs an action, it is best to execute it using the Evaluate Exp.

KbPut for LCD sets is not supported. This affects NumLock, CapsLock, and ScrollLock.

See also: Literals, KbGet

LastPark

Last Parked Control

Returns the name of the control on which the user last parked in the specified task.

Syntax: LastPark (generation)

Parameter: *generation*: a number representing the task's hierarchic position in the task tree. 0 represents the current task; 1 represents the task's immediate ancestor; and so on.

Returns: The name of the control on which the user last parked.

Example: LastPark(1)
returns the control name on which the user last parked in the parent task.

See also: CTop, CLeft, CTopMDI, CWidth, CHeight, CLeftMDI

LDAPError Returns the last error message from the LDAP server. The function scope is per context.

Syntax: LDAPError()

Parameters: None

Returns: The last error message returned from the LDAP server.

LDAPGet Retrieves the user information stored in a Lightweight Directory Access Protocol (LDAP) operating system directory.

Syntax: LDAPGet(search base, search level, search filter, attribute, delimiter)

Parameters: *search base* - An Alpha variable that contains the search path in the LDAP.

search level - Specify the search level by selecting an option from the following list:

B - Base search

T - Sub-tree search

O - One-level search

search filter - An Alpha variable that contains a valid LDAP filter.

attribute - An Alpha variable that defines the information required.

delimiter - An Alpha string used to separate one result from another.

Returns: An Alpha string containing the information requested.

Examples: LDAPGet('somebase', 'B', 'objectclass=person', 'mail', '\$\$\$')
returns

dn=somename\$\$\$mail1@some.com\$\$\$

mail2@some.com\$\$\$dn=someothername\$\$\$

mail3@some.com

LDAPGet('dc=magdomain, dc=com', 'T', 'cn=Thomas', 'mgr', '\$\$\$') returns the name of Thomas' manager.

Left

Get Characters from Left of string

Returns a specified number of characters from an Alpha string, starting from the leftmost character.

Syntax: Left(string,length)

Parameters: *string*: An Alpha string.

length: The number of characters to be returned, starting from the leftmost character.

Returns: Output Alpha string.

Example: Left('abcdefg',3)
returns 'abc'

See also: Right, MID

Len

Length of String

Returns the defined length of an Alpha string.

Syntax: Len(string)

Parameter: *string*: Input Alpha string.

Returns: Length.

Example: Len ('abcdefg')
returns 7

Note: To retrieve the number of characters in an Alpha column, use Len(RTrim(*field*)). If A is a 20-character Alpha field containing the string 'John ____', where ____ implies blanks, Len (RTrim(A)) returns 4

See also: Trim, RTrim, LTrim

Level

Query Task Execution Level

Checks the execution level in a task.

Syntax: Level(generation)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

Returns: Alpha string containing:

- TP** meaning Task Prefix
- TS** meaning Task Suffix
- RP** means Record Prefix
- RM** meaning Record Main (Magic Version 8 events are imported as handlers in eDeveloper Version 9.)
- RS** means Record Suffix
- CP_<control name>** meaning Control Prefix
- CS_<control name>** meaning Control Suffix
- CV_<control name>** meaning Control Verification
- CC_<control name>** meaning Control Change
- GP** The depth of the Group Prefix (Magic Version 8 events are imported with a group counted as 2. Every group above are numbered as 3,4,5, and so on.)
- GS** The depth of the Group Suffix.
- HS_<key combination>** meaning a user-defined handler of a system event
- HI_<internal event name>** meaning a user-defined handler of an internal event
- HU_<user event name>** meaning a user-defined handler of a user-defined event
- HT_<timer setting>** meaning a user-defined handler of a defined timer
- HE_<expression>** meaning a user-defined handler of a defined expression
- HR_<error name>** meaning a user-defined handler of an error
- HX_<event name>** meaning an ActiveX handler

Examples: Level(x) where x is 0, 1, etc.
Level(1)='RP'
is True if the current level in the parent task is Record Prefix.
IF(Level(1)='', 'Parent not found', '')
outputs the 'Parent not found' message if the current task has no parent.

See also: TDepth

Note: ENDTASK: Level (1) <>'RM' is used in synchronized one-to-many tasks.

LIKE

Determines whether a given character string matches a specified pattern.

Syntax: string LIKE pattern

Parameters: string: The string with which a pattern is matched.
pattern: The pattern to be matched with the given string.

Returns: A True value if the character string matches the specified pattern. A False value if the character string does not match the specified pattern.

Examples: 'abcd' LIKE 'a*d' returns True

'abcd' LIKE 'a?d' returns False

'abc' LIKE 'a?c' returns True

Note: A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must exactly match the characters specified in the character string. Wildcard characters, however, can be matched with arbitrary fragments of the character string. Wildcard characters are:

'*' - any string of zero or more characters.

'?' - any individual character

To search for character strings that include a wildcard character, a back slash (\) must precede the wildcard character.

When performing string comparisons with LIKE, all characters in the pattern string are significant.

Line

Query Current Line Number

Returns the current line number in an output IO file.

Syntax: Line(generation,file)

Parameters: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

file: A number that represents the sequence number of the output file in the specified task.

Returns: Current line number in the output file. In text-based output, it returns the text line. In a graphical output, it returns the

current print position using the current units of measurement specified in the printed form.

Example: Line(1,1)
returns the current line number in the first output file of the immediate parent.

See also: Page

LMChkIn

License Manager Check In

Allows the developer to check-in a user instance of a specified license.

Syntax: LMChkIn (feature name)

Parameters: *feature name*: A string that specifies the name of a feature.

Returns: The function returns a numeric value indicating success or failure of the function execution as follows:
0 - Success.
8 - Failed to check-in a user instance of a selected license.

LMChkOut

License Manager Check Out

Allows the user to check out a user instance of a specified feature license.

Syntax: LMChkOut (license file name, feature name, version)

Parameters: *license file name*: A string value that indicates the license data file or the location of the license server.

feature name: A string that specifies the name of the feature.

version: A string that specifies the version of the requested feature.

Returns: The function returns a numeric value indicating success or failure of the function execution as follows:
0 - Success.
1 - Failure to initialize connection to the license server.
2 - The license file is damaged or does not exist.
3 - The value of the feature name in the function is blank.
4 - The specified feature does not exist in the license file.
5 - The expiration date of the feature has expired.
6 - The feature requested does not exist in the license file of the server.
7 - Failed to check out a user instance of a license.

LMUVStr

Returns the vendor string of the user license that was checked out by using the LMChkOut function.

The vendor string is part of the license entry in the license file that provides additional information, such as license limitations. The vendor string can be defined when using the Makekey utility.

Syntax: LMUVStr ()

Parameter: None

Returns: The vendor string of the currently checked out user license that was checked out by using the LMChkOut function.

Example: LMUVStr() returns PRODUCTS=30, TRANS=50

See also: LMVStr, LMChkIn, LMChkOut, Makekey utility

LMVStr

Returns the vendor string of the license that is checked out by the eDeveloper engine.

The vendor string is part of the license entry in the license file that provides additional information about the license, such as license limitations.

Syntax: LMVStr ()

Parameter: None

Returns: The vendor string of the current license that is checked out by the eDeveloper engine.

Example: LMVStr () executed on an eDeveloper engine that is assigned to a MGDEMO license returns the following string:
PT=MGDEMO,C=3FFFFFF,P=N,MR=500,MP=100,
MD=20,MC=8

See also: LMUVStr

Lock

Returns an evaluated expression that locks a table row or task for DM statements implemented in a deferred transaction.

Syntax: Lock (resource, timeout)

Parameters: *resource* - Any expression that returns an alpha string. Its maximum length is 128.

timeout - The maximum wait time in seconds. A negative value is equal to an infinite wait.

Returns: Number that indicates if lock was successful:
0 - Lock was successful.
1 - Resource is already locked by the same session.
2 - Resource is locked by another user and the timeout has expired.

See also: Unlock

Note: A zero length string is a valid name. In case of a recursive lock, the user must issue an Unlock command for each Lock command in order to fully unlock the resource.

LOG

Natural Logarithm
Returns the natural logarithm of a number.

Syntax: LOG(number)
Parameter: *number*: Number input value
Returns: Logarithm.
Example: LOG(2.71828)
returns 1
See also: EXP

Logical

Converts a string from a visual representation to a logical one. This function gives the developer the ability to change the way eDeveloper handles mixed strings. The possible ways of handing mixed strings are Logical and Visual. Logical is the way the engine treats the strings internally, but sometimes the strings are provided as Visual. So, if a developer would like the 'Logical' strings to be presented as Visual, they can be converted using the Visual function. Likewise, if a developer needs to handle a set of data that was provided as 'Visual' and wants to make it coherent with the way the eDeveloper engine is handling the strings, then they can be converted using the Logical function.

Syntax: Logical (string, reverse)
Parameters: string: Input alpha string.
reverse: A Boolean True or False, specifying whether to reverse the result. If the result is intended for a left-to-right environment, no reverse is needed.

Returns: If True, the string will be displayed with best results when presented from Right to Left. If False, the string will be displayed with best results when presented from left to right.

Note: This function supports Hebrew applications.

See also: Logical Operators on page 515.

Logon

Lets the user log into the current application.

Syntax: Logon (user name, password)

Parameters: *user name:* An Alpha string containing the user name.
password: An Alpha string containing the password.

Returns: Boolean indicating success or failure of the operation.

Example: Logon ('CARL',PASS321') performs similarly to the User Logon. Unlike the User logon, Logon processes the user name and password in a batch mode.

Note: When binding to an LDAP server (System Logon = LDAP), instead of using the Logon function, you can choose to define two secret names, LDAP_USER and LDAP_PASS, for the user name and password. The \$USER\$ alias in the connection string will be substituted with the value of the LDAP_USER secret name. For more information, see the LDAP Address, LDAP Connection String environment settings in Chapter 2, Settings.

LoopCounter Returns the current count of the block loop cycle.

Syntax: LoopCounter()

Parameter: None

Returns: The cycle count of the current Block Loop operation.

Example: LoopCounter() returns 7 for the seventh execution cycle of the Block Loop.

Note: The LoopCounter function is evaluated each time the Block Loop completes the Block cycle. The function returns 0 if it is evaluated outside of the Block Loop.

Lower

Converts uppercase letters to lowercase letters.

Syntax: Lower(string)
Parameter: *string*: Inputs an Alpha string or an Alpha expression
Returns: An Alpha string
Example: Lower('JOHN')
returns 'john'
See also: Upper

LTrim

Removes leading blanks from an Alpha string or an Alpha expression.

Syntax: LTrim(string)
Parameter: *string*: An input alpha string
Returns: Trimmed alpha string
Example: LTrim('John')
returns 'John'
See also: RTrim, Trim

MailBoxSet

Lets you switch to another mailbox of a currently connected IMAP mail server.

Syntax: MailBoxSet (mailbox name)
Parameter: *mailbox name* - A string representing the mailbox.
Returns: Numeric - The function returns the number of documents that exist in the specified mailbox. If the function fails to access the mailbox, it returns a negative number that represents the error code.
Example: MailBoxSet('Deleted') sets the Deleted mailbox as the active mailbox. Any other mail-related function that handles the content of the mail account relates to the Deleted mailbox.
Note: This function is a server-side function.

MailConnect

Opens a connection to a mail server.

The SMTP server is used to send mail. Some SMTP servers require a user name and password.

When this function is used to connect to a POP3 or an IMAP server, eDeveloper will connect to the user's mailbox on the server. For POP3 and IMAP servers, a user name and password are required.

When connected, the MailConnect function retrieves the new mails in the user's mailbox. New mail that the user receives after the connection has been made will not be available during that connection session but only after a new connection is made. Only one connection can be set to a send mail server and one connection to a receive mail server. If connected twice, the second connection overrides the first.

Syntax: MailConnect(type, server, user, pass)

Parameters: Type: number, 1 = SMTP server, 2 = POP3 server, 3 = IMAP server

Server: string, address of mail server

User, pass: string, user id and password of the mailbox

Returns: When you try to connect to an SMTP mail server: a 0 return value means a successful connection, a negative value means a failed connection, and each negative value represents a specific error. A successful connection to a POP3/IMAP email server is indicated by 0 or a positive value, where each positive value represents the actual number of mails in the mailbox, and each negative value represents a specific error.

Note: When you connect to a POP3 mail server, only the new messages are received. When you connect to an IMAP mail server, all existing messages, including both new and seen messages, are received.

MailDisconnect Closes a connection to an email server.

Syntax: MailDisconnect(type, delete-all)

Parameters: Type: number, 1 – mail send server, 2 – mail receive server
Delete-all: Boolean, delete all emails from mail box

Returns: 0- Success
<0 – Error code

Note: When disconnecting from a POP3 mail server and setting the Delete All parameter to Yes, only the messages that were

received by the connection to the POP 3 mail server are deleted. When disconnecting from an IMAP mail server and setting the Delete All parameter to Yes, all the existing messages, including both new and seen messages, are deleted.

MailError The MailError function translates a given mail error code that has been returned from one of the functions described above to a readable error message.

Syntax: MailError(error-code)

Parameters: error-code: number, error code returned from a function

Returns: String - the error message

MailFileSave Saves a message attachment to a file on a disk.

Syntax: MailFileSave(mail-index, file-index, save-path, overwrite)

Parameters: mail-index: number, index of mail in the mail box
file-index: number, index of attachment in the message
save-path: string, filename and path or only the path
overwrite: Boolean, overwrite file if already exists

Returns: 0 Success
<0 – Error code

Note: 1. When providing only a path name in the save-path parameter, the function saves the attachment under its defined name.
2. Setting file-index as zero instructs the function to extract all the message attachments to the given file or path name

MailLastRC Lets you retrieve the most recent error that occurred when using any of the mail functions.

Syntax: MailLastRC()

Parameter: No parameters are required.

Returns: Numeric - the function returns the number that represents the most recent mail-related error code.

Note: You can use the MailError function to translate the error code for a readable error message.

MailMsgBCC Retrieves the *BCC* string of the selected mail message.

Syntax: MailMsgBCC (index)

Parameter: *index*: The mail message index

Returns: The *BCC* string of the mail message

Example: MailMsgBCC (12) returns John Smith<johnsmith@mymail.com>, Jane Doe<janedoe@mymail.com>

MailMsgCC Returns a comma-delimited string of all the CC addresses.

Syntax: MailMsgCC (index)

Parameters: *index*: the sequential number of the message within the sequence of messages received upon connection.

Returns: A string – a comma-delimited string of all the CC addresses.

MailMsgDate Retrieves the date and time information of the selected mail message identified by its index.

Syntax: MailMsgDate(index)

Parameter: *index*: The index of the mail message.

Returns: The date and time combination of the mail message.

Example: MailMsgDate(12) returns the date and time that the mail was sent as it appeared in the mail message header - for example, 12/03/2001 12:32:05.

MailMsgDel Deletes a message from the server mailbox.

Syntax: MailMsgDel(index)

Parameters: *index*: number, index of the message in the mailbox

Returns: 0 Success
<0 – Error code

Note: The deletion of a specific message from an IMAP mail server

is committed immediately. The deletion of a specific message from a POP3 mail server is committed only upon disconnecting from the mail server.

- MailMsgFile** Returns the file name of the specific attachment of the message.
- Syntax:** MailMsgFile (index, file-index)
- Parameters:** index: the sequential number of the message within the sequence of messages received upon connection.
file-index: the sequential number of the attachment within the sequence of the message's attachments.
- Returns:** A string – the file name of the specific attachment of the message.
- MailMsgFiles** Returns the number of attachments of the message.
- Syntax:** MailMsgFiles (index)
- Parameters:** index: the sequential number of the message within the sequence of messages received upon connection.
- Returns:** A number – the number of attachments of the message.
- MailMsgFrom** Returns the address from which the message was sent.
- Syntax:** MailMsgFrom (index)
- Parameters:** index: the sequential number of the message within the sequence of messages received upon connection.
- Returns:** A string – The address from which the message was sent.
- MailMsgHeader** Retrieves the header information of the selected mail message.
- Syntax:** MailMsgHeader (index, header-key)
- Parameter:** *index*: The mail message index.
header-key: A text string that corresponds to the header information segment.
- Returns:** If the *header-key* is blank, the returned text string displays the entire header information.

If the *header key* is not blank, the text string displays the value of the header segment defined by the *header-key*.

- MailMsgId** Returns the unique id of an email message.
- Syntax:** MailMsgId(index)
- Parameters:** index: the sequential number of the message out of the messages received upon connection.
- Returns:** A string – The unique ID of the message.
- MailMsgReplyTo** Retrieves the *reply to* string of the selected mail message.
- Syntax:** MailMsgReplyTo (index)
- Parameter:** *index*: The mail message index.
- Returns:** The *reply to* information of the mail message.
- Example:** MailMsgReplyTo (12) returns John Doe<johndoe@mymail.com>
- MailMsgSubj** Returns the subject string of the message.
- Syntax:** MailMsgSubj (index)
- Parameters:** index: the sequential number of the message within the sequence of messages received upon connection.
- Returns:** A string – the subject string of the message.
- MailMsgText** Returns the body text string of the message.
- Syntax:** MailMsgText (index)
- Parameters:** index: the sequential number of the message within the sequence of messages received upon connection.
- Returns:** A string – the body text string of the message.
- MailMsgTo** Returns a comma-delimited string of all the main addresses to which the message is sent.

Syntax: MailMsgTo (index)

Parameters: index: the sequential number of the message within the sequence of messages received upon connection.

Returns: A string – a comma-delimited string of all the main addresses to which the message is sent.

MailSend

Used to send an email. It requires all the information needed to send an email.

Syntax: MailSend(from, to, cc, bcc, subject, message, file, ...)

Parameters: from: string, email address of sender
to: string, a comma delimited list of main addresses
cc: string, a comma delimited list of CC addresses
bcc: string, a comma delimited list of BCC addresses
subject: string, the title of the message
message: string, the content of the message
file, ...: string, file name and path to be used as attachments to the mail

Returns: 0 Success
<0 – Error code

MainLevel

The MainLevel function returns the main level of the task. Unlike the Level function, this function returns RM instead of the actual handler level when querying the level of an online or browser task that executes a handler of an asynchronous event. The RM string is also returned when the specified task executes a handler of an event that is raised synchronously from a control level handler or from a handler of an asynchronously raised event.

When querying the level of a batch task that executes a handler of an asynchronous event, this function returns RS instead of the actual handler level. The RS string is also returned when the specified task executes a handler of an event that was raised synchronously from a handler of an asynchronously raised event.

Syntax: MainLevel(*task generation*)

Parameters: *task generation* – A number representing the task's hierarchy position in the task tree. 0 represents the current

task, 1 is the immediate ancestor, and so on.

Returns: True for an alpha string containing:

- **RP** – Record Prefix
- **RM** – Record Main
- **RS** – Record Suffix
- **GP_<Group depth>** – Group Prefix
- **GS_<Group depth>** – Group Suffix
- **TP** – Task Prefix
- **TS** – Task Suffix

MAX

Select the Greatest of the Input Values

Returns the greatest of values with the same attributes.

Syntax: MAX(value1,value2,...valuen)

Parameters: *value1*: A number, string or logical
value2: same attribute as value 1
valuen: same attribute as value 1

Returns: Largest value.

Example: MAX(8,9.2,5,4) returns 9
MAX('ABC', 'ACD')
returns 'ACD'

Note: The number of parameters is limited to 30.
The display of the expanded expression in the Expression Rules repository will include ellipsis ('...') as the last parameter, to indicate a variable number of parameters.

See also: MIN, IF

MAXMagic

Maximize eDeveloper window

Syntax: MAXMagic()

Parameter: None

Returns: Boolean True or False indicating success or failure.

Example: MAXMagic()

MDate

Query Magic's Date

Returns the Magic date (the date entered in the Logon window).

Syntax: MDate()

Parameter: None

Returns: Date

Examples: MDate()

MDate()+5

returns a date that is five days later than the Magic date.
The numbers added or subtracted are interpreted as numbers of days.

See also: Date

Menu

Identify Menu Path

Returns the menu path that leads to the current program. The resulting alpha string contains the menu names, as well as the final menu line. Names are separated by the ';' character.

Syntax: Menu()

Parameter: None

Returns: Alpha string

Example: Menu()

returns names separated by the ';' character

Note: Returns '(null)' if program was called from TOOLKIT.

See also: Prog

MID

Substring of String

Extracts a specified number of characters (a substring) from an alpha string.

Syntax: MID(string,start,length)

Parameters: *string*: Input alpha string.

start: Number representing the starting position of the substring within the string.

length: Number of characters to be extracted; i.e., length of sub-string.

Example: MID('John',3,2) returns 'hn'

See also: Left, Right

MIN

Selects the smallest of the Input values.
Returns the smallest of a group of values with the same attribute.

Syntax: MIN(value1,value2,...,valuen)

Parameters: *value1*: A number, string or logical
value2: Same attribute as *value1*
...*valuen*: Same attribute as *value1*

Returns: Smallest value.

Examples: MIN(8,9,2,5,4) returns 2
MIN('ABC', 'ACD') returns 'ABC'

Note: The number of parameters is limited to 30.
The display of the expanded expression in the Expression Rules repository will include ellipsis ('...') as the last parameter, to indicate a variable number of parameters.

See also: MAX, IF

MINMagic

Minimize eDeveloper window

Syntax: MINMagic()

Parameter: None

Returns: Boolean True or False indicating success or failure.

Example: MINMagic()

Minute

Minutes Value of Time Value
Returns a number that represents the minutes part of a time value.

Syntax: Minute(time)

Parameter: *time*: A time value.

Returns: Number, 0-59.

Examples: Minute('12:15:00'Time) returns 15

If A contains the time value '12:15:00', Minute(A)+2 returns 17.

Note: The use of the Time literal following '12:15:00' should not be confused with the Time() function.

See also: Literals, Second, Hour, Time

MIstans

Translate String

Returns translation of a string in multi-lingual environments, based on the active entry in the Languages table.

Syntax: MIstans ('string')

Parameter: *string*: A string to be translated, up to 32000 characters long.

Returns: Alpha string, the result of the translation.

Example: MIstans ('Cancel')

MMClear

Clears marked records.

Syntax: MMClear

Parameters: None

Returns: Boolean - A True value is returned if records were marked. A False value is returned if no records were marked.

Note: If the function is evaluated during the handling of marked records, the marked records are cleared when the last marked record is completed.

MMCount

Returns the number of marked table rows in the task that is specified by the generation parameter.

Syntax: MMCount(generation)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, and so on.

Returns: The number of marked table rows in the task that is specified by the generation parameter.

MMCurr	Returns the current row in process counting from the total marked rows.
Syntax:	MMCurr(generation)
Parameters:	<i>generation</i> : A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, and so on.
Returns:	The current row in process counting from the total marked table rows.
MMStop	Multi-mark Stop
	Stops the multi-mark handler.
Syntax:	MMStop()
Parameters:	None
Returns:	When this function is evaluated, the execution of the handler for the current running record is stopped, the marked records processing is cancelled, and the engine parks on the record for which the MMStop function is evaluated.
Note:	Any operation after the evaluation of the MMStop, within the MM event, will not be executed.
MnuCheck	Check-mark Menu
	Sets a check sign next to an invoked menu entry on or off.
Syntax:	MnuCheck(Menu Name,True/False)
Parameters:	<i>Menu Name</i> is a name entry from a menu table. <i>True/False</i> is a Boolean True or False. Indicate True to set the check sign on. Indicate False to set the check sign off.
Returns:	True
Example:	MnuCheck('ENTRY','True')
Note:	The menu functions search all OS Command and Menu type menu entries in all pulldown menu structures and perform the function for all entries found. This function is not applicable to the top level pulldown menu bar.

MnuEnabl

Enables or disables a menu entry.

Syntax: MnuEnabl(Menu Name,True/False)

Parameters: *Menu Name* is a name entry from a menu table.

True/False is a Boolean True or False. Indicate True to enable a menu entry. Indicate False to disable a menu entry.

Returns: True

Example: MnuEnabl('Prog1','True'LOG)

Note: The menu functions search all OS Command and Menu type menu entries in all pulldown menu structures and perform the function for all entries found.

This function is not available for internal eDeveloper actions but only for User Actions

MnuName

This function sets the menu entry text of a selected menu.

Syntax: MnuName (entry name, entry text)

Parameters: *entry name*: The entry name of a specific menu entry as defined in the Menu repository.

entry text: The new text that replaces the menu entry's existing text.

Returns: Boolean – A True value is returned if a menu entry with the given entry name is found. If not, a False value is returned.

Example: MnuName ('entity_list', '&Customer list')

-Customer list is the text assigned to the menu entry

-The first letter is an accelerator as indicated by the ampersand (&) that precedes it.

-Entity_list is the type of menu entry.

MnuShow

Hides or shows a menu entry.

Syntax: MnuShow ('Menu Name',True/False)

Parameters: *Menu Name* is a name entry from a menu table.

True/False is a Boolean True or False value. Indicate True to show a menu entry. Indicate False to hide a menu entry.

Returns: True

Example. MnuShow('Menu','True'Logical)

Note: The menu functions searches all OS Command and Menu type menu entries in all pulldown menu structures and perform the function for all entries found.

Month

Month Value of a Date Value

Returns the month portion of a date.

Syntax: Month(*date*)

Parameter: *date*: A date value.

Returns: Number, 1-12.

Example: Month('01/28/1992'Date)
returns 1

Note: The use of the Date literal following '01/28/1992' in the above example should not be confused with the Date() function.

See also: Literals, Day, Year

MStr

Number to Alpha String Conversion

Converts a number to an eDeveloper Number with specified length in bytes.

Syntax: MStr(number,length)

Parameters: *number* - The number that will be converted.
length - The length of the Alpha string in bytes.

Returns: The Alpha string containing the number.

Example: MStr(123456,4) converts 123456 into a 4-byte alpha string.

Note: This function is especially designed to store data in arrays. The maximum precision is identified with the field storage type 'eDeveloper number'.

See also: MVal

MTblGet

Retrieves the content of a memory table as a BLOB variable.

Syntax: MTblGet(table entry, DB table name)

Parameters: *table entry* - A literal or an entry number value that corresponds to a memory table in the Table repository.
DB table name - This parameter lets you refer to a different instance of the table by its name.

Returns: The content of a memory table as a BLOB variable.

Note: If performed on a table of a database other than the Memory database, the function fails and returns a Null value.

The BLOB data item content is the memory table content that exists before the last committed transaction in the table.

The DB table name parameter, which enables you to refer to another table by its name, is effective only if the same table entry is not opened by any active task at the time the function is evaluated.

MTblSet

Creates a record in a memory table where a BLOB variable is used as the table's content.

Syntax: MTblSet(BLOB, table entry, DB table name, mode)

Parameters: *BLOB* - A BLOB value that keeps the content of the memory table.

table entry - A literal or numeric value that corresponds to a memory table in the Table repository.

DB table name - This parameter lets you refer to a different instance of the table by its name.

mode - This parameter sets the mode for updating the memory table. The mode values are:

0 - Append and abort on duplicate records. A new record from the BLOB will be appended to the existing table. When encountering a duplicate record, the entire operation is aborted and no record is set.

1 - Append and skip on duplicate records. New records from the BLOB are appended to the existing table. When

encountering a duplicate record, the record is skipped, and the next record in the BLOB is inserted.

2 - Append and overwrite duplicate records. New records from the BLOB are appended to the existing table. When encountering a duplicate record, the new record replaces the existing duplicate record.

3 - Reset existing table. The existing table is deleted and the records from the BLOB are recreated in a new table.

The mode value is set to 0 when a valid value (0,1,2,3) is not entered.

Returns: 0 when all the records are created successfully in the memory table.

The values returned when the function fails are:

-1 meaning that the Mode parameter is set to 0, but the memory table has duplicate indexes.

-2 meaning that the memory table structure does not match the table structure provided by the BLOB parameter.

-3 meaning that the table is not a memory table.

-4 meaning that the memory table has not been updated.

-5 meaning that the Mode parameter is set to 3, but the existing memory table cannot be deleted.

Note: Every DB error found during the execution of the function is skipped until the end of the table content.

The function updates the memory table as a nested transaction, regardless of the transaction setting of the current task. This means that all new records are committed in the table only when the function is completed.

The memory table structure and the BLOB parameter structure must be the same and are checked by matching their column order, column number, and column attribute. Indexes and foreign keys are not checked.

The DB Table Name parameter, which enables you to refer to another table by its name, is effective only if the same table entry is not opened by any active task at the time the function is executed.

mTime	Retrieves the time value in milliseconds from midnight to the current time.
Syntax:	mTime()
Parameters:	None
Returns:	A numeric value displaying the time in milliseconds that has elapsed from midnight to the current time.
Example:	mTime() evaluated at 02:30:21.5 AM returns the value 9021500, the number of milliseconds from midnight to 02:30:21.5 AM.
Note:	The Time data attribute does not support milliseconds.
mTStr	Converts a time value in milliseconds to a specified Alpha string picture format.
Syntax:	mTStr(time value,picture)
Parameters:	<i>time value</i> - A numeric value representing the time value in milliseconds. <i>picture</i> - The picture format for the Alpha string.
Returns:	An Alpha string of the time value in milliseconds.
Example:	mTStr(52221123,'HH:MM:SS.mmm') returns the Alpha string 14:30:21.123 mTStr(52221123,'HH:MM:SS') returns the Alpha string, 14:30:21
mTVal	Converts a time value in milliseconds from an Alpha string to a numeric value.
Syntax:	mTVal(string,picture)
Parameters:	<i>string</i> - An Alpha string containing a time value, for example, 14:30:00.123 <i>picture</i> - The picture format for the Alpha string.
Returns:	A numeric value displaying the elapsed time in milliseconds.
Example:	TVal('14:30:21.123','HH:MM:SS') returns the numeric value 52220999, the number of milliseconds from midnight to 14:30:21.123 (2:30:21.123 PM).
Note:	The Time data attribute does not support milliseconds.

MVal

Converts an eDeveloper number to a number.

Syntax: MVal(string)

Parameter: *string*: An alpha string containing a number.

Returns: The numeric value.

Example: MVal(MStr(1234,4))
returns 1234

See also: MStr

NDOW

Day of Week Number to Day Conversion

Converts the number of the day of the week (e.g. 1, 2) to the corresponding name (e.g., Sunday, Monday).

Syntax: NDOW(day of week)

Parameter: *day of week*: A number that represents the day of the week

Returns: Alpha string (containing day-of-week description)

Examples: NDOW(1) returns 'Sunday' NDOW(DOW('01/28/1992'Date))
returns 'Tuesday'

Note: If the day of week is greater than 7, it is reduced by 7; e.g.,
0,7,14,21... = Saturday.

See also: DOW, CDOW

NMonth

Month Number to Name Conversion

Converts the number of the month, 1-12, to the corresponding name, January-December.

Syntax: NMonth(month)

Parameter: *month*: A number that represents the month of the year,
1 to 12.

Returns: Alpha string.

Examples: NMonth(1) returns 'January'
NMonth(Month('01/28/1992'Date)) returns 'January'

See also: Month, CMonth, NDOW

NULL

Nullify a variable. Sets any variable to a NULL value.

Syntax: NULL()

Note: This function has no parameters and is simply used to set any variable to a NULL value. NULL support is limited to those eDeveloper Database Gateways supporting NULL values.

The developer can specify the behavior of the application when performing arithmetic that involves NULL values. This may be either to use the NULL value for calculations specified at the field or column level or the Application level, or to set the result of the calculation to NULL.

A field or column will not be set to NULL if NULL values are not allowed for it. Refer to the section on Type and Column Properties “Allow Null” settings in Chapter 4, Tables.

OEM2ANSI Converts an OEM character set to ANSI.

Syntax: OEM2ANSI (string)

Parameter: string

Returns: Parameter *string* returns a translation to ANSI.

OSEnvGet This function returns an alpha string containing the value of the operating system variable.

Syntax: OSEnvGet(*variable*)

Parameters: *variable* – A string value representing an operating system environment variable.

Returns: An alpha containing the value of the variable. If the variable is not set, an empty string is returned.

Example: OSEnvGet('PATH') reads the value of the operating system PATH environment variable.

OSEnvSet This function sets the value of an operating system environment variable. The duration of this setting is until the eDeveloper process terminates.

Syntax: OSEnvSet(*variable*, *value*)

Parameters: *variable* – A string value representing an operating system environment variable.

value – A string value representing the value that the variable is set to.

Returns: True

Example: OSEnvSet('PATH','C:\Program Files\Dev;C:\winnt\system32' sets the value of the operating system PATH environment variable to 'C:\Program Files\Dev;C:\winnt\system32'

Owner

Owner's Name

The Owner function returns the present value of the Owner parameter defined in the Environment screen as a 30 character string. This value can also be retrieved by the INIGet and Env functions. The Owner function retrieves the value from memory without any disk access, as opposed to the INIGet function. Use the Owner function for better performance.

Syntax: Owner()

Parameter: None

Returns: Alpha string (30).

Example: Owner()
returns a string with the owner's name.

See also: INIGet

Page

Current Page Number of an Output IO File

Returns the current page number in an IO output file.

Syntax: Page(generation,IOfile)

Parameters: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, etc.

IOfile: A number that represents the sequence number of the output IOfile in the specified task.

Returns: Number

Example: Page(1,1) returns the page number in the first output file of the immediate parent task.

See also: Line

ParamsPack Packs a collection of global values that have been set by the SetParam function into a BLOB variable. The BLOB variable can be transmitted to another engine context or process to recreate the global values by using the ParamsUnPack function.

Syntax: ParamsPack()

Parameters: None

Returns: A BLOB variable that stores the collection of global values.

ParamsUnPack Unpacks the global values from a BLOB variable created by using the ParamsPack function and sets them for the current engine context.

Syntax: ParamsUnPack (package)

Parameters: *package* - A BLOB variable created by the ParamsPack function.

Returns: True

Example: ParamsUnPack(A) unpacks the global values created in BLOB Variable A.

Note: If a global value with the same name already exists for the current context, eDeveloper replaces the global value with the most recently unpacked global value.

PPD Protection Device Read
Reads the protection device code programmed in the application, if a protection device is to be used.

Syntax: PPD()

Parameter: None

Example: PPD()
returns a 10 character string containing PPD code.

Pref Query Application Prefix
Returns the application prefix from the Application repository, including the path, if any.

Syntax: Pref()

Parameter: None

Returns: Alpha string containing application prefix.
Example: If the application Prefix is AP and the application resides in C:\APPL in a Windows environment, Pref returns 'C:\APPL\AP'
See also: Sys

Prog

Query Task Path
Returns the task path that leads to the current task. The resulting alpha string contains the task names, as well as the current name of the task. Names are separated by the ';' character.

Syntax: Prog()
Parameter: None
Returns: Alpha string containing execution path.
Example: Prog()
returns, for example, 'Sales Orders (by customer); Customer (Header); Order Lines'
See also: Menu

Progl dx

Program Index
Returns the current index number of the program selected in the Program repository to be used in the Call Expression operation.

Syntax: Progl dx(alpha,logical)
Parameter: *alpha*: A program name in the Program repository.
logical: False refers to the program's name. True refers to the program's public name.
Returns: The current index of the specified program in the Program repository.
Example: Progl dx('Calculate', 'FALSE'log)
or
Progl dx(A, 'TRUE'log) where A is an alpha variable.

RAND

Random Number Generator
Generates random numbers.

Syntax: RAND(startnumber)

Parameter: *startnumber*: A seed number used to define the sequence.

Returns: Random number.

Examples: If startnumber=-1, the seed is initialized randomly.
 If startnumber=n (where n is not 0 or -1), the seed is initialized to n.
 If startnumber=0, the next random number is generated based on the existing seed.
 After initializing the seed, continue with RAND(0) to generate each random number.

Range

Range Check
 Checks whether a number falls within a range, and returns a Boolean True or False.

Syntax: Range(value,lower,upper)

Parameters: *value*: The value to be checked.
lower: A value that represents the lower limit of the range.
upper: A value that represents the upper limit of the range.

Returns: 'TRUE' if Lower <= VALUE <= Upper

Example: Range(10,5,15)
 evaluates to True.

See also: MIN, MAX

Note: Like MIN or MAX, any attributes are allowed as long as they are shared by all parameters.

Rep

Replace Substring Within a String
 Replaces an alpha substring within a string with another substring.

Syntax: Rep(target,origin,pos_target,len_origin)

Parameters:

target: The target alpha string or expression where the replacement will take place.

origin: The alpha string or expression that provides the substring to be copied to target.

pos_target: The first position in the target string that will receive the substring from origin.
len_origin: The number of characters that will be moved from origin to target, starting from the leftmost character of origin.

Returns: Alpha string containing modified target string

Example: Rep('12345','abcde',3,2)
returns '12ab5'

RepStr

This function replaces any occurrences of an alpha substring with a string that has another substring.

Syntax: RepStr(*source string*, *original substring*, *new substring*)

Parameters: *source string* - The string in which the substring will be replaced.

original substring – The substring to be replaced.

new substring – Substring that will replace any occurrence of the substring being replaced.

Returns: The string that contains the modified substring.

Example: RepStr ('BB//CC//DD//EE', '//','==') returns
BB==CC==DD==EE

Note: The original substring and the new substring may differ in length. If the original substring is longer than the new substring, then the remaining spaces are truncated and the result string will be shorter than the source string.
If the original substring is shorter than the new string, then the result string will be lengthier than the source string.

ResMagic

Restore eDeveloper

Restores an eDeveloper window to normal size.

Syntax: ResMagic()

Parameter: None

Returns: Boolean True or False indicating success or failure.

Example: ResMagic()

Right

Get characters of string

Returns a specified number of characters from an alpha string, starting with the rightmost character.

Syntax: Right(string,length)

Parameters: *string*: An alpha string from which the characters will be taken.

length: The number of characters to be retrieved, starting from the right most character.

Returns: Output string

Example: Right('abcdefg',3) returns 'efg.'

Note: If string is an alpha parameter, the function refers to the full length of the parameter. If field A is a 20-character alpha parameter containing 'abcdefg', Right(A,3) returns ' ' (blanks, the last three positions of the parameter). To retrieve the characters from the actual alpha string contained in a field, use Right(RTrim(A),3)

See also: Left, MID, RTrim

RightAdd

Add Right

Assigns a Right to a user in the Security File from within an application.

Syntax: RightAdd(user,right)

Parameters: *user*: The id of the user to be assigned to the user Right given by the second parameter.

right: The right to be granted to user.

Returns: Logical True or False according to the success or failure of the operation.

Example: RightAdd('Accountant', 'Issue Invoice') will assign the Right 'Issue Invoice' to the user called Accountant.

Note: Only a user logged on as SUPERVISOR can use this function successfully.

Rights

Query Right Ownership

Queries whether user has given right.

Returns True if the user has the right.

Returns False if the user does not have the right.

Syntax: Rights('string'RIGHT)

Parameter: *string*: An alpha string, the name of a right, with the Right literal.

Returns: Boolean (True) if the user has the given right.

Example: Rights('right #4'Right) returns True if the user has right #4.

Note: Right is a literal type. Refer to the section on literals on page 511.

See also: GROUP, User

Rollback

Roll Back
Rolls a transaction back to its beginning.
Should be used if a particular condition occurs at some point within an application.

Syntax: Rollback(*logical,generation*)

Parameters: *logical*: True or False. If True, eDeveloper displays the **Confirm Transaction Rollback Request?** message, requiring user confirmation. If False, no confirmation is requested.

generation: A number representing the task, as listed below.

- 0 Rollback to the root task
- 1 Rollback to the current task
- 2 Rollback to the parent task

Example: Rollback('TRUE'LOG,0)

Returns: Logical True if transaction was rolled back, False if transaction was not aborted.

Note: While "transaction rollback" is generally thought of in terms of database failures and/or system crashes, it can also be usefully employed to handle exceptional situations related to application logic. Perhaps a batch program is processing financial transactions that update account balances in master records, and as a result of reading the current transaction record, a master record balance with a normal

range of \$5,000 to \$100,000 suddenly exceeds \$10,000,000. It may be necessary or desirable to not include this transaction in either the master account record or in related totals. This can easily be achieved through the use of the Rollback function in an Evaluate Expression operation with a condition checking the balance.

Round

Rounding

Extracts a specified part of a number and rounds the result.

Syntax: Round(number,whole,decimal)

Parameters: *number*: The number subjected to the operation.

whole: The number of digits to be extracted from the integer part of number. eDeveloper counts the digits left from the decimal separator.

decimal: The number of digits to be extracted from the decimal part of number. eDeveloper counts the digits right from the decimal separator.

Returns: Rounded number

Example: Round(345.995,2,2) returns 46.00

See also: Fix

RqCtxInf

This function returns an information string of a given context identified by the context ID.

Syntax: RqCtxInf(service/server name, context id, query password)

Parameters: service/server name - An alpha variable or constant with a service or server name.

context id - A value of the context ID.

query password - An alpha variable or constant with the password specified in the Broker initialization file.

Returns: A string with the following comma-delimited information:
Enterprise server - The host and port of the enterprise server that handles this context.
Application name - the application within which the context was opened.

- Example:** RqCtxInf('My Service','1324543108','MY PASS')
This example returns the following information string for the given context:
machine1/1607,My Application
- Note:** The RqCtxInf function is only relevant for a browser task request, because only a context opened by a browser task is saved throughout the requests. The context of other types of requests is terminated when the request is completed.

RqCtxTrm

Terminates a given context. The context is identified by its entry number as retrieved by the RqRtCtxs function.

- Syntax:** RqCtxTrm(service/server name, context entry number, supervisor password)
- Parameters:** service/server name: An alpha variable or constant with a service or server name.
Context entry number: Numeric value indicating the internal number of the context within all contexts queried using RqRtCtxs. The RqRts function must be called first.
supervisor password: An alpha variable or constant with the password specified in the Broker initialization file.
- Returns:** Boolean indicating success or failure.
- Example:** RqCtxTrm('My Service',14,'MY PASS')
This example terminates the 14th context as retrieved by the RqRtCtxs function.

RqExe

Request Executable

Requests a MRB to load a new entry from a predefined list of executables on the MRB local computer.

- Syntax:** RqExe (service/server name, executable entry name, arguments, supervisor password)
- Parameters:** *service/server name*: An alpha variable or constant from eDeveloper's servers list.
executable entry name: An alpha variable or constant identifying an executable from the predefined list of executable written as an ASCII file pointed to the

MRB_EXECUTABLES_LIST or MRB_REMOTE
EXECUTABLES_List entry in MGRB.INI.

arguments: Optional arguments for the loaded executable (e.g. /StartApplication=5).

supervisor password: An alpha variable defining the supervisor password of the Broker.

Returns: Boolean value indicating if requested executable is being loaded by the MRB. The MRB loads the executable in an asynchronous call, so the returned value indicates only if the access to the MRB was accepted and the executable could be loaded.

Example. RqExe ('Default Broker', 'Online', '/StartApplication=5', 'Secret')

RqHTTPHeader

Sets the required HTTP Header information for the returned HTTP result of a batch program.

Syntax: RqHTTPHeader(header string,[header string]...)

Parameters: *header string* - a string representing a required HTTP Header data. For example, 'Content-type: application/pdf' to indicate that the requester output data is in PDF format. A header string must follow the [Type]:[Value]

Returns: Logical - The function returns a True value if the header information is defined in the [Type]:[Value] format.

Example: To post a PDF file by using the File2Req function, you must define the HTTP Header as RqHTTPHeader('Content-type: application/pdf') within the execution of the request.

Note:

1. The HTTP Header is set for the request just as the request is completed and the output is sent back to the requester. This means that the RqHTTPHeader function can be evaluated at any point in the request task flow.
2. The HTTP Header information that was last evaluated will take effect. The next request will not be affected by the previous RqHTTPHeader evaluation.

RqLoad

Requester Load.

Provides statistical information about the load of one or all services of a single broker.

Syntax: RqLoad (service/server name, supervisor password)

Parameters: *service/server name*: An alpha variable or constant with a service or server name.

supervisor password: An alpha variable defining the supervisor password of the service broker.

Returns: String with the following comma-delimited information:
Average queue time - float numeric value with 2 digits after the decimal point. This is the average time that a user has to wait for an Enterprise Server to be assigned to them.
Total number of requests - numeric value.
Number of pending/in-queue requests - numeric value.
Number of requests in progress (received from the Enterprise Server) - numeric value.
Number of requests that were executed - numeric value.
Number of requests that failed - numeric value.

Note: Invalid service name causes the function to fail and return a blank string. If the password is not the supervisor password, then the function also fails.

RqQueDel

Requester Queue Delete

Deletes an entry in the Service Queue.

Syntax: RqQueDel (service/server name, req id, supervisor password)

Parameters: *service/server name*: An alpha variable or constant with a service or server name.

req id: An Alpha value defining the entry in the queue.

supervisor password: An alpha variable defining the supervisor password of the service broker or an empty string.

Returns: Boolean indicating success or failure.

Example: RqQueDel ('Default Service', BA, 'Secret')

removes the request identified by the value, stored in the BA variable, from the queue.

Note: Invalid service name or request id causes the function to fail and return False. If the password is an empty string, only entries submitted by the current user can be deleted. If the password is other than the current user or supervisor the function will also fail.

RqQueLst

Requester Queue List

Provides number of entries pending in the Queue.

Syntax: RqQueLst (service/server name, supervisor password)

Parameter: *service/server name:* An alpha variable or constant with a service or server name.

supervisor password: An alpha variable defining the supervisor password of the Service Broker or an empty string.

Returns: *entries* - Number of entries pending in the Queue for this service.

Example: RqQueLst('Default Service', 'Secret')

RqQuePri

Requester Queue priority

Resets the priority for a pending request in the Queue.

Syntax: RqQuePri (service/server name, req id, new priority, supervisor password)

Parameters: *service/server name:* An alpha variable or constant with a service or server name.

req id: Alpha value defining the entry in the queue.

new priority: Numeric value defining the new priority (0-9).

supervisor password: An alpha variable defining the supervisor password of the Service Broker or an empty string.

Returns: Boolean value indicating the success of the operation.

- Example:** RqQuePri('Default Service',BA, 'Secret')
The BA virtual variable temporarily stores the required ID returned by the Call Remote command.
- Note:** If the service name, req id, or priority is invalid, the function returns a value of False. If the password is invalid, the function also returns a value of False.

RqReqInf

Requester Request ID Values

Provides information about a request through a list of values that are generated from the Queue or from the Broker's history log. Executes only when RqQueLst or RqReqLst is called before.

Syntax: . RqReqInf (service/server name, entry number)

Parameters: . *service/server name* - Alpha variable or constant with a service or server name.

entry number - Numeric value specifying an index in the list of requests queried by RqQueLst or RqReqLst (one of them must be called before). This value must be between 1 to the number of entries returned by either RqQueLst or RqReqLst.

Returns: String with the following comma-delimited information:

Application Name - The name of the application of the request.

Program Name - The public name of the program in the application.

User Name - The user name to access the application.

Priority - The priority of the execution.

Submit Host - The host of the client that submitted the request.

PID - The process ID of the client that submitted the request.

Submit Time - The time when the request was submitted (string: HH:MM:SS).

Elapsed Execution Time - Number of seconds that the request executed.

Request ID - The request id associated with the request.

Status - Request Status (1 - Pending, 2 - Executing, 3 - Completed, 4 - Failed, 5 - Cleared).

Requester Return Code - Of the Request submission.

DBMS Return Code - Of the task execution.

Example: RqReqInf ('Default service', C) where the virtual numeric variable C temporarily stores an entry number calculated through the execution of the RqReqLst function.

Note: If the password is an empty string, only information about requests that were submitted from the current user can be displayed. If the service name, entry number, or supervisor password are invalid, an empty string is returned.

RqReqLst

Requester Request List

Returns the number of request entries, specified by a range of request identifications, from the broker's log.

Syntax: RqReqLst (service/server name, request id min, request id max, supervisor password)

Parameter: *service/server name:* An alpha variable or constant with a service or server name.

request id min: Identification range minimum.

request id max: Identification range maximum.

supervisor password: An alpha variable defining the supervisor password of the Service Broker or an empty string.

Returns: Number: The number of the request entries queried.

Example: RqReqLst('Web Server', BA, BB, 'Secret')

Note: If the password is an empty string, only information about requests that were submitted from the current user can be displayed. If the service name or entry number or supervisor password are invalid, an empty string is returned.

RqRtApp

Requester Supported Application Information

Returns information about one application supported by one or more runtime engines registered in the Broker.

Syntax: RqRtApp (service/server name, entry number)

Parameters: *service/server name:* An alpha variable or constant with a service or server name.
entry number: Numeric value indicating the internal number of the application, within all applications queried using RqRtApps (RqRtApps must be called before).

Returns: A string with the following comma-delimited information:
 Application Name: The application name.
 Host Name: The host where the Enterprise Server is activated.
 Port Number: The port that the Enterprise Server is listening to.

Example: RqRtApp('Data Server', BD)
 The virtual, numeric variable BD temporarily stores an application number that has been generated through the execution of the RqRtApps function.

RqRtApps

Requester Runtime applications.
 Provides the number of applications supported by one or all enterprise servers associated with a broker.

Syntax: RqRtApps (service/server name, runtime engine number, supervisor password)

Parameters: *service/server name:* An alpha or constant variable with a service or server name.
runtime engine number: Numeric value indicating the internal number of the runtime engine (if not 0), within all runtime engines associated with the service (RqRts must be called before).
supervisor password: An alpha variable with the password specified in the Magic Request Broker ini file.

Returns: *entries* - A numeric variable with the number of applications (how many) that are supported by the runtime engine, registered for this service.

Example: RqRtApps('Data Server', BA, '')
 The virtual, numeric variable BA temporarily stores runtime engine number for the number of applications that are

supported by the runtime engine. This number is returned from the execution of the RqRts function.

RqRtCtx

This function returns the information of a given context entry of a given service or server name. The RqRtCtx function should be run before using this function.

Syntax: RqRtCtx(service/server name, context entry number)

Parameters: service/server name: An alpha variable or constant with a service or server name.
Context entry number: Numeric value indicating the internal number of the context within all contexts queried using RqRtCtx. The RqRts function must be called first.

Returns: A string with the following comma-delimited information:
Context ID: The ID of the context.
Request ID: The ID of the request that is active within this context.
Program name: The name of the program that opened this context.
User name: the name of the user logged into the system within this context.
Status: The status of the context
E – Executing - the current entry is currently executing
P – Pending - the current entry is waiting for an event
T – Terminated - the current entry has terminated
Last used: the amount of time that has passed since the last time the context was active in the HH:MM:SS format.

Example: RqRtCtx('My Service',14)
This example returns the following information about the 14th entry that was queried using the 'My Service' service.
1324543108,128, Customer Entry, George, E, 00:00:00

Note: The Request ID is only available if the status of the context is Executing. Otherwise the Request ID is set to zero. For example:
2314672146,0, Customer Entry, George, P, 00:13:47

RqRtCtxs

Loads the information of all of the opened contexts of the enterprise server in which it is evaluated. The function returns the total number of

these contexts. The RqRtCtxs function needs to be evaluated prior to the RqRtCtxs function that queries a given context.

Syntax: RqRtCtxs(service/server name, runtime engine number)

Parameters: service/server name: An alpha variable or constant with a service name.
runtime engine number: Numeric value indicating the internal sequential number of the runtime engine within all the runtime engines associated with the service. The RqRts function must be called first.

Example: RqRtCtxs('My Service',3)
This example loads the information of all the contexts that are opened by Engine number 3 of the 'My Service' service, and returns the total number of opened context by the given engine.

Note: 1. Every RqRtCtxs activation clears the previous activation of the function.
2. This function is relevant for multi threaded background enterprise servers. When it is used in a non-background enterprise server, it will always return 0.

RqRtInf

Requester runtime information.

Gives information about a specific enterprise server associated with a broker.

Syntax: RqRtInf (service/server name, runtime engine number)

Parameters: *service/server name*: An alpha variable or constant with a server or a service name.
runtime engine number: Numeric value indicating the internal number of the runtime engine, within all runtime engines associated with the RqRts list (RqRts must be called before).

Returns: String with the following comma-delimited information:
Dotted Address - In the form of nnn.nnn.nnn.nnn
Host Name - The host where the runtime engine is activated.

Port Number - The port that the runtime engine is monitoring.

Process ID - The process ID of the runtime engine in the host.

Status - A numeric value indicating the following

- 1 - Available idle
- 2 - Available running
- 3 - Busy Request
- 4 - Busy Toolkit
- 5 - Not responding
- 6 - Crashed
- 7 - Query Only
- 8 - License Expired
- 9 - Connection Problem
- 14 - Shutting Down

Open Application - The name of the opened application.

Running threads - The number of running threads.

Threads peak - The maximum number of open threads that was reached.

Allowed threads - The maximum number of allowed threads. If the engine setting is zero, then it should return the maximum allowed according to the given license.

Example: *RqRtInf('Data Server', BA)*

The virtual, numeric variable BA temporarily stores the runtime engine number. This number is returned from the execution of the RqRts function

Note: In case an invalid service name or runtime engine number, for the current application, is sent, the function returns an empty string.

RqRts

Requester runtime

Gives the number of enterprise servers associated with a broker.

Syntax: RqRts (service/server name, supervisor password)

Parameters: *service/server name:* An alpha variable or constant with a service or server name.

supervisor password: An alpha variable or constant with the password specified in the broker's initialization file.

Returns: *enterprise servers:* a numeric value; the number of enterprise servers that are registered for this service, if service name was passed, or the total number of enterprise servers registered by the Broker.

Example: RqRts ('Data Server', 'Secret')

RqRtTrm

Terminates all enterprise servers associated with a requester.

Syntax: RqRtTrm(service name, entry number, supervisor password)

Parameters: *service name:* An alpha variable or constant with a service name.

entry number: Numeric values indicating the internal number of the runtime engine in the list returned by RqRts. If the value is 0, all runtime engines will be terminated.

supervisor password: An alpha variable defining the supervisor password of the Service Broker.

Returns: True when the enterprise server has been terminated. False is returned when the enterprise server has not been terminated.

Example: RqRtTrm('Data Server', BA, 'Secret')
The virtual, numeric variable BA temporarily stores the entry number. This number is returned from the execution of the RqRts function.

Note: An invalid service name or runtime engine number causes the function to fail. If another password is used, other than the supervisor password, the function fails.

RqRtTrmEx

Terminates all enterprise servers associated with a requester by a graceful timeout.

Syntax: RqRtTrmEx (service name, entry number, supervisor password, graceful timeout)

Parameters: *service name:* An alpha variable or constant with a service name.

entry number: Numeric values indicating the internal number of the runtime engine in the list returned by RqRts. If the value is 0, all runtime engines will be terminated.

supervisor password: An alpha variable defining the supervisor password of the Service Broker.

graceful timeout: The number of seconds until the server engine is terminated after the function is executed.

Returns: True when the enterprise server has been terminated. False is returned when the enterprise server has not been terminated.

Example: RqRtTrmEx ('Data Server', BA, 'Secret', 900)
The virtual, numeric variable BA temporarily stores the entry number. This number is returned from the execution of the RqRts function.

Note: An invalid service name or runtime engine number causes the function to fail. If another password is used, other than the supervisor password, the function fails. When the Graceful Timeout parameter is not set or is not a valid value, the default timeout value will be zero.

RqTrmTimeoutRetrieves the remaining number of seconds before the server termination occurs.

Syntax: RqTrmTimeout()

Parameters: None

Returns: A numeric value representing the number of seconds before the server termination occurs. The function returns 0 when there is no server termination.

Example: RqTrmTimeout returns 30, thirty seconds before the server engine is terminated.

RqStat

Request Status

Returns a simple numeric value indicating the status of a single request.

Syntax: RqStat (service/server name, request ID, supervisor password)

Parameters: *service/server name* - Alpha variable or constant with a service or server name.

request ID - The Request ID is returned as a return value from the remote call.

supervisor password - Alpha variable defining the supervisor password of the broker.

Returns: 0 - Not found
1 - Pending
2 - Executing
3 - Completed
4 - Failed
5 - Cleared

RTrim

Remove Trailing Blanks
Removes trailing blanks from an alpha string.

Syntax: RTrim(string)
Parameter: *string*: An alpha input string.
Returns: Trimmed alpha string.
Example: RTrim('John') returns 'John'
See also: LTrim, Trim

RunMode

Runmode
Returns a numeric code corresponding to the engine run mode.

Syntax: RunMode()
Parameter: None
Returns: -1 - When the main program of an application is executed for the first time on an enterprise server.

When the application is closed and another is opened for the first context.

When an application is opened for the first time under a multi-threaded background engine.

0 - When an application is run under a full runtime engine, foreground runtime engine, background runtime engine, or background generator engine.

If the function is evaluated in called programs and subtasks, RunMode returns 0.

- 1 - When an application is run under a toolkit engine and the application is initially opened in runtime mode.
- 2 - When an application is run under a toolkit engine and the application is switched to runtime mode.
- 3 - When an application is run under a toolkit engine and a program is executed directly from the toolkit mode.

Second

Seconds Value of a Time Value

Returns a number that represents the seconds part of a time value.

Syntax: Second(*time*)

Parameter: *time*: A time value or a time expression

Returns: Number, 0-59.

Examples: Second('12:15:48'Time) returns 48
If A contains the time value '12:15:48', Second(A)+2 returns 50

See also: Minute, Hour

SetBufCnvParam

Sets parameters that determine the writing and reading conversion values to and from the buffer.

Syntax: SetBufCnvParam (conversion parameter name, conversion parameter value)

Parameters: conversion parameter name - An Alpha string representing the name of the conversion parameter. The parameter values are displayed under Note.

conversion parameter value - The conversion value. Conversion values are displayed under Note.

Returns: True if successful. The function fails when the conversion parameter name or value is not valid.

Note: The conversion values are displayed in the table below.

Parameter Name	Description	Attributes	Default
----------------	-------------	------------	---------

Low-Hi	Indicates whether the numbers should be written or read in Windows standard word order, low-hi, or in UNIX's standard word order, hi-low.	Logical	True
Encoding	Encoding types for the Alpha value: 1. ANSI 2. EBCDIC 3. UNICODE	Numeric	1
Code Page	Used for EBCDIC and UNICODE encoding	Numeric	None

SetCsr

Set Cursor Shape.

Changes the cursor shape. SetCsr is activated using the Evaluate operation.

Syntax: SetCsr(number)

Parameter: *number*: A number, from 1 - 14. The expected numeric values and the cursor shape they produce are listed below:

1. Standard arrow
2. Hourglass
3. Hand
4. Standard arrow and small hourglass
5. Crosshair
6. Arrow and question mark
7. I-beam
8. Slashed circle
9. four-pointer arrow pointing north, south, east, and west.
10. Double-pointed arrow pointing northeast and southwest
11. Double-pointed arrow pointing north and south
12. Double-pointed arrow pointing northwest and southeast

- 13. Double-pointed arrow pointing west and east
- 14. Vertical Arrow.

Returns: True/False

Note: This function returns False in non-Windows platforms.
When the parameter value is <>1-14, SetCrsr returns False.

SetLang

Sets the current language to the language string parameter. The string is used as an entry point in the Language repository.

Syntax: SetLang(string)

Parameter: *String:* An alpha string specifying a language.

Returns: True when the language is set.

Example: SetLang('French')

SetParam

Sets global parameters for a single context.

Syntax: SetParam (parameter name, value)

Parameters: *parameter name:* A string representing the name of a global parameter.

value: A value that can be returned.

Returns: Boolean value indicating success or failure.

Example: SetParam('P-Employee name', BA)
The variable BA contains a string value to be passed as a parameter.

SharedValGet Retrieves a shared value according to its name. A shared value is a value stored in the memory of the eDeveloper process. Once created, this value can be retrieved by all active contexts and new contexts.

Syntax: SharedValGet(name)

Parameters: *name* - The name of the shared value.

Returns: The function returns the shared value by the defined name. The attribute of the returned value is determined by the shared value attribute as set by the SharedValSet function.

If the shared value does not exist, the function returns a Null value.

Example: SharedValGet('RATIO') retrieves the shared value name 'RATIO'.

SharedValPack

Packs a collection of shared values, set by the SharedValSet function, into a BLOB variable. The BLOB can be transmitted to another process to recreate the shared values by using the SharedValUnpack function.

Syntax: SharedValPack()

Parameters: None

Returns: A BLOB that stores a collection of shared values.

Example: SharedValPack() packs all the shared values of the current engine.

SharedValSet Creates a shared value, which is a value stored in the memory of the eDeveloper process. Once created, this value can be retrieved by all active contexts and new contexts.

Syntax: SharedValSet(name, value)

Parameters: *name* - The name of the shared value.
value - The value that can be returned.

Returns: This function always returns True.

Example: SharedValSet('RATIO', 0.34) creates a shared numeric value 0.34 with the name of RATIO.

Note: The shared value can be retrieved by using the SharedValGet function with the name of the shared value.

SharedValUnpack

Creates shared values, retrieved by the SharedValGet function, from a BLOB value created by using the SharedValPack function.

Syntax: SharedValUnpack(*package*)

Parameters: *package* – A BLOB value created by the SharedValPack function.

Returns: True when the package is not a NULL and the collection of shared values was packed using either the ParamsPack or SharedValPack functions.

Example: SharedValUnpack(A) creates shared values as created in BLOB Variable A.

SIN

Trigonometric Sine Function

Returns the sine of an angle, where the angle is expressed in radians.

Syntax: SIN(radians)

Parameter: *radians*: A number that represents the angle expressed in radians

Returns: Sine value

Example: SIN(0.7854)
returns 0.70711

See also: ASIN, ACOS, ATAN, COS, TAN

SNMPNotify

Sends an application trap message to a Network Management Station.

Syntax: SNMPNotify(message, severity)

Parameters: *message* - An Alpha string.
severity - A value from 1 (highest) to 3.

Returns: A logical value

Example: SNMPNotify('The transaction failed',1) returns False.

Note: This function should be used for application error messages.

SoundX

Compare Homonyms

Returns an alpha string that enables comparison between two alpha strings that sound alike, but are spelled differently.

Syntax: SoundX(string)

Parameter: *string*: An alpha string.

Returns: Alpha string containing 4 characters.

Example: SoundX('John Doe')=SoundX('Jhn De')
returns True

Note: The function returns a four-character alpha value. 'John Doe', 'Jhn De', 'JOhN Deo' each return 'J530'. 'Steve' returns 'S310' while 'Svte' returns 'S130'. SoundX values of strings can be determined and then used in comparisons and in condition statements. These values can also be stored in a file and indexed to allow direct access.

SplitterOffset Retrieves the current split offset percentage or unit of measurement

Syntax: SplitterOffset(mode)

Parameters: mode - The mode values are:

- **0** - Returns the split offset as a percentage
- **1** - Returns the split offset as a unit of measurement

Returns: The split offset measurement by the selected mode value.

Note: If there is no split form in the task, the function returns 0.

Stat

Check Status

Tests a task's mode (e.g., Query, Modify, etc.).

Syntax: Stat(generation, modes)

Parameters: *generation*: A number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 its immediate ancestor, etc.

modes: An alpha string that represents the tested mode or modes. For a full list of task modes see the P.

Returns: Boolean (True, False)

Example: If the current task is in Create or Modify mode,
Stat(0, 'CM' MODE)
returns True

Note: The D (Delete) status can be checked only in Record Suffix. The use of MODE following 'CM' above indicates the MODE literal.

See also: Literals

Str

Translate Number to String

Converts a number to an alpha string, according to a picture.

Syntax: Str(number,picture)

Parameters: *number*: The number to be converted into an alpha string.
picture: The format of the resulting string.

Returns: Alpha string containing converted number.

Example: Str(45.12,'###.##') returns '45.1'

Note: Because a picture is used, the function may cause rounding. Str(39.999,'###.##') returns '40.00'. For a full description of pictures, refer to the Picture discussion in the Data Items.

See also: Val

StrToken

Returns a token from a delimited string.

Syntax: StrToken(source string, token index, delimiters string)

Parameters: *source string*: Delimited alpha string with tokens.
Token Index: Requested token index (numeric).
Delimiters String: Alpha string continuing the delimiter string.

Returns: Alpha - contains the requested token or empty string when not found.

Note: The Delimiters String can have more than one character for delimiter.
 When the index is 1 and the delimiter is empty, or not found, the full source string will return.
 An empty string is returned when a delimiter was not found (empty delimiter, out of range index, or not exist delimiter), and the index is more than 1.
 Every delimiter will be counted for the index calculation (no repetition).
 A space cannot be a delimiter.

Example: StrToken(BA,2,',')
 BA - abcd,cdef,ghik,lmnp
 returns cdef

StrTokenCnt

Returns the number of existing delimited tokens in a given string.

Syntax: StrTokenCnt(source string, delimiter string)

Parameters: *source string*: A group of tokens delimited by an alpha string.
delimiter string: An alpha string containing the delimiter string.

Returns: When the delimiter is empty or was not found in the source string, the returned value is one.

Example: The expression StrTokenCnt('abcd\cdef\ghik\lmnp','\') returns 4.

Note: The delimiter string can have more than one character for delimiter.

StrTokenIdx Returns the token index in a delimited Alpha string.

Syntax: StrTokenIdx(*source string*, *token*, *delimiter*)

Parameters: *source string* - A delimited Alpha string containing at least one token.
token - The token value.
delimiter - Characters separating the tokens in an Alpha string.

Returns: A Numeric value - The position of the token in the delimited Alpha string as determined by the delimiters.

Example: StrTokenIdx('AA,BB,CC,DD','CC',';') returns 3.

Note: A space cannot be a delimiter.

Sys System Name
Returns the name of the application as it appears in the Name column of the Application repository.

Syntax: Sys()

Parameter: None

Returns: Alpha string

Example: Sys()
returns application name

See also: Pref

TAN Trigonometric Tangent Function
Returns the tangent of an angle, where the angle is expressed in radians.

Syntax: TAN(radians)
Parameter: *radians*: A number that represents the angle, as expressed in radians.
Returns: Number - The tangent value.
Example: TAN(0.7853981634)
returns 1
See also: ASIN, ACOS, ATAN, SIN, COS

TDepth Task Depth

Syntax: TDepth()
Parameter: None
Returns: A number representing the current task's depth in the task execution hierarchy.
Example: TDepth()
returns 1 if called from the root task
See also: Level

Term Query Environment Dialog for Terminal Number
Returns the terminal number as specified in the Environment table.

Syntax: Term()
Parameter: None
Returns: Number of terminal
Example: Term()
returns terminal number
Note: Term may be used for providing a unique value in a multi-user system. It is your responsibility to ensure each user is allocated their own MAGIC.INI "terminal" value. This can be done by using different MAGIC.INI files, or by using the command line override feature.
See also: INIGet, INIPut, User

Text

Tests for background Server mode in Client/Server installations of eDeveloper.

Returns True to indicate that the application is running in background Server mode, with no user interface; otherwise it returns False.

Syntax: Text()

Parameter: None

Returns: True for UNIX/iSeries and False for Windows.

Example: Text()

THIS

Directs a variable-related function or a task generation related function to the variable or task from which an event was triggered. A raised event may be handled by a handler of a higher level task. In such a case the dataview and the runtime tree below the handler's task level are not available.

In variable-related functions (VarAttr, VarCurr, VarMod, VarName, VarPrev, VarSet), the THIS() function is used to represent the index of the variable from which the event was raised. In generation-related functions (CHeight, CLeft, CLeftMDI, Counter, CTop, CTopMDI, CWidth, DbCache, EOF, EOP, LastPark, Level, Line, Page, Stat, ViewMod, WINBox, WINHWND) you use the function to represent the generation of the task from which the event was raised.

Syntax: THIS()

Parameters: None

Returns: The index of a variable from which an event was raised or the generation of the task from which the event was raised.

Example: VarCurr(THIS()) returns the current value of the variable from which the handled event was raised.
Stat(THIS(), 'C'MODE) returns True if the task from which the handled event was raised is in create mode.

Note: The THIS function cannot be applied to any function that is not listed above, and cannot be evaluated independently. The THIS function also cannot be followed by other arithmetic or conversion functions.

Time	<p>System Time Returns the system time.</p> <p>Syntax: Time()</p> <p>Parameter: None</p> <p>Returns: Current Time value</p> <p>Example: Time() returns 17:08:42 Time()+5 adds 5 seconds to the system time, returning 17:08:47</p> <p>See also: Date, MDate</p>
Translate	<p>Translates all logical names and nested logical names to their actual values. Secret names are not translated.</p> <p>Syntax: Translate(<i>string</i>)</p> <p>Parameter: <i>string</i> - An alpha value with logical names.</p> <p>Returns: The actual values represented by logical names and nested logical names.</p> <p>Example: From a string with the logical name Temp='%Driver%\temp\not found!' returns 'C:\temp\not found!'</p> <p>Note: If a logical name is not found, the part of the string remains as is.</p>
TransMode	<p>Provides information about the current active-transaction, whether it is deferred or physical.</p> <p>Syntax: TransMode()</p> <p>Returns: One of the following values: D - If the transaction is deferred. P - If the active transaction is physical. ' ' (blank) - If there is no active transaction.</p>
TreeLevel	<p>Retrieves the current level of the selected node in the data tree.</p>

Syntax: TreeLevel()
Parameters: None
Returns: The level of the selected node.

TreeNodeGoto Parks on a tree node that is identified by the tree node identifier.

Syntax: TreeNodeGoto(node id)
Parameters: *node id* - The Node ID value of the destination tree node.
Returns: True if successful.
Example: TreeNodeGoto('A001') locates the tree node with an identification value of 'A001'.
Note: eDeveloper can park on a tree node only if it is already loaded in the tree.

TreeValue Retrieves the node identification determined by the current level of the selected node.

Syntax: TreeValue(tree level)
Parameters: *tree level* - A number representing the node's hierarchic position in the data tree. 0 represents the current node, 1 its immediate ancestor, and so on.
Returns: The node identifier.

Trim Remove Blanks
Remove leading and trailing blanks from a string.

Syntax: Trim(string)
Parameters: *string*: An input Alpha string.
Returns: The edited input string, with the leading and trailing blanks removed.
Example: Trim('abc')
returns the string 'abc' free of spaces.
See also: LTrim, RTrim

TStr Translate Time to Character
Converts a time to an alpha string, according to a picture specification.

Syntax: TStr(time,picture)
Parameters: *time*: A time value to be converted.
picture: The format of the resulting character string. For a full description of pictures, refer to the Pictures section in Chapter 3, Data Items.
Returns: Alpha string containing converted time
Example: TStr ('14:30'Time,'HH:MM PM')
returns '2:30 PM'
Note: A blank picture converts using "HH:MM:SS".
See also: TVal, DStr, DVal

TVal

Alpha to Time Conversion
Converts a time value stored as an alpha string to a numeric value, according to a picture.

Syntax: TVal(string,picture)
Parameters: *string*: An alpha string that can be interpreted as a time value (e.g., '02:30:00 PM').
picture: The format for *string* in which the time is stored. This parameter helps eDeveloper read and interpret the character alpha string. For a full description of pictures refer to the Pictures section in Chapter 3, Data Items.
Returns: Time value
Example: TVal('02:30:00 PM','HH:MM:SS PM')
returns 14:30:00
Note: A blank picture interprets the string as 'HH:MM:SS'.
See also: TStr, DVal, DStr

UDF

User Defined Function
User functions written in the third generation programming languages, like C or Pascal, can be called as functions within eDeveloper expressions (cdeccl).

Syntax: UDF('module_name.function_name',[parameters])
Parameters: *module_name*: The external module or program containing the user defined function.

function_name: The named function.

parameters: Up to 30 parameters, as required by the user defined function, separated by commas.

Returns: The result of the external third generation function.

Note: UDF uses a mechanism similar to that of User Procedures. See the Installation and Platform Information literature for your platform.

It is possible to activate a User Procedure (UP) or a User Defined Function (UDF) on a host machine when using the eDeveloper Client/Server architecture. To activate the remote UP or UDF, the server name should be part of the module name, as in (myserver)myproc.add.

UDFF

User Defined Function called by the fastcall convention.

User functions written in third generation programming languages, like C or Pascal, can be called as functions by the fastcall convention.

Syntax: UDFF('module_name.function_name'[,parameters])

Parameters: *module_name*: The external module or program containing the user defined function.

function_name: The named function.

parameters: Up to 30 parameters, as required by the user defined function, separated by commas.

Returns: The result of the external third generation function.

Note: UDFF uses a mechanism similar to that of User Procedures. See the Installation and Platform Information literature for your platform.

It is possible to activate a User Procedure (UP) or a User Defined Function (UDF) on a host machine when using the eDeveloper Client/Server architecture. To activate the remote UP or UDFF, the server name should be part of the module name, as in (myserver)myproc.add.

UDFS

User-defined function by the stdcall convention

User functions written in third generation programming languages (like C or Pascal) can be called as functions by the stdcall convention.

Syntax: UDFS(module_name.function_name[,parameters])

Parameters: *module_name*: The external module or program containing the user defined function.

function_name: The named function.

parameters: Up to 30 parameters, as required by the user defined function, separated by commas.

Returns: The result of the external 3GL function.

Note: UDFS uses a mechanism similar to that of User Procedures. See the Installation and Platform Information literature for your platform.

It is possible to activate a User Procedure (UP) or a User Defined Function (UDF) on a host machine when using the eDeveloper Client/Server architecture. To activate the remote UP or UDFS, the server name should be part of the module name, as in (myserver)myproc.add.

Unlock

Unlock Row

Returns an evaluated expression that unlocks a table row or task that is locked.

Syntax: Unlock(resource)

Parameter: *resource*: Any expression that returns an alpha string. Its maximum length is 128.

Returns: Number that indicates if lock was successful:

0 - Resource is not locked by the same session or has not been locked at all.

1 - Unlock was successful.

See also: Lock

Note: Longer expressions will be truncated. Zero length is a valid name. A resource may only be unlocked by the same document that locked it.

Upper

Switch Lower Case to Upper Case
Converts lowercase letters to uppercase letters.

Syntax: Upper(string)
Parameter: *string*: An alpha string to convert.
Returns: Alpha string converted to upper case.
Example: Upper('john')
returns ('JOHN')
See also: Lower

User

Query User Data
Returns user data as specified in the User IDs repository.

Syntax: User(number)
Parameter: *number*: 0, 1, or 2.
User(0) returns the User ID.
User(1) returns the User Name.
User(2) returns the User Information variable.
Returns: Alpha string.
Example: User(1)
returns the User Name of the current user.
Note: The User function in previous releases had no parameters and returned the User ID. In accordance with changes to the authorization system from Magic Version 5.5, the function has been revised. The import facility now converts the User function in existing applications to User(0).
See also: INIGet, INIPut

UserAdd

User Added to Security File
Add a user record into the Security file from within an application.

Syntax: UserAdd(user,name,password,info)
Parameters: *user*: The user identification number to insert into the security file.
name: The name of the new user.
password: The user password.

info: The user information parameter.

Returns: Logical True or False in the User Information variable.

Example: UserAdd('John','John Doe','xyz','no info') will insert a user called John to the Security file, with the full name John Doe, the password xyz and the string 'no info' in the User Information variable.

Note: Only a supervisor user can use the function successfully.

UserDel

Lets the supervisor delete a user identification in a security file from within an eDeveloper application.

Syntax: UserDel(*user identification name*)

Parameter: *user identification name* - The user identification number in the security file to be deleted.

Returns: True or False, depending on whether the user entry was deleted or not.

Example: UserDelete('John') deletes a user identification called John from the security file.

Note: Only a supervisor or a user assigned to a supervisor group can use the UserDel function.

UTF8FromAnsi

Converts data encoded in ANSI to UTF8

Syntax: UTF8FromAnsi(string value)

Parameters: *string value* - The task variable that will be converted. The variable attribute can be Alpha, Memo, or BLOB RTF.

Returns: A BLOB RTF encoded in UTF8.

Example: UTF8FromAnsi(A), where A is a variable in the current task, returns a BLOB containing the value encoded in UTF8.

UTF8FromAnsi('Hello World') returns a BLOB containing 'Hello World' encoded in UTF8.

UTF8ToAnsi

Converts data encoded in UTF8 to an ANSI string. This function uses code pages defined in the CodePage function. If you are not using the CodePage function, the default OS code page is used in a similar way as with Java functions.

Syntax: UTF8ToAnsi(string value)

Parameters: *string value* - The task variable that will be converted. The variable attribute can be an Alpha, Memo, or BLOB attribute.

Returns: A BLOB RTF encoded in ANSI. A Null is returned when using an invalid code page.

Example: Where A is a BLOB variable in the current task, UTF8ToAnsi (A) returns a BLOB containing the value of A encoded in ANSI.

Val

Alpha to Numeric Conversion
Converts an alpha string to a numeric value, according to a picture.

Syntax: Val(string,picture)

Parameters: *string*: The alpha string to be converted to a numeric value.
picture: The format in which the number is stored in the string. For a full description of pictures, refer to Pictures section in the Chapter 3, Data Items.

Returns: Number values

Example: Val('45.12','###.##')
returns 45.1

Note: A blank picture may be specified for converting standard numbers.

See also: Str

VarAttr

Variable Attribute Retrieval
Provides a variable's attribute.

Syntax: VarAttr(variable)

Parameter: *variable*: Value representing a variable index within the Variable list.

Returns: A string containing the variable's attribute.

Examples: VarAttr('BE'VAR)

Note: VarAttr is helpful when implementing a generic cut and paste mechanism using Application events.

See also: . VarCurr, VarInp, VarMod, VarName, VarPrev

VarCurr

Current Retrieval

VarCurr retrieves the current value of a variable, based on a dynamic value representing a variable index within the Variable list

Syntax: VarCurr(variable)

Parameter: *variable*: Value representing a variable index within the Variable list.

Returns: Current value of requested variable, in the variable's attribute

Examples: VarCurr('BE'VAR)
returns the current value of the variable BE
VarCurr('CR'VAR + 7)
returns the current value of the variable CY

Note: The variable parameter should use the VAR literal so that eDeveloper can automatically reorder this parameter. This function may be used for implementing arrays, by adding to or subtracting from the base variable.

See also: Literals, VarPrev, VarMod

VarCurrN

Returns the current value of a variable according to the variable's name.

Syntax: VarCurrN(variable name)

Parameter: Variable name: A string representing a variable's name.

Returns: Current value of requested variable. The attribute of the returned value is the referred variable's attribute.

Examples: VarCurrN('Customer ID') returns the current value of the variable named 'Customer ID'.

Note: If the variable name is not found, the function returns NULL. When the given variable name exists more than once, the function relates to the lowest variable in the task tree.

VarDbName

Queries a given dataview to retrieve the physical definition of each variable.

Syntax: VarDbName(variable index)

Parameter: variable index – The value that represents a variable index

in the Variable list.

Returns: String – If the variable is a real field of an SQL table, the function returns the table owner, the table's physical name, and the column's physical name of the defined variable, delimited by a period. If the variable is not a real field of an SQL table, the function returns an empty string.

Examples: VarDbName('C'VAR) returns the CUSTOMER_ID, which is the name of the variable as defined in the database.

Note: VarDbName is relevant only for real fields of SQL tables. The function returns a blank string when evaluating real variables from a Btrieve or Memory table, or a local variable that is not part of a Direct SQL statement.

VariantAttr Retrieves the eDeveloper attribute corresponding to the variant data type.

Syntax: VariantAttr(variant value)

Parameters: variant value - A BLOB value storing the variant data type.

Returns: A single character for the variant data type, as shown below:

A - Alpha

N - Numeric

L - Logical

D - Date

B - BLOB

V - Vector

0 - No value

Example: VariantAttr(B) returns A when the value of the variant is a VT_BSTR data type.

VariantAttr(C) returns N when the value of the variant is a VT_I4, or VT_R8 data type.

VariantAttr(D) returns 0 when the value of the variant is a non-supported data type.

VariantCreate Creates a variant according to the provided data.

Syntax: VariantCreate(VT type, value, optional time value)

Parameters: VT type - A numeric value representing the variant type.

0	VT_EMPTY
1	VT_NULL
2	VT_I2
3	VT_I4
4	VT_R4
5	VT_R8
6	VT_CY
7	VT_DATE
8	VT_BSTR
9	VT_DISPATCH
10	VT_ERROR
11	VT_BOOL
12	VT_VARIANT
13	VT_UNKNOWN
14	VT_DECIMAL
16	VT_I1
17	VT_UI1
18	VT_UI2
19	VT_UI4
20	VT_I8
21	VT_UI8

22	VT_INT
23	VT_UINT
24	VT_VOID
25	VT_HRESULT
26	VT_PTR
27	VT_SAFEARRAY
28	VT_CARRAY
29	VT_USERDEFINED
30	VT_LPSTR
31	VT_LPWSTR
36	VT_RECORD
64	VT_FILETIME
65	VT_BLOB
66	VT_STREAM
67	VT_STORAGE
68	VT_STREAMED_OBJECT
69	VT_STORED_OBJECT
70	VT_BLOB_OBJECT
71	VT_CF
72	VT_CLSID
4095	VT_BSTR_BLOB
4096	VT_VECTOR
8192	VT_ARRAY
16834	VT_BYREF

32768 VT_RESERVED

value - The value selected for the variant.

optional time value - Relevant when the eDeveloper attribute parameter is set to D. The Date data type of a variant stores both date and time values. Set the date in the value parameter and the time in the optional time value parameter.

Returns: A BLOB value representing the created variant. You can use the returned value to update a BLOB field, keep the variant value, or pass the BLOB value directly to an external object.

Example: VariantCreate (8,'Hello') returns a variant set by the string value as a VT_BSTR data type.

VariantCreate (7,Date(),Time()) returns a variant set with the Date and Time values as VT_DATE data type.

VariantGet Retrieves the value of a specified variant data type.

Syntax: VariantGet (variant value, attribute)

Parameters: variant value - A BLOB value storing the variant data type.

attribute - Enter a character below to indicate the eDeveloper data attribute:

A - Alpha

N - Numeric

L - Logical

D - Date

B - BLOB

Returns: The value as specified by the variant data type and the eDeveloper data attribute.

Example: VariantGet (B,'A') returns an Alpha value.

VariantGet (C,'D') returns a Date value.

VariantType Retrieves the variant data type's storage type identifier.

Syntax: VariantType(variant value)

Parameters: variant value - A BLOB value storing the variant data type.

Returns: The variant storage type identifier. If the BLOB value is not a valid variant, the function returns Null.

Example: VariantType(B) returns 20 when the data type is VT_I8.

Note: The Variant Storage Type table is shown below.

Table 1 - Basic Types

The following table shows the VARTYPE values that indicate valid Variant types.

VARTYPE Value	Variant Storage Type	Description	Range
0	VT_EMPTY	No value was specified	
1	VT_NULL	SQL-style Null	
2	VT_I2	Signed 2-byte integer	-32,768 to 32,767
3	VT_I4	Signed 4-byte integer	-2,147,483,648 to 2,147,483,647
4	VT_R4	Signed 4-byte real	1.1E -38 to 3.4E +38 (7 digits)
5	VT_R8	Signed 8-byte real	2.2E -308 to 1.7 E +308 (15 digits)
6	VT_CY	Currency	
7	VT_DATE	Date	
8	VT_BSTR	Automation string	
9	VT_DISPATCH	A pointer to an object that implements IDispatch	
10	VT_ERROR	SCODE	

VARTYPE Value	Variant Storage Type	Description	Range
11	VT_BOOL	Boolean	
12	VT_VARIANT		
13	VT_UNKNOWN	A pointer to an object that implements IUnknown	
14	VT_DECIMAL	Decimal	
16	VT_I1	1-byte character	-128 to 127
17	VT_UI1	Unsigned 1-byte character	0 to 255
18	VT_UI2	Unsigned 2-byte integer	0 to 65,535
19	VT_UI4	Unsigned 4-byte integer	0 to 4,294,967,295
22	VT_INT	Signed machine integer	
23	VT_UINT	Unsigned machine integer	

Example: A 4-byte integer value, which is indicated by VT_I4, is represented by the value of 3.

When the variant type is a reference or an array, the following values should be added to the value that represents the actual storage.

Table 2 - Arrays and References

The following values are used for Bit-Wise Logical OR operations. These values should be used in addition to the values in Table 1. They are not valid alone.

VARTYPE value	Enumeration Symbol	Description
4096	VT_BYREF	A reference to data type
8192	VT_ARRAY	An array of data type

Note:

The symbol | is used in the documentation to indicate a Bit-Wise Logical OR operation and is common when indicating variants, such as VT_I2|VT_BYREF. When you use the values from Table 2, you should use the arithmetical ADD operation, instead of the symbol |.

Example of *by reference* storage

16387 represents a reference that is a signed 4-byte integer, indicated by VT_I4|VT_BYREF (16387 is a result of 3 + 16384).

Example of an array

8195 represents an array that is a signed 4-byte integer, indicated by VT_I4|VT_ARRAY (8195 is a result of 3 + 8192).

Exceptions

The exceptions are:

1. VT_VARIANT cannot be used alone and must be used with a value from Table 2
2. The following combinations are not valid:
 - VT_NULL|VT_BYREF
 - VT_EMPTY|VT_BYREF
 - VT_NULL|VT_ARRAY
 - VT_EMPTY|VT_ARRAY

VarIndex

Returns the index of a variable according to the variable's name. The VarIndex function can be used in other VAR functions such as VarMod, VarAttr, VarPrev, VarSet, VarPic as the variable index.

Syntax: VarIndex(variable name)

Parameter: variable name: A string representing a variable's name.

Returns: The index of a requested variable.

Examples: VarAttr(VarIndex('Customer ID')) returns the attribute of the variable named 'Customer ID'.

Note:

1. If the variable name is not found, this function returns Zero.
2. When the given variable name exists more than once, the function relates to the lowest variable in the task tree.

VarInp

Last variable where the insertion point is parked.

Identifies the last variable where the insertion point has parked, allowing input.

Syntax: VarInp(generation)

Parameter: *generation*: A number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 its immediate ancestor, etc.

Returns: The number of the variable in the specified task where the insertion point has parked. Variable numbers start with A for the first variable of the root task, and are incremented sequentially for every new variable selected. Subtask variables continue the sequence of their ancestor task.

Example: Task 1 contains two select field operations. Its subtask contains two more select fields. The variable list for this program appears as follows:

```
A VAR 1
B VAR 2
-----
C VAR 3
D VAR 4
```


The insertion point is parked on variable B. The function VarInp(1) evaluated at the subtask level will return B.

Note: The numbering of variables in sibling tasks starts from the last variable of their common parent. Therefore the variable identifications of both subtask's variables are the same for the first group of variables (group size is the minimum number of variables between the two tasks).

See also: VarAttr, VarCurr, VarMod, VarName, VarPrev

VarMod

Variable Modification Check

Checks whether the contents of a variable were changed since the dataview record was fetched.

Syntax: VarMod(variable)

Parameter: *variable*: Value representing a variable index.

Returns: True if modified

Example: VarMod('A' VAR) evaluates to True if the contents of variable A were changed since the dataview record was fetched.

VarName

Variable Origin and Description

Provides a variable's origin and description.

Syntax: VarName(variable)

Parameter: *variable*: Value representing a variable's index.

Returns: A string containing the table name where the variable originates, concatenated with '.' and the field description of the variable in that table. If the variable is a virtual one, then the table name would indicate 'Virtual'.

Examples: If variable A is 'Customer Number' from the Customer table the function VarName('A'VAR) returns 'Customer.Customer Number'

Note: The variable parameter should use the VAR literal in order that eDeveloper may automatically reorder this parameter.

See also: VarAttr, VarCurr, VarInp, VarMod, VarPrev

VarPic

String value for a picture field

Returns a string value that represents a picture field.

Syntax: VarPic (variable, mode)

Parameter: variable - '#' variable or a numeric value
mode - the function mode

Returns: Mode=0 returns the picture field as defined in the field definition.

Mode=1 returns the picture field as defined by the control assigned to the field. Note that only control assigned to Class 0 forms can be referred to by function mode.

Note: If the control does not have a picture property, the string returns the field definition of the picture, as if Mode=0. If the variable is not placed on the form, the string returns the field definition of the picture, as if Mode=0.

Mode values other than 0 or 1 are considered as Mode=0.

VarPrev

Variable Retrieval - Previous Value

Retrieves the original value of a variable, based on a dynamic value representing a variable index within the Variable list.

Syntax: VarPrev(variable)

Parameter: *variable*: Value representing a variable index within the Variable list.

Returns: Original value of requested variable, in the variable's attribute

Examples: VarPrev('BE'VAR)
returns the original value of variable BE
VarPrev('CR'VAR + 7)
returns the original value of variable CY
VarPrev('W'VAR) <> W
checks if the variable has been changed - similar to VarMod('W'VAR)

Note: The variable parameter should use the VAR literal to enable eDeveloper to automatically reorder this parameter.

This function should be used to retrieve the value of a variable as it was when the Dataview record was fetched.

VarPrev should not be used during the initializing stage of a record since eDeveloper does not have the necessary information ready for the function. Therefore you should refrain from using this function in INIT expressions. In any case, during this stage of the engine operation, the previous value of a variable is equal to the current value.

See also: Literals, VarCurr, VarMod

VarSet

Sets a named variable to a specific value.

Syntax: VarSet(variable,value)

Parameters: *variable*: A value representing a variable index. It should be used with the VAR literal.

value: The value to which the variable is to be set.

Returns: True or False. If a Null value or a non-existent variable is entered, the function returns a False value.

Note: The VarSet function is Boolean and always returns the value 'TRUE'LOG. The update operation is always Normal (it cannot be specified as Incremental).

Recomputation is performed after the VarSet function terminates.

With the availability of both VarCurr and VarSet, you can implement certain categories of database applications whose natural database organization would normally be arrays. As an example, consider an integer numeric array of size 1,000 rows by 100 columns, stored as 1,000 records of 100 variables each. In the application it is required that value_a be added to the current value in a given cell of the array identified by row and column. Define a batch subtask whose Min/Max Range limits its dataview to the record identified by row and, in the Record Suffix, "update" the field identified by column. Without VarCurr and VarSet it would require over 200 Update and other operations. These can be replaced with one Evaluate Expression operation using the following expression:

VarSet('A'VAR + column-1, VarCurr('A'VAR + column-1)+value_a)

where it is assumed that 'A'VAR is the eDeveloper token identifier of the first variable in the record, and the values of column begin from the value '1'.

VecCellAttr	Returns the vector's cell attribute. Syntax: VecCellAttr (vector) Parameters: <i>vector</i> - The Vector field Returns: An Alpha string containing the cell's attribute. A blank is returned when the Vector field has not been updated.
VecGet	Returns the value of a specified cell. Syntax: VecGet(Vector, Cell Index) Parameters: <i>Vector</i> - The vector value. <i>Cell Index</i> - The index of the cell that is retrieved. Returns: The value as specified by the cell model attribute. A Null value is returned if the field is not a vector or if the cell index is negative. When the cell has not been updated, the function returns the default value of the cell model.
VecSet	Updates the value of a selected cell with a given vector. Syntax: VecSet(Vector Field Reference, Cell Index, Value) Parameters: <i>Vector Field Reference</i> - A numeric value representing a vector variable index within the Variable list. <i>Cell Index</i> - The index of the retrieved cell. <i>Value</i> - Depends on the data attribute stored in the cell vector. Returns: True when successful. False when the field reference is not a vector field, the value parameter is not the same data attribute value as stored in the cell vector, or a negative index value is provided.
VecSize	Returns the number of cells for the given vector.

Syntax: VecSize (Vector)
Parameters: *Vector* - A vector variable.
Returns: A numeric value. -1 when an attribute is not a vector, or when the vector is a Null or a BLOB that is a Null (not updated).

ViewMod

Dataview Record Modification Check
Checks whether the current dataview record was changed since fetched.

Syntax: ViewMod(generation)
Parameter: *generation*: A number representing the task's hierarchic position in the task tree. 0 represents the current task, 1 its immediate ancestor, etc.
Returns: True if modified.
Example: Introduced as a condition to a Verify Exp operation, the expression:
ViewMod(0)
displays the Verify Exp message if the current dataview record in the current task was changed since fetched.
See also: VarMod

Visual

This function supports Hebrew applications. This function gives the developer the ability to change the way eDeveloper handles mixed strings. The possible ways of handing mixed strings are Logical and Visual. Logical is the way the engine treats the strings internally, but sometimes the strings are provided as Visual. So, if a developer wants the 'Logical' strings to be presented as Visual, they can be converted using the Visual function. Likewise, if a developer needs to handle a set of data that was provided as 'Visual' and wants to make it coherent with the way the eDeveloper engine is handling the strings, then they can be converted using the Logical function.

Syntax: Visual (string, True/False)
Parameters: *string* - The string of which to perform the conversion.
True/False is a Boolean True or False.
Returns: If True, the string will be displayed with best results when presented from Right to Left. If False, the string will be

displayed with best results when presented from Left to Right.

WebRef

Converts a text string to an input field on an HTML web page for use in Web Online applications.

Syntax: WebRef (string)

Parameter: *string*: The name of an input field in string format.

Returns: The string with % characters before and after.

Examples:

1. The expression WebRef('CustomerName') returns '%CustomerName%'
2. The expression WebRef('VarLine'&Str(Counter(0),'1')), where Counter(0)=2, returns '%VarLine2%'
3. The expression 'Thank you'&WebRef(BC)& 'for purchasing our product', where the BC variable contains the string 'CustomerName', returns 'Thank you %CustomerName% for purchasing our product'

WINBox

Window Box.

Returns a Window dimension, X position, Y position, Width or Height

Syntax: WINBox(numeric, alpha)

Parameters: *numeric*: A number representing a generation.

0 = the current generation, 1 = the parent generation.

alpha: A representation of the dimension to be returned.

Use X to return the X position, Y to return the Y position, W to return the width, and H to return the height.

Returns: Numeric value of the dimension requested.

Example: WINBox (0, 'W')

WinHelp

This function opens a specified Help file and performs a selected command.

Syntax: WinHelp(help file, command, help key)

Parameters: *help file* - A string specifying the Help file name and path.

command - A number specifying the Help command that is performed when the Help file is opened.

The numbers below represent the supported commands:

- 1 - Context
- 2 - Contents
- 3 - SetContents
- 4 - ContextPopup
- 5 - Key
- 6 - Command
- 7 - ForceFile
- 8 - HelpOnHelp
- 9 - Quit

For more information about the supported commands, See “Windows WinHelp Connections” on page 978.

help key - A string value providing additional information required for some of the commands.

For more information about the Help key, see page page 978.

You should specify a blank string for commands that do not require Help key information.

Returns: True when the Help file successfully opens. False is returned when the function fails.

Example: WinHelp(`Support\MGHELPW.CHM`,2,`) opens the contents page of the eDeveloper Help file.
WinHelp(`Support\MGHELPW.CHM`,1,`1001`) opens an eDeveloper Help file and displays the Help topic assigned to context number 1001.

Note: SetContents, ContextPopup, Key, Command, ForceFile, and HelpOnHelp commands are not supported for CHM Help files.

WINHWND

eDeveloper window's window handle
Returns a Window's window handle (HWND) for an eDeveloper window. This can be used in a user-defined function that requires an HWND.

Syntax: *WINHWND (numeric)*

Parameter: *numeric*: Value representing an eDeveloper generation, where 0 = the current task, 1 = the parent task, and so on.

Returns: Value that represents the window handle, HWND.

Example: WINHWND(1)

WSAttachmentAdd

Web Service (WS) DIME addresses the difficulties involved in embedding binary data, such as video, graphics, and sound into XML documents. eDeveloper supports DIME attachments when executed in RPC or DOC/Literal Web service provider.

A provider Web service program can retrieve attachments from the SOAP request or to add attachments to the SOAP response using the WSAttachmentGet and WSAttachmentAdd functions.

This function attaches a BLOB variable to a Web service message.

Syntax: WSAttachmentAdd (*Attachment*)

Parameters: *Attachment* - The BLOB variable to be attached to the message.

Returns: An alphanumeric Universal Unique Identifier value (UUID) automatically generated by the outgoing attachment, which can be used as a reference in the returned XML document.

The function returns a Null value when the attached BLOB variable is empty or the WSAttachmentAdd function is not activated during a provider program execution.

Example: WSAttachmentAdd (A),

where A is a BLOB variable, retrieves a UUID that can be sent in the XML message.

Note: Outgoing attachments are cleared from memory after the Task Suffix operations have been executed in the Web service provider program.

WSAttachmentGet

Retrieves a Web Service (WS) DIME attachment received by a Web service request.

Syntax: WSAttachmentGet (*Identifier*)

Parameters: *Identifier* - Select an identifier type from the options below:

1. Universal Unique Identifier (UUID) - An alphanumeric value that uniquely identifies the WS DIME attachment. In some cases the sender can add the UDDI into the XML message, but this is not mandatory.
2. Index - Numeric value. When eDeveloper receives several attachments, the function retrieves each WS DIME attachment by its index number in the message, starting from 1.

Returns: A BLOB containing the incoming Web Service (WS) DIME attachment.

A Null value is returned when the WS DIME attachment is not found by the UUID or Index, or the function had not been activated during a provider program execution.

Example:

1. WSAttachmentGet (1), returns the first WS DIME attachment from the message.
2. WSAttachmentGet ('F2DA3C9C-74D3-4A46-B925-B150D62D9483'), returns the identified WS DIME attachment.

Note: In runtime, incoming attachments are available during the web service provider program by using the following syntax: WSAttachmentGet (*uuid*) or WSAttachmentGet (*index*). For indexes, the incoming attachments are added in their order of appearance in the DIME message.

Incoming attachments are cleared from memory after the Task Suffix operations have been executed in the Web service provider program.

XMLBlobGet Returns the value of an XML element or an XML attribute according to its element path.

Syntax: XMLBlobGet(*generation, file, element path, attribute name*)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.
file: A number that represents the sequence number of the I/O file in the current task.

element path: A string that represents the path of an XML element. The syntax is:

element name[[**index**]] [**.element name**[[**index**]]...]

where:

element name is the name of an XML element. You can assign an alias to the element name. The alias is separated from the element name by a colon symbol (:), for example, `al1:My_Element`.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

attribute name: The name of the XML attribute.

Returns: BLOB. If the XML parsing process is completed successfully, the requested element will exist and will return a value. If the XML parsing process was not successful, an empty string is returned.

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <name>John Smith</name>
    <address>
      <street>Somewhere</street>
      <city>Nowhere</city>
    </address>
  </issued_by>
  <item cat_num="2145451544">
    <price>99.99</price>
    <amount>2</amount>
  </item>
  <item cat_num="1384325456">
    <price>19.99</price>
    <amount>10</amount>
  </item>
</order>
```

Given that the XML above is saved into the file C:\myxml.xml, and that the current task first I/O file points to an XML file, the following expressions apply:

XMLBlobGet (0, 1, 'order.issued_by', 'name') returns John Smith.

XMLBlobGet (0, 1, 'order.send_to', 'name') returns an empty string because there is no *send_to* element in the order.

XMLBlobGet (0, 1, 'order.item[2]', 'price') returns 19.99.

XMLBlobGet (0, 1, 'order.item[5]', 'price') returns an empty string because there are only 2 items in the order.

XMLBlobGet (0, 1, 'order', 'id') returns 123.

XMLBlobGet (0, 1, 'order.issued_by', 'date') returns an empty string because there is no *date* attribute in the *issued_by* element.

XMLBlobGet (0, 1, 'order.issued_by', 'name') returns an empty string because *name* is a child element of the *issued_by* element but not an attribute of it.

Note: When an XML file is encoded in a different code page than the code page of the operating system, you can change the code page by using the CodePage function. The code page is assigned per context.

XMLCnt

Returns the number of occurrences of an XML element or an XML attribute according to its path.

Syntax: XMLCnt (generation, file, element path, attribute name)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, and so on.

file: A number that represents the sequence number of the I/O file in the current task.

element path: A string that represents the path of an XML element. The syntax is:

element name[[index]] [.element name[[index]]...]

where:

element name is the name of an XML element. You can

assign an alias to the element name. The alias is separated from the element name by a colon symbol (:), for example, `al1:My_Element`.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

attribute name: The name of an XML attribute that is the attribute of the multi-occurrence element name. Its value is compared to the Value parameter.

Returns: The number of occurrences of that element in the specified path. Returns 0 if the XML parsing path was not completed, or if the requested element does not exist.

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <name>John Smith</name>
    <address>
      <street>Somewhere</street>
      <city>Nowhere</city>
    </address>
  </issued_by>
  <item cat_num="2145451544">
    <price>99.99</price>
    <amount>2</amount>
  </item>
  <item cat_num="1384325456">
    <price>19.99</price>
    <amount>10</amount>
  </item>
</order>
```

Given that the XML above is saved into the file `C:\myxml.xml`, and that the current task's first I/O file points to an XML file, the following expressions apply:

XMLCnt (0, 1, 'order.issued_by.name')

Returns 1.

XMLCnt (0, 1, 'order.send_to,.name')

Returns 0 because there is no *send_to* element in the order.

XMLCnt (0, 1, 'order.item')

Returns 2.

XMLCnt (0, 1, 'order.item[2]', 'cat_num')

Returns 1.

Note: When an XML file is encoded in a different code page than the code page of the operating system, you can change the code page by using the CodePage function. The code page is assigned per context.

XMLDelete Lets you delete an XML element or attribute.

Syntax: XMLDelete (*generation*, *I/O file entry*, *element path*, *attribute name*)

Parameters: *generation* - A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

element path - A string representing the XML element path. The XML element path syntax is: **element name[[index]] [.element name[[index]]...]**

where:

element name is the name of an XML element. You can assign an alias that is separated from the element name by a colon symbol (:), for example, al1:My_Element.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

attribute name - The name of an XML attribute. If you are inserting an XML attribute, you must specify the attribute name. If you are inserting an XML element, leave this parameter blank.

Returns: A numeric value. 0 is returned when the XML element or attribute path is deleted. When this function fails, eDeveloper returns one of the following error codes:

- 1 Invalid I/O file
- 2 Inserted element or attribute alias is not defined
- 3 The I/O file not opened in Write mode
- 4 Element path not found
- 11 Invalid path, non-valid index
- 20 Invalid XML file (XML parsing failed)

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <address>
      <street>Somewhere</street>
      <city>Nowhere</city>
    </address>
  </issued_by>
  <item cat_num="2145451544">
    <price>99.99</price>
    <amount>2</amount>
  </item>
  <item cat_num="1384325456">
    <price>19.99</price>
    <amount>10</amount>
  </item>
</order>
```

- XMLDelete (0, 1, 'order.item[2].price',“)

In this example, the `<price>19.99</price>` element is removed from the document.

- **XMLDelete** (0, 1, 'order', 'id')
In this example, the `id` attribute is removed from the `order` element.

XMLExist

Returns a True value if an XML element or an XML attribute can be located by the XML's element path.

Syntax: `XMLExist (generation, file, element path [, attribute name])`

Parameters: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, and so on.

file: A number that represents the sequence number of the I/O file in the current task.

element path: A string that represents the path of an XML element. The syntax is:

element name`[[index]]` [**.element name**`[[index]]`]...

where:

element name is the name of an XML element. You can assign an alias to the element name. The alias is separated from the element name by a colon symbol (:), for example, `al1:My_Element`.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

attribute name: The name of an XML attribute that is the attribute of the multi-occurrence element name. Its value is compared to the Value parameter.

Returns: A True value. Returns a False value if the XML parsing process was not completed, or if the requested element does not exist.

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <name>John Smith</name>
```

```
<address>
  <street>Somewhere</street>
  <city>Nowhere</city>
</address>
</issued_by>
<item cat_num="2145451544">
  <price>99.99</price>
  <amount>2</amount>
</item>
<item cat_num="1384325456">
  <price>19.99</price>
  <amount>10</amount>
</item>
</order>
```

Given that the XML above is saved into the file C:\myxml.xml, and that the current task's first I/O file points to an XML file, the following expressions apply:

XMLExist (0, 1, 'order.issued_by name') returns True.

XMLExist (0, 1, 'order.send_to name') returns False because there is no *send_to* element in the order.

XMLExist (0, 1, 'order.item[2] price') returns True.

XMLExist (0, 1, 'order.item[5] price') returns False because there are only 2 items in the order.

XMLExist (0, 1, 'order', 'id') returns True.

XMLExist (0, 1, 'order.issued_by', 'date') returns False because there is no *date* attribute in the *issued_by* element.

XMLExist (0, 1, 'order.issued_by name') returns False because *name* is a child element of the *issued_by* element, but not an attribute of it.

XMLFind

Returns the index of an XML element that has a value equal to a specified value, or if one of its attribute values is equal to a specified value.

Syntax: XMLFind (*generation, file, element path, element name, num, child element name/attribute name, value, [begin at]*)

Parameters: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 its immediate ancestor, and so on.

file: A number that represents the sequence number of the I/O file in the current task.

element path: A string that represents the path of an XML element. The path string syntax is:

element name[[index]] [.element name[[index]]...]

where:

element name is the name of an XML element. You can assign an alias to the element name. The alias is separated from the element name by a colon symbol (:), for example, al1:My_Element.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

element name: A name of an XML element that has multiple occurrences under the specified element path.

num: A numeric value. 0 means that the next parameter is a child element name and 1 means that the next parameter is an attribute name, as shown below:

- *child element name*: If you have entered 0 for the *num* parameter, you must specify the name of an XML element that is the child element of the multi-occurrence element name. The child element name value is compared to the value parameter. If the Value parameter string is empty, the value will be compared to the value of the multi-occurrence element name. If you have specified a child element name, the attribute name parameter must be empty.
- *attribute name*: If you have entered 1 for the *num* parameter, you must specify the name of an XML attribute that is the attribute of the multi-occurrence element name. Its

value is compared to the Value parameter. If you have specified an attribute name, the child element name parameter must be empty.

value: A string that contains the value to which the child element name and the attribute name is compared.

begin at: You can determine where the index starts in the element path for the Find operation. The default value starts at the first XML element occurrence in the element path.

Returns: The index of the XML element that can have multiple occurrences. If the XML parsing process was not completed or the requested element does not exist, a 0 value is returned.

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <name>John Smith</name>
    <address>
      <street>Somewhere</street>
      <city>Nowhere</city>
    </address>
  </issued_by>
  <item cat_num="2145451544">
    <price>99.99</price>
    <amount>2</amount>
  </item>
  <item cat_num="1384325456">
    <price>19.99</price>
    <amount>10</amount>
  </item>
</order>
```

Given that the XML above is saved into the file C:\myxml.xml, and that the current task's first I/O file points to an XML file, the following expressions apply:

XMLFind (0, 1, 'order.issued_by', 'name', 0, "", 'John Smith') returns 1.

XMLFind (0, 1, 'order.issued_by', 'name', 0, "", 'John') returns 0 because there is no name element with the value John under the order.issued_by path.

XMLFind (0, 1, 'order', 'item', 1, 'cat_num', '1384325456') returns 2.

XMLFind (0, 1, 'order', 'item', 1, 'cat_num', '12345') returns 0 because there is no item in the order whose catalog number equals 12345.

XMLFind (0, 1, 'order', 'item', 0, 'price', '99.99') returns 1.

XMLFind (0, 1, 'order', 'item', 0, 'amount', '2', 2) returns 2.

XMLGet

Returns the value of an XML element or an XML attribute according to its element path.

Syntax: XMLGet (*generation*, *file*, *element path*, *attribute name*)

Parameter: *generation*: A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.
file: A number that represents the sequence number of the I/O file in the current task.
element path: A string that represents the path of an XML element. The path string syntax is:
element name[[**index**]] [**.element name**[[**index**]]...]
where:

element name is the name of an XML element. You can assign an alias to the element name. The alias is separated from the element name by a colon symbol (:), for example, al1:My_Element.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

attribute name: The name of the XML attribute.

Returns: Alpha: If the XML parsing process is completed successfully, the requested element will exist and will return a value. If the XML parsing process was not successful, an empty string is returned.

Example: <?xml version="1.0"?>
 <order id="123">
 <issued_by>
 <name>John Smith</name>
 <address>
 <street>Somewhere</street>
 <city>Nowhere</city>
 </address>
 </issued_by>
 <item cat_num="2145451544">
 <price>99.99</price>
 <amount>2</amount>
 </item>
 <item cat_num="1384325456">
 <price>19.99</price>
 <amount>10</amount>
 </item>
 </order>

Given that the XML above is saved into the file
C:\myxml.xml, and that the current task first I/O file points
to an XML file, the following expressions apply:

XMLGet (0, 1, 'order.issued_by', 'name') returns John Smith.

XMLGet (0, 1, 'order.send_to', 'name') returns an empty
string because there is no *send_to* element in the order.

XMLGet (0, 1, 'order.item[2]', 'price') returns 19.99.

XMLGet (0, 1, 'order.item[5]', 'price') returns an empty
string because there are only 2 items in the order.

XMLGet (0, 1, 'order', 'id') returns 123.

XMLGet (0, 1, 'order.issued_by', 'date') returns an empty
string because there is no *date* attribute in the *issued_by*
element.

XMLGet (0, 1, 'order.issued_by', 'name') returns an empty
string because *name* is a child element of the *issued_by*
element but not an attribute of it.

XMLGet (0, 1, 'person.name', "") - No Namespace mode
XMLGet (0, 1, ':person.name', "") - The default namespace URI is used.

XMLGet (0, 1, 'al1:person.name', "") - The al1 namespace URI used for the person element. The default namespace URI is used for the name element.

XMLGet (0, 1, 'al1:person.name', "") - The al1 namespace URI used for the "person" element. The al2 namespace URI used for the "name" element.

Note: When an XML file is encoded in a different code page than the code page of the operating system, you can change the code page by using the CodePage function. The code page is assigned per context.

XMLGetAlias Retrieves the alias associated with a namespace for the root element.

Syntax: XMLGetAlias (*generation*, *I/O file entry*, *URI*)

Parameters: *generation* -A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

URI - A alphanumeric string representing a namespace URI. An empty URI is not allowed.

Returns: An alphanumeric string representing the alias associated with the URI. If the URI is considered the default namespace for the XML, an empty string is returned. If an error occurs or there is no matching alias for the URI, a NULL value is returned.

This function fails when:

- There is no alias-namespace pair defined for the specified URI.
- The specified URI is not valid.

Example: XMLGetAlias(0, 1, 'http://www.magicsoftware.com/mgns') retrieves the alias associated with URI - http://

XMLGetEncoding

Retrieves the encoding of an XML document.

Syntax: XMLGetEncoding (*generation*, *I/O entry*)

Parameters: *generation* - A number that represents the hierarchy of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

Returns: An alpha string that includes the encoding for the XML file that was opened under the specific I/O entry determined by the generation and I/O entry.

See also: XMLSetEncoding

XMLInsert

Lets you insert an XML element at a specified location in an XML document or add an attribute to an existing XML element.

Syntax: XMLInsert (*generation*, *I/O file entry*, *element path*, *attribute*, *value* [,*before/after flag*, *reference element*, *auto convert*])

Parameters: *generation* - A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

element path - A string representing the XML element path. If the element path includes a single element name, a root element is added. The XML element path syntax is:

element name[[**index**]] [**.element name**[[**index**]]...]

where:

element name is the name of an XML element. You can assign an alias that is separated from the element name by a colon symbol (:), for example, al1:My_Element.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid. When inserting

an element, the attribute parameter has an empty string, you cannot specify the index for the rightmost element in the element path.

attribute - A string representing the attribute name that is added to the XML element specified by the element path. If this parameter has an empty string, an XML element is added, as specified by the element path.

value - An Alpha, BLOB (Rich Edit), or Memo string type containing the value of the inserted element or attribute.

before/after flag (optional) - Represents one of the following values:

- A - The element is added after the reference element.
- B - The element is added before the reference element.

If this parameter is empty for an XML element, the XML element is added as the last element inside its parent element.

For an attribute, this parameter is ignored.

reference element (optional) - A string representing an XML element name used as a reference for the insertion of the new XML element. For an attribute, this parameter is ignored.

If there is no reference element value, the new element will be added as the first or last element depending on the before/after flag value. If this parameter is set to A, the element is added as the last element inside the parent element. If this parameter is set to B, the element is added as the first element inside the parent element.

auto convert (optional) - Enter one of the following logical values:

- True - Converts the value into a valid XML string, which has the same effect as the XMLVal function.
- False - Does not convert the value into an XML string.

Returns: A numeric value. If a new element has been entered successfully, the function returns the index of the newly inserted element. If a new attribute has been inserted successfully, the function returns 0.

When the function fails, eDeveloper returns the following error codes, highlighted below:

- 1 Invalid I/O file
- 2 Inserted element or attribute alias is not defined
- 3 The I/O file not opened in Write mode
- 4 Element path not found
- 6 Attribute already defined for element
- 7 Invalid Before/After flag
- 8 Reference element not found
- 9 The document contains a root element. Multiple roots not permitted.
- 10 Invalid auto convert flag
- 11 Invalid path, non-valid index
- 13 Invalid alias, qualified name
- 20 Invalid XML file (XML parsing failed)

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <address>
      <street>Somewhere</street>
      <city>Nowhere</city>
    </address>
  </issued_by>
  <item cat_num="2145451544">
    <price>99.99</price>
    <amount>2</amount>
```



```
</item>
<item cat_num="1384325456">
  <price>19.99</price>
  <amount>10</amount>
</item>
</order>
```

Adding a New XML Element as the Last Element

Given that the first I/O file of the current task points to an XML file, consider the following expressions:

XMLInsert (0, 1, 'order.issued_by.name', "", 'John Smith')

This example adds a new XML element **name** as the last element inside the issued_by type. The issued_by section appears as:

```
<issued_by>
  <address>
    <street>Somewhere</street>
    <city>Nowhere</city>
  </address>
  <name>John Smith</name>
</issued_by>
```

The return value is 1.

Adding a New Element

To add a new **state** element after the city element, use:

XMLInsert (0,1, 'order.issued_by.address.state', "", 'Some state', 'A', 'city')

The issued_by section now appears as:

```
<issued_by>
  <name>John Smith</name>
  <address>
```

```
<street>Somewhere</street>
<city>Nowhere</city>
<state>Some state</state>
</address>
<name>John Smith</name>
</issued_by>
```

The return value is 1.

Adding an Attribute to an Existing Element

To add an **identification to** attribute an existing element name, use:

XMLInsert (0, 1, 'order.issued_by.name','ID','1')

This example inserts a new XML attribute to the specified element name. The issued_by

section appears as:

```
<issued_by>
<address>
  <street>Somewhere</street>
  <city>Nowhere</city>
  <state>Some state</state>
</address>
<name ID="1">John Smith</name>
</issued_by>
```

The return value is 0.

Adding an Attribute Type

To add an attribute type to the second item, use

XMLInsert (0, 1, 'order.item[2]', 'type', 'standard')

The function returns 0.

Index Specified for the Rightmost Element

XMLInsert (0, 1, order.item[3],",")

This example fails and returns Error Code -11 - **Invalid path, non-valid index** - because an element name is inserted and an index is specified for the rightmost element in the element path.

XMLModify Lets you modify the value of an XML element or attribute.

Syntax: XMLModify (*generation*, *I/O file entry*, *element path*, *attribute name*, *value*, [, *Auto Convert*])

Parameters: *generation* - A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

element path - A string representing the XML element path. The XML element path syntax is: **element name**[[**index**]] [**.element name**[[**index**]]...]

where:

element name is the name of an XML element. You can assign an alias that is separated from the element name by a colon symbol (:), for example, al1:My_Element.

index is the index of a specified occurrence of an element that has multiple occurrences. The index value must be greater than 0. Negative values are invalid.

attribute - A string representing the attribute name that is modified. If this parameter has an empty string, the value of the XML element is modified.

value - An alpha string with the actual value of the XML element or attribute.

auto convert (optional) - Enter one of the following logical values:

- True - Converts the value into a valid XML string, which has the same effect as the XMLVal function.
- False - Does not convert the value into an XML string.

Returns: A numeric value. 0 is returned when the XML element or

attribute path modification has been successful. When the function fails, eDeveloper returns one of the following error codes:

- 1 Invalid I/O file
- 2 Inserted element or attribute alias is not defined
- 3 The I/O file not opened in Write mode
- 4 Element path not found
- 5 Attribute not found
- 10 Invalid auto convert flag
- 11 Invalid path, non-valid index
- 20 Invalid XML file (XML parsing failed)

Example:

```
<?xml version="1.0"?>
<order id="123">
  <issued_by>
    <address>
      <street>Somewhere</street>
      <city>Nowhere</city>
    </address>
  </issued_by>
  <item cat_num="2145451544">
    <price>99.99</price>
    <amount>2</amount>
  </item>
  <item cat_num="1384325456">
    <price>19.99</price>
    <amount>10</amount>
  </item>
```

`</order>`

To Update an Element Value

XMLModify (0,1,'order.item[2].price','','30')

This example updates the value of the **price element**, inside the second item element, from 19.99 to 30.00.

The return value is 0.

To Update an Attribute

XMLModify (0,1,'order.item[1]','cat_num','123456')

This example updates the **cat_num attribute** value of the first item element from '2145451544' to '123456'.

The return value is 0.

XMLSetEncoding

Sets the encoding of an XML document that was opened for Write access. The encoding affects both the encoding attribute in the document header, such as encoding=UTF 8, and the encoding of the actual XML document.

Syntax: XMLSetEncoding (*generation, I/O entry, encoding*)

Parameters: *generation* - A number that represents the hierarchic position of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

encoding - An alpha string representing the encoding used for an XML function to add data to the XML document.

Returns: A numeric value. If the encoding has been set properly, the function returns 0. If the function fails, eDeveloper returns the following error code:

-3 - The I/O file was not opened in Write mode.

See also: XMLGetEncoding

XMLSetNS

Associates an alias with a namespace URI for the root element.

Syntax: XMLSetNS (*generation, I/O file entry, alias, URI*)

Parameters: *generation* - A number that represents the hierarchy of the task. 0 represents the current task, 1 represents its immediate ancestor, and so on.

I/O file entry - A number that represents the sequence number of the I/O file entry in the current task.

alias - An alphanumeric string representing the name of the namespace alias. The string must begin with an alpha character. If this string is empty, eDeveloper uses the default namespace.

URI - An alphanumeric string representing a namespace URI. An empty URI is not allowed.

Returns: 0 when the operation has been successful. If an error occurred, this function returns one of the following error codes:

-3 - The I/O file not opened in Write mode

-12 - Alias already used

-13 - Invalid alias, qualified name

Example: `XMLSetNS(0, 1, "", 'http://www.magicsoftware.com/mgns')` defines a default XML namespace with URI - `http://www.magicsoftware.com/mgns`

`XMLSetNS(0, 1, 'al1', 'http://www.magicsoftware.com/mgns')` defines an XML namespace with URI - `http://www.magicsoftware.com/mgns` and alias - `al1`

XMLStr

Converts a valid XML string into an Alpha string.

Syntax: `XMLStr(source string)`

Parameter: *source string* - A valid XML string.

Returns: The function returns an Alpha string that has replaced the XML reserved characters with standard Alpha characters.

Example: `XMLStr('<Hello " World>')` returns '`<Hello " World>`'

XMLVal

Converts a string value into a valid XML string.

Syntax: XMLVal(source string)
Parameter: *source string* - Any string value.
Returns: The function returns an Alpha string converted into a valid XML string.
Example: XMLVal('<Hello " World!') returns '<Hello " World>'

Year

Year Value of the Date
Returns the year portion of a date in a four digit format.

Syntax: Year(date)
Parameter: *date*: Input date value or date expression
Returns: Number of years.
Example: If the system date is 01/28/92, Year(*date*()) returns 1992.

Display forms are used for online graphic displays. Their interface type can be either Browser, for browser-based display, or GUI Display, for eDeveloper-Client display. Their Class must be defined as 0.

In this chapter:

• Browser Form Properties
• Browser Controls
• GUI Display Form Properties
• GUI Display Commands
• GUI Display Controls

Browser Forms

Browser forms are used when deploying a Web-based application.

You can create a browser form only if the task is defined as a browser task in the Task Properties dialog.

You can create a Browser form with an external HTML editor. The eDeveloper toolkit easily integrates your browser task with the HTML file and provides easy handling of the HTML elements as eDeveloper controls.

When you zoom from the Browser entry of the Form repository, the HTML Control repository appears, as shown below.

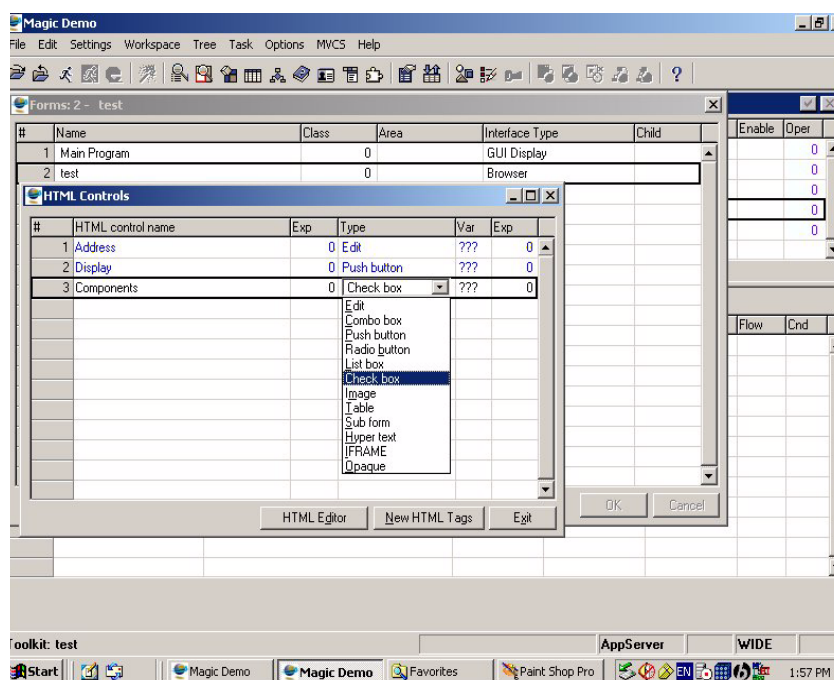


Figure 9-1 HTML Control Repository

Browser Subforms

The subform lets you call a program or task that is executed and displayed as part of a browser form. Subforms let you create sophisticated screens for the Internet by implementing a relationship between a called program or task and a browser form. A browser form can support multiple subforms.

You can zoom from a browser form entry in the Forms repository to the HTML Control repository and define a subform. The subform has the following properties:

- **Connect To** - Lets the developer connect to either a program or task.
- **Program/Task Number** - Lets the developer select a program or task entry number. The Call Program operation requires an entry number.
- **Arguments** - Lets the developer select the arguments that will be passed to the subform.
- **Tab Into** - Select Yes to let the end-user tab to the subform. If No is selected, the end-user must move the cursor to the subform.
- **Is Cached** - When this property is set to Yes, the client caches the subform dataview for each selected record of the parent browser task. This means that the client does not have to perform the redundant and time-consuming operation of accessing the server for the subform dataview when the end-user returns to a previously selected record.

The subform dataviews are cleared from the subform cache when:

- A View Refresh action is used.
- Selecting the Create mode. If you create a new line when in Modify mode, the client will not reset the subform cache.
- A Rollback action is used.
- The browser task is closed.

When No is selected, the client does not cache the subform dataviews, but accesses the server every time the subform dataview changes.

Keyboard Access to Subforms

You can access a subform from the parent task by using the Tab key.

The tab order of every subform is determined by its location in the HTML Control dialog, and the tabbing order of the variables as determined by the order of Select operations.

The subform location in the tab order is set between the variable that is defined in the HTML control before the subform control.

Listed below is the different behavior when accessing to the subform:

- When the Next Field event is selected from the parkable control of the parent task, defined before the subform entry, the cursor appears in the first parkable control of the subform.
- When the cursor is parked in the last parkable control of the subform, and the Next Field event is executed, the cursor moves to the control that is next to the last parkable control of the parent according to the order of the Select operations.
- If the next control is another subform, the cursor appears in the first parkable control of the next subform.
- When the subform is not the first control and the cursor is parked in the first parkable control, the Previous Field event moves the cursor to the

parkable control that is defined in the HTML Control list of the parent task previous to the subform.

- If the previous control is another subform, the cursor moves to the last parkable control of the previous subform.
- When the subform is defined as the first HTML control in the parent task, the task parks directly on the first parkable control of the subform. This is true also for nested subforms.
- The Cycle on Record Main property setting for the various subforms is ignored.



Set the Tab Into property to No, to prevent unintentionally accessing the subform by pushing the TAB key.

Browser Form Properties

The Browser form properties are listed below.

Model

- Model - The browser form model lets you do the following:
 - Select another model
 - Display the default form properties
 - Disinherit all inherited properties
 - Inherit all disinherited properties

Details

- # of Repeated Lines - Specifies the maximum number of lines in a Table control.
- HTML File - The HTML file assigned as the interface definition source for the Browser form.
- Form Name - Defines the title of the Browser form.

Input

- Modal Window - A window that halts the execution of a calling task. This means that a task that calls a modal window task will stop its execution until the called task is closed. If the task is set to be Modal=No, then it will let the end-user to run both tasks (the caller task and the called task) simultaneously.

Appearance

- Help Screen - You can zoom from this property to the Help List and select the appropriate Help.
- Wallpaper - The image file used as wallpaper for the form.

Navigation

- Left - Specifies the position of the left edge of the form's frame. You can specify the value at runtime by zooming to the Expression Rules repository, and entering an expression that evaluates to the coordinates of the left edge of the form's frame.
- Top - Specifies the position of the top edge of the form's frame. You can specify the value at runtime by zooming to the Expression Rules

repository, and entering an expression that evaluates to the coordinates of the top edge of the form's frame.

- Width - Determines the width of the form.
- Height - Determines the height of the form.

HTML Control Repository

The HTML Control repository displays the HTML input tag assigned to an eDeveloper variable. An HTML Control entry has the following properties:

- # - An automatically generated entry number.
- HTML Control Name - The control label. If you give the control a name, the Control Name Expression property is disabled.
- Control Name Expression - You can define an expression that will display the Control Name. If you define an expression, the Control Name property is disabled.
- Type - Lets you select the control type that is associated with an eDeveloper variable.
- Variable - Lets you select the variable, from the Variable list, that you want to associate with a specific control type. If you select a variable, the Variable Expression property is disabled.
- Expression - You can define an expression that will be used as the control's data content.
- The New HTML Tags button opens a list of unassigned HTML controls. You may select one or several controls of this list to be added as an HTML control that can be handled by eDeveloper through the HTML Control list.



Moving, repeating, or overwriting an entry are enabled in the HTML Control list of the Browser form.

To repeat an entry, you are required to assign a unique name at the Control level of where the action is repeated.

Browser Controls

The controls below are unique to browser tasks.

IFRAME - An element that defines an inline or floating frame for displaying HTML documents and other external objects.

Opaque - An element that can be defined as any HTML element that eDeveloper does not support.

Browser Control Properties

Browser Edit Control Properties

Model

- Model - The model from which the control can inherit or re-use predefined properties.

Details

- Format - Specifies the picture of the control by which the data is entered and displayed.
- Attribute - Lets you select a field attribute (such as alpha, numeric, date, and so on).

Input

- Password Edit - Determines whether the control content is displayed by asterisks as a password edit control.
- Must Input - Specifies whether the end-user is required to input a value to control.
- Modifiable - Specifies whether the end-user can change the value in a control during runtime.
- Multi-line Edit - Lets the end-user enter multiple lines of text.
- Select Program - You can determine the program that will be opened when the end-user clicks on this control.
- Select Mode - Determines the mode of the selected program.
 - Before - The selected program is opened by a zoom event. The Magi engine remains on the field when the selected program is closed.
 - Prompt - The selected program opens immediately when the end-user parks on the control.

- After - The selected program is opened by a zoom event. The eDeveloper engine proceeds to the next field when the selected program is closed.

Appearance

- Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- Color - Specifies the control color.
- Help Screen - Specifies whether a help screen is associated with the control.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Help Prompt - Specifies whether a Help prompt is associated with the control.
- Visible - Specifies whether the control is displayed.
- Enabled - Specifies whether a displayed control is enabled.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Radio Button Control Properties

Model

- Model -The model from which the control can inherit or re-use predefined properties.

Input

- Select Program - You can determine the program that will be opened when the end-user clicks on this control.
- Select Mode - Determines the mode of the selected program.
 - Before - The selected program is opened by a zoom event. The Magi engine remains on the field when the selected program is closed.
 - Prompt - The selected program opens immediately when the end-user parks on the control.
 - After - The selected program is opened by a zoom event. The eDeveloper engine proceeds to the next field when the selected program is closed.

Appearance

- Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- Color - Specifies the control color.
- Help Screen - Specifies whether a Help screen is associated with the control.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Help Prompt - Specifies whether a Help prompt is associated with the control.
- Visible - Specifies whether the control is displayed.
- Enabled - Specifies whether a displayed control is enabled.

Browser Hyper-Text Control Properties

Model

- Model - The model from which the control can inherit or re-use a predefined property.

Details

- Format - Specifies the picture of the control by which the data is entered and displayed.

- Attribute - Lets you select a field attribute (such as alpha, numeric, date, and so on).

Input

- Hyperlink - You can define the hyperlink details from the Hyperlink dialog box, as listed below to call a program:
 - Hyperlink Type - Select either eDeveloper Program or URL.
 - Magic System - For an eDeveloper Program. Specify a defined eDeveloper application.
 - Public Name - For an eDeveloper Program. Enter a public name.
 - Arguments - For an eDeveloper Program. Specify program arguments.
 - URL - For a URL. Enter the URL address.
 - Expression - For a URL. Select an expression that supplies an URL address when it evaluates to true in runtime.
 - Destination Frame - For a URL. Enter the name of the destination frame.

Appearance

- Default Class - Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- Color - Specifies the control color.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Visible - Specifies whether the control appears to the end-user.
- Enabled - Specifies if the control is enabled.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Push Button Control Properties

Model

- Model -The model from which the control can inherit or re-use predefined properties.

Details

- **Format** - Specifies the picture of the control by which the data is entered and displayed.
- **Attribute** - Lets you select a field attribute (such as alpha, numeric, date, and so on).
- **Raise Event** - Lets the end-user activate an event during the flow of the eDeveloper engine.

Input

- **Select Program** - You can determine the program that will be opened when the end-user clicks on this control.
- **Select Mode** - Determines the mode of the selected program.
 - **Before** - The selected program is opened by a zoom event. The Magi engine remains on the field when the selected program is closed.
 - **Prompt** - The selected program opens immediately when the end-user parks on the control.
 - **After** - The selected program is opened by a zoom event. The eDeveloper engine proceeds to the next field when the selected program is closed.

Appearance

- Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- Help Screen - Specifies whether a help screen is associated with the control.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Help Prompt - Specifies whether a Help prompt is associated with the control.
- Visible - Specifies whether the control is displayed.
- Enabled - Specifies whether a displayed control is enabled.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Check Box Control Properties

Model

- Model - The model from which the control can inherit or re-use predefined properties.

Details

- Attribute - Lets you select a field attribute (such as alpha, numeric, date, and so on).

Input

- Modifiable - Specifies whether the end-user can change the value in a control during runtime.
- Select Program - You can determine the program that will be opened when the end-user clicks on this control.
- Select Mode - Determines the mode of the selected program.
 - Before - The selected program is opened by a zoom event. The Magi engine remains on the field when the selected program is closed.
 - Prompt - The selected program opens immediately when the end-user parks on the control.
 - After - The selected program is opened by a zoom event. The eDeveloper engine proceeds to the next field when the selected program is closed.

Appearance

- Default Class - Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- Color - Specifies the control color.
- Help Screen - Specifies whether a Help screen is associated with the control.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Help Prompt - Specifies whether a Help prompt is associated with the control.
- Visible - Specifies whether the control is displayed.
- Enable- Specifies whether a displayed control is enabled.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser List Box Control Properties

Model

- Model - The model from which the control can inherit or re-use predefined properties.

Details

- Label - A string that defines the available options to be selected from the list box. The options are concatenated one after the other in the string, where each option is delimited by a comma character (,).
- Attribute - Lets you select a field attribute (such as alpha, numeric, date, and so on).
- Source table - You may specify a table from the table repository to be used as a source for the options in the list box.
- Display Field - When a source table is defined for a list box, you can define a field from the source table to be used as the description for each list box entry.
- Linked Field - When a source table is defined for a list box, you may define a field from the source table to be used as the returned value of the selected option.
- Index - When a source table is defined for a list box, you may select an index to determine the order of list box options.
- Field Ranges - When a source table is defined for a list box, you can define a range of values for each field to limit the range of displayed options.

Input

- Modifiable - Specifies whether the end-user can change the value in a control during runtime.
- Select Program - You can determine the program that will be opened when the end-user clicks on this control.
- Select Mode - Determines the mode of the selected program.

- Before - The selected program is opened by a zoom event. The Magi engine remains on the field when the selected program is closed.
- Prompt - The selected program opens immediately when the end-user parks on the control.
- After - The selected program is opened by a zoom event. The eDeveloper engine proceeds to the next field when the selected program is closed.

Appearance

- Default Class - Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- Color - Specifies the control color.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- # of Selection Rows - The number of rows that appear in the list box.
- Color - Specifies the control color.
- Help Screen - Specifies whether a help screen is associated with the control.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Help Prompt - Specifies whether a Help prompt is associated with the control.
- Visible - Specifies whether the control is displayed.

- Enabled - Specifies whether a displayed control is enabled.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Combo Box Control Properties

Model

- Model - The model from which the control can inherit predefined properties.

Details

- Label - A string that defines the available options to be selected from the list box. The options are concatenated one after the other in the string, where each option is delimited by a comma character (,).
- Attribute - Lets you select a field attribute (such as alpha, numeric, date, and so on).
- Source table - You may specify a table from the table repository to be used as a source for the options in the list box.
- Display Field - When a source table is defined for a list box, you can define a field from the source table to be used as the description for each list box entry.

- **Linked Field** - When a source table is defined for a list box, you may define a field from the source table to be used as the returned value of the selected option.
- **Index** - When a source table is defined for a list box, you may select an index to determine the order of list box options.
- **Field Ranges** - When a source table is defined for a list box, you can define a range of values for each field to limit the range of displayed options.

Input

- **Modifiable** - Specifies whether the end-user can change the value in a control during runtime.
- **Select Program** - You can determine the program that will be opened when the end-user clicks on this control.
- **Select Mode** - Determines the mode of the selected program.
 - **Before** - The selected program is opened by a zoom event. The Magi engine remains on the field when the selected program is closed.
 - **Prompt** - The selected program opens immediately when the end-user parks on the control.
 - **After** - The selected program is opened by a zoom event. The eDeveloper engine proceeds to the next field when the selected program is closed.

Appearance

- **Default Class** - Default Class - Specify the class name of the control by an expression.
- **Default Styles** - Specifies a list of styles for a control.
- **Color** - Specifies the control color.
- **Help Screen** - Specifies whether a help screen is associated with the control.

- Help Prompt -Specifies whether a Help prompt is associated with the control.
- Visible - Specifies whether the control is displayed.
- Enabled - Specifies whether a displayed control is enabled.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Image Control Properties

Model

- Model- The model from which the control can inherit predefined properties.

Details

- Attribute - Lets you select a field attribute (such as alpha, numeric, date, and so on)

Appearance

- Default Class - Default Class - Specify the class name of the control by an expression.
- MouseOver Class - Specify by an expression the class of the control in a MouseOver state.
- MouseDown Class - Specify by an expression the class of the control in a MouseDown state.
- Default Styles - Specifies a list of styles for a control.
- MouseOver Styles - Specifies the list of styles to take effect on a control in a MouseOver state.
- MouseDown Styles - Specify the list of styles to take effect on a control in a MouseDown state.
- ToolTip - Specifies whether a tooltip is associated with the control.
- Visible - A value that specifies whether the control appears to the end-user.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Table Control Properties

Model

- Model -The model from which the control can inherit predefined properties.

Details

- Magic Row Highlighting - When a cursor parks on a table row, the row is highlighted.

- Details Line # - You can specify the detail line that you want to repeat for the length of your table. The execution of a browser task in the Details Line Number property is set to zero. The number of repeated lines specified by the physical size of the smallest table in the HTML file may be slower than when it is set to a value larger than zero.

Appearance

- Default Class - Specify the class name of the control by an expression.
- Default Styles - Specifies a list of styles for a control.
- Color - Specifies the control color.
- ToolTip - Specifies whether a ToolTip is associated with the control.
- Visible - Specifies whether the control appears to the end-user.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.

Browser IFRAME Control Properties

Model

- Model -The model from which the control can inherit predefined properties.

Appearance

- ToolTip - Specifies whether a ToolTip is associated with the control.
- Visible - Specifies whether the control appears to the end-user.
- Default Class - Specifies the class name of the control by an expression.
- Default Styles - Specifies a list of styles for a control.

Navigation

- Left - Determines the left position of the control on the form.
- Top - Determines the top position of the control on the form.
- Width - Determines the width of the control.
- Height - Determines the height of the control.

Browser Opaque Control Properties

Model

- Model- The model from which the control can inherit predefined properties.

Appearance

- Tooltip - Specifies whether a ToolTip is associated with the control.
- Visible - Specifies whether the control appears to the end-user.
- Enabled - Defines whether or not the control will be active.
- Default Class - Specifies the class name of the control by an expression.

- **MouseOver Class** - Specifies by an expression the class of the control in a MouseOver state.
- **MouseDown Class** - Specifies by an expression the class of the control in a MouseDown state.
- **Default Styles** - Specifies a list of styles for a control.
- **MouseOver Styles** - Specifies the list of styles to take effect on a control in a MouseOver state.
- **MouseDown Styles** - Specifies the list of styles to take effect on a control in a MouseDown state.

Navigation

- **Left** - Determines the left position of the control on the form.
- **Top** - Determines the top position of the control on the form.
- **Width** - Determines the width of the control.
- **Height** - Determines the height of the control.

GUI Display Forms

GUI Display Form Properties

The Form Properties sheet specifies the appearance and the behavior of a form. Select *Edit/Properties* or press CTRL+P while on an entry in the Form repository to access the Form Properties sheet.

For Class > 0 forms, the forms in the Form repository with the same Class are displayed together. If you change the following form properties for one form, eDeveloper will automatically change the properties of all of the other forms of the same Class to the new property settings: Form Units, Vertical Factors, Horizontal Factors, Grid X, Grid Y, and Font.

eDeveloper supports character-based forms only of Class > 0. properties that do not apply to character-based forms are disabled.

The GUI Form properties are listed below.

Model

- **Model** - The model from which the control can inherit a predefined property

Details

- **Modal Window** - You can specify that the form behave as a modal window, that is, you cannot click another window without first closing this window.
- **Floating Window** - You can specify that the form behave as a floating window, that is, the window can be dragged outside the boundaries of the eDeveloper main window.
- **Form Units** - You define the size and position of forms in terms of units of measurement. Units of measurement also specify the resolution of the form. You can define a form in dialog units: characters, centimeters, or inches.
- **Vertical Factor** - The Vertical Factor specifies the vertical resolution within each unit of measurement. A higher value gives a finer resolution.
- **Horizontal Factor** - The Horizontal Factor specifies the horizontal resolution within each unit of measurement. A higher value gives a finer resolution.
- **Show Grid** - Defines whether the Form Editor will display the dot grid. If you enter a Yes value, the form layout will appear with a grid. If you enter a No value, the form layout will appear without a grid.
- **Grid X, Grid Y** - Defines the distance between the grid dots on the form.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the form from other controls in the same application or other applications that run on the same machine. The data copied on to the form

can be set manually by defining a handler over the Drop event or automatically by the eDeveloper engine.

Input

- **Title Bar** - Specifies whether a GUI form has a title bar. Remember that without a title bar the end-user cannot move the form, and you cannot display Minimize and Maximize buttons. The Title Bar can also be specified by defining an expression from the Expression Rules repository.
- **System Menu** - Specifies if a GUI form will include a System Menu button on the left side of the title bar. When clicked, a Windows System menu appears. The default setting is Yes.
- **Minimize Button** - Specifies if a GUI form will include a Minimize button on its upper right hand corner. The default setting is Yes.
- **Maximize Button** - Specifies if a GUI form will include a Maximize button on its upper right hand corner. The default setting is Yes.
- **Child Window** - Specifies if eDeveloper will open this form as a child window as part of its parent, or as a separate window.
- **Split Window Child** - If you want the task to be displayed in a secondary window in the split parent task window, set this property to Yes.
- **Average Palette** - In some forms that include many graphic images, the images may use different color palettes whose combination produces unsuitable resolution in the form. In such cases, you can set Use Average Palette to Yes to enable eDeveloper to use the palette that gives the average color for a selected graphic image.
- **Default Button** - A Default button of a form is a mode of the form to highlight the main button and to activate this button whenever the end user presses the Enter key even if the user is not parked on the button.

This property lets you specify the Default Button by selecting a push button control name for an online task from the Control list. You can also specify an expression that determines the control name at runtime.

Activation

At runtime, when the end user presses the ENTER key, eDeveloper activates the push button specified in the Default Button property, which in turn raises the event that is specified in the Raise Event property of the Push button.

Default Button Indication

At runtime, if a push button is evaluated as the Default Button, it is highlighted in the standard manner to indicate that it is a default push button.

Note: In a program set to be a Selection table, if you set the default push button to raise the Select event, when you press ENTER the default push button is not activated and the Select event is directly triggered by the ENTER key as though you hadn't defined a default push button. This results in slightly modified behavior where the flow will not reach the default push button as it does for other programs.

Appearance

- **Wallpaper** - Specifies the screen pattern that serves as the background of the window. You can create an expression to specify the wallpaper of a task window, by zooming to the Expression Rules repository.
- **Font** - Specifies the font of the form. Zoom or double-click on the property to choose a font from the Font repository. You can specify the value at runtime by zooming to the Expression Rules repository to enter an expression that evaluates to the number of a font in the Font list.
- **Color** - Specifies the color of the form. Zoom or double-click to the Colors list. You can specify the value at runtime by zooming to the Expression Rules repository, and entering an expression that evaluates to the number of a color in the Color list. If you use an expression, the expression result overrides the specific Color set in this property.
- **Help Screen** - Specifies whether to attach a Help screen to the form. Zoom to the Help list to choose a Help screen
- **Border Style** - Specifies the style of the form's border. The valid values for this property are Thick, Thin or No Border.

Split

A split bar divides the form into two separate windows adjacent to each other. Split properties are described below:

- **Split Window** - This property lets you select a Vertical or Horizontal split bar. The default value is No.
- **Primary Display** - This property lets you determine where the content of the parent task is displayed. When the Split Window property is set to Vertical, the content can be displayed either in the left or right split window. When you select Default, the default value is the left split window for left-to-right applications and the right split window for right-to-left applications. When the Split Window property is set to Horizontal, the

content can be displayed either in the Top or Bottom split window. The default value is Top.

- **Split Offset %** - This property lets you enter a percent number or set an expression to offset the Vertical or Horizontal split from the top, left corner.
- **Split Placement** - This property lets you determine the split placement factor where the splitter moves according to a change of the window's dimensions. The default value is 0.
- **Splitter Style** - This property lets you assign the splitter style as either 2D or 3D style.

Navigation

- **Fit to MDI** - Enables you to set the display form for a task to fit the available space of the eDeveloper MDI. This property simulates the behavior of a maximized form.

When the Fit to MDI property is set to Yes, the Modal window, Floating window, Minimize button, Maximize button, and Child window are automatically set to No. In addition, the Startup Position property is disregarded.

- **Startup Position** - This property lets you to define the mode for how the GUI Display form opens. The Startup Position options are:
 - **Customized** - The window opens at the location defined in the Top and Left navigation properties of the form. The form size is defined from the Width and Height navigation properties.
 - **Centered to Magic** - The window opens centered within the eDeveloper client area. The form size is defined by the Width and Height properties.
 - **Centered to Desktop** - The window opens centered within the desktop area. The form size is defined by the Width and Height properties.
 - **Centered to Parent** - The window opens centered within the parent window. The form size is defined by the Width and Height properties.

- **Default** - The window opens with the default location and size provided by the operating system. The Top, Left, Width, and Height properties are ignored.

Note: If the parent task does not have an opened form, the next ancestor form is regarded as the parent form. If the opened form does not have a form from an ancestor task, the eDeveloper client area is regarded as the parent form, which will produce the same result as the Centered to Magic option.

- **Placement** - The Placement property specifies whether GUI forms that are defined as child forms are resized when a parent form is resized. Standalone forms do not support the Placement property. The Placement property is a combination of the following four percentage settings in the Placement dialog: Left placement, Right placement, Top placement, and Bottom placement. Zoom, **F5**, from this property to open the Placement dialog.

When a GUI form's Placement property equals 0, the relative size does not change when the size of the parent form is changed in runtime. When the Placement property is > 0, the relative size changes when the size of the parent form is changed in runtime.

eDeveloper lets you define both a Placement value and an expression for a corresponding Navigation property. In runtime, eDeveloper combines the expression and the Placement property values.

- **Left** - Specifies the position of the left edge of a GUI form's frame. You can specify the value at runtime by zooming to the Expression Rules repository and entering an expression that evaluates to the coordinates of the left edge of the form's frame.
- **Top** - Specifies the position of the top edge of a GUI form's frame. You can specify the value at runtime by zooming to the Expression Rules repository and entering an expression that evaluates the coordinates of the top edge of the form's frame.
- **Width** - The horizontal dimension of the frame around the form. If the frame is too small to display the controls, eDeveloper will display a scroll

bar to scroll the contents within the window. You can change the Frame Width by dragging the right edge of the frame.

- **Height** - The vertical dimension of the frame around the form. If the frame is too small to display the controls, eDeveloper will display a scroll bar to scroll the contents within the window. You can change the Frame Height by dragging the bottom edge of the frame with the mouse.

GUI Display Commands




You can use the Command palette, the Align and Size pulldown menus, or the context menu to edit the controls you have placed on the form. The Command palette includes tools for aligning, resizing, and for arranging the Z-Order of controls.

The buttons in the GUI Command palette are color-coded:

- Aqua-colored buttons align a selected group of controls.
- Yellow-colored buttons perform sizing operations.
- Green-colored buttons are used to determine the order of control placement.

The maximum number of command operations saved in the Undo buffer is 10. When expressions are deleted during the editing of control properties, the Undo command operation becomes disabled and the Undo buffer is cleared.

The tool buttons that align controls are described in the following table:

Aligning Tool	Description
	Aligns all selected controls to the left side of the select frame.
	Aligns all selected controls to the horizontal center of the select frame.
	Aligns all selected controls to the horizontal center of the form.



Spaces all selected controls equally horizontally within the select frame.



Aligns all selected controls to the right side of the select frame.



Aligns all selected controls along the top of the select frame.



Aligns all selected controls to the vertical center of the select frame.



Aligns all selected controls to the vertical center of the form.



Spaces all selected controls vertically within the select frame.



Aligns all selected controls along the bottom of the select frame.

These tool buttons that resize controls are described in the following table.

Sizing Tool	Description
	Stretches all selected controls to the width of the widest control.
	Stretches all selected controls to the width of the narrowest control.
	Stretches all selected controls to the height of the tallest control.
	Stretches all selected controls to the height of the smallest control.

The buttons used to move controls forward or back, arrange the Z-Order of controls on the form, and to remove dividers between breaks in the form are described below.

Sizing Tool	Description
	Brings a selected control one layer forward in the form.



Sends a selected control one layer back in the form



Brings a selected control to the front of the form.



Brings a selected control to the front of the form.



Displays the Z-order of the controls in the form.










Enables the automatic Z-order of controls in the form



Removes dividers between breaks in the form

The buttons used for tables, links, undo and redo of the previous action, and to resize a control's frame to fit its text are described below.

Sizing Tool	Description
	Undo last action.
	Redo last action.
	Creates a parent-child link between controls.
	Breaks a parent-child link between controls.
	Attaches controls in a table.
	Indicates a parent control.
	Fits a group of selected controls as they appear on the form.

Note: the maximum number of operations saved in the Undo buffer is 10. When expressions are deleted during the editing of properties, the Undo control becomes disabled, and the Undo buffer is cleared.

Z-Order of Controls

The *Z-Order* of controls in a form is the depth of the controls as you insert them. The Z-Order becomes particularly important when you superimpose controls on top of one another.

eDeveloper recognizes two groups of controls with regards to Z-Order:

- Group I
 - Push buttons
 - Check boxes
 - Radio buttons
 - Sliders
 - Combo boxes
 - List boxes
 - OLEs
 - RTFs
 - Tree Control
- Group II
 - Images
 - Tabs
 - Static controls
 - Edit controls
 - Lines
 - Groups

You can rearrange the Z-Order within the two groups, but not between them. eDeveloper automatically arranges the Z-Order between Group I and Group II controls. For example, if you were to place an edit control on top of a push button control, the Z-Order would arrange the controls so the push button

control would be placed in front. You cannot bring a control from the second group in front of a control from the first group. You use the *Arrange* context menu or the *Command* palette to display or change the Z-Order of controls on a form.

Parent-Child Linking of Controls

The Parent-Child feature lets you define a “parent” control to which “child” controls can be linked.

Activation

- Linking is done in the Form Editor.
- By default, all controls on a form are children of the form.
- Each control may be assigned a parent control other than the form.
- A control that is a parent for other controls can be a child of another control.
- One or more controls may be assigned to the same parent control.
- Parent-child relationships cannot be circular.
- Controls that are children of a Table control cannot have other controls linked to them as children.

Runtime Behavior

- When a parent control becomes invisible, all its children are hidden.
- When a parent control is disabled, all its children are disabled.
- If an end-user invokes Help when parked on a control that does not have a Help property, eDeveloper will search for and display the Help screen associated with the Help property of the current control’s parent. If the parent control also does not have a Help property, eDeveloper will

continue to search ancestor controls until an ancestor with a Help property is found. Note that the primary ancestor is the form itself.

Parent-Child Links Using Choice Controls

All choice controls enable the selection of a value out of a list of possible values: radio button, tabs, list box, combo box, and check box. The allowed values are based on the label property of the control. The possible choice values in the Label property are comma delimited. The data value that can be assigned to the data property variable is dependent on the data's picture. An expression can be given for the Label property of the control, allowing dynamic creation of allowed choice values.

When the value of the variable assigned to the data property is not in the allowed range, none of the options will be visually selected.

Keyboard shortcuts can be created to browse among choices in all controls.

You can associate child controls with each of the allowed choice values of a Choice control (Radio, Tab, List, or Combo). In this case, the parent shows only the children that are associated with the selected choice. When selected, the parent control is displayed with a purple ruling box (a normal control appears with a blue ruling box).

You can define child controls that are not associated with any choice of a selection control, and are visible regardless of the selected choice.

When you drop a new control on a parent control, the new control automatically becomes a child of the currently displayed layer of the parent Choice control. If the current layer cannot be determined, the new control is linked as a child of all the layers of the parent control.

When you change the displayed layer of a Choice control, the child controls linked to the selected layer are displayed, and the child controls linked to other layers are hidden.

GUI Display Color Palette






The Color palette shares the same area as the Command palette. To toggle between the two palettes, click on the tabs at the bottom of the palette.











The list of default and user defined controls colors is displayed on a palette. Each color is identified by a Color button. Click a color button to insert that color for a selected control. Access the Color repository to define the color palette settings.







To change the color of a text or control background, select a form control and click the color combination that you want.

GUI Display Controls

The buttons in the GUI Control palette are described in the table below:

Control Type	Description
	The selection tool in the Control palette. Use this tool to select or unselect a control. You must select a control in order to move, resize or delete it, or to zoom to the Control Properties dialog.
	Inserts an edit control. You can attach this type of control to a variable or an expression in the Data property of the Control Properties dialog.
	Inserts a static text control.
	Inserts a push button that the user clicks to launch an action in the application.
	Inserts a check box that the user can toggle on/off. Use this type of control to represent logical variables or expressions.

Control Type	Description
	Inserts a box for radio buttons.
	Inserts a tab control containing two default tabs. You can place other control types such as edit controls and push buttons on a tab control.
	Inserts a list box control containing two default items.
	Inserts a combo box containing two default items.
	Draws a static rectangular box to group a number of controls together visually. You can include a label to describe the group.
	Draws a static rectangle. You can add static text to the rectangle and specify its position in the rectangle in the Control Properties dialog. Define the rectangle's color in the Color property of the Control Properties dialog.
	Draws a static ellipse. You can add static text to the ellipse and specify its position in the ellipse in the Control Properties dialog. To define the ellipse's color, set the Color property in the Control properties dialog
	Draws a graphic line. To define the line's color, set the Color property in the Control properties dialog.
	Inserts a vertical slider with direction arrows.
	Inserts a horizontal slider with direction arrows.

Control Type	Description
	Inserts a table control into the form. You can place other control types, such as edit controls, and push buttons on a table control.
	Inserts a graphic image control.
	Inserts an OLE container
	Inserts a rich text control containing static text that can be formatted during development. The formatting of a rich text control is not implemented on the control as a whole, but only during the editing of the selected text of the control.
	Inserts a rich edit control that contains a variable or expression. This control is implemented as a rich text control when the text originates from a BLOB variable or expression.
	Inserts a Tree control that displays a selected task dataview in a tree style with parent and child nodes.

The GUI controls are grouped in the following way:

- Static Controls
- Choice Controls
- Slider Controls
- Edit, Action, and Image Controls
- Table Controls
- Tree Controls

Static Controls

Four controls are static in nature:

- Group - A rectangular background on which you can place controls that you want to associate visually with one another. There is no logical linking of the controls within the group.
- Text - Static text.
- Rectangle - A rectangular graphic shape.
- Ellipse - An elliptical graphic shape.

Choice Controls

Choice controls display multiple values. When you click a Choice control when pressing the **ALT** key, all the controls associated with the selected layer are marked.

The Choice controls are:

- Radio Buttons - Radio buttons are used for multiple-choice objects. A radio button relates to a column with a discrete input range specification. Each of the range values is displayed as a single radio button, with the value displayed alongside it. The buttons appear inside a box that groups them together as one control, although the buttons look like separate entities. The user selects only one button from the group. When the user selects a button, a black dot appears in the circle to indicate that it has been

selected. The column gets the value represented by the button selected. If a logical column is displayed as a radio button and its range is not defined, eDeveloper uses the default values True and False.

- **Tab Control** - A Tab control is similar to a divider in a file cabinet or notebook. You can use the Tab control to define multiple logical pages or sections of information within the same window. Tab titles or labels can include text or graphics. The user selects the tab title to open the tab card containing the child controls of the selected tab title.
- **List Box Control** - A pre-constructed control for displaying lists of choices for end-users. The List box control is used to display a collection of items, and in most cases supports the selection of an item or items from the list.
- **Combo Box Control** - Combines a text box with a list box, allowing the user to choose an entry from the list. The Combo Box Height property sets the height of the closed combo box. An open combo box automatically closes after the time specified for the Pulldown menu close timeout property located under Preferences in the Environment dialog.

Data Bound Choice Controls

The various choice controls can be set to be data bound, which will provide the end-user with a range of available options taken from a defined database table.

You can easily make a choice control data bound by defining the following properties:

- **Source table** - the table from which the choice control options are taken.
- **Display field** - The field that is displayed as the option to be selected.
- **Linked field** - The field value that is returned as the selected value.
- **Index** - The index by which the table records are ordered.
- **Range Fields** - A list of range values set for defined variables of the source table to limit the range of options.

A data control may display duplicate values of the displayed field in the choice control. Selecting any one of the duplicate options returns the corresponding

linked field. Duplicate value options of the linked field are disregarded and are not displayed. Only the first occurrence of the linked field value is displayed.

The view created by a choice control is cached for every data control.

1. The view is cached according to its defined range criterion. The view of a data control will be taken from the cache only if the full range criterion matches a previously fetched view of the same data control.
2. The cache of the various views is handled separately for each select operation, and it is not reused for identical data controls of other select operations. Each data control of a specific variable has its own cache of records.

Slider Controls

The slider controls include a Horizontal Slider control and a Vertical Slider - A slider is based on a numeric column with a range that specifies the scale of values the slider represents. The column is displayed as a rule bar, and the position of the moving indicator on the rule bar specifies the value of the column. You can control the slider's appearance on the form by setting the x,y position indicators in the Control Properties dialog. The slider can be displayed in any x,y combination. You can add vertical and horizontal sliders anywhere in a form.

Editing, Action, and Image Controls

Edit Controls

Edit controls contain either variables or expressions, specified in the Value and Exp columns of the Data property. To customize an edit control, select it and zoom to access the Control Properties dialog.

Drag and Drop operations can be performed only for the selected text in the Edit control.

Push Button Controls

Push buttons are used as an alternative means of invoking eDeveloper internal events, such as the zoom action. The content of the Push button property becomes the button label. You specify the Push button's action in the Raise Event property in the Control Properties dialog. Choose an action from the Action list.

At runtime, pushing the button triggers a zoom action that then causes the Call Task, Call Program, or triggers an event.

Other applications for Push buttons are the OK, Cancel, and Help buttons. These buttons can issue OK, Cancel, and Help actions respectively. The actions set by the buttons will be captured by the eDeveloper programs according to the program context.

You can use the default Windows Push Button design, or design your own Push Button controls by setting the Button Style property of the Control Properties dialog to Yes.

Image Push Button

The user only needs to provide one bitmap image that can be used for the four different image conditions of the button: default in toolkit, focused, selected, or disabled in runtime. The Image Push Button feature changes the color and shading for a single bitmap image to create the three dimensional effect required for each button condition.

Checker Messages

The syntax checker issues the following warnings when the push button is handled by the Click event or the push button handler raises the Internal Click event:

- Handling a push button should be done by using a user event.
- A push button should not be set to raise a Click event.
Consider raising a user event instead.

Check Boxes

Check boxes are used to set logical fields to True when checked and to False when not checked. The field name is the description of the content, as in any ordinary field. The input value of the check box is supplied by the space bar or a mouse click on the check box. Either a space bar or a mouse click toggles the check mark on and off.

Image Controls

Image controls contain graphic images. You can insert these graphic images by specifying a file name, by mapping a variable that is a graphic image, or by defining an expression that evaluates to yield a graphic image.

Line Controls

The Line control draws a line between two grid points. Lines can also be placed with an Edit box.

OLE Controls

OLE lets you combine objects supported by different applications. eDeveloper's OLE control is a container for either embedded or linked OLE objects in your application.

OLE embedded objects are data objects, that, when moved or copied, retain their native editing and operating capabilities in their new container.

OLE linked objects are pointers or links to data objects in other locations or in other containers. For further details see OLE Control Properties.

OLE controls do not support eDeveloper's Drag and Drop operations.

RTF Controls

The Rich Text control, shown on the left, allows user to insert a Rich Text Format that can be better formatted during development.

The Rich Text control is part of the static control group.

The Rich Text control allows the developer to set the color and font for each character and set the paragraph design for each line inside the control.

To change the properties of text in an Rich Text control:

1. Park on the Rich Text control.
2. Press Enter. The control's border changes from blue to pink, which is the control's Edit mode.
3. Mark some of the text in the Rich Text control.
4. Right-click the mouse to open the Rich Text control shortcut menu, where you can make changes to the properties of the text.

Another RTF control, shown on the left, enables the user to insert a Rich Edit control that contains a variable or expression.

The Rich Edit control behaves the same as the Rich Text control, but only during runtime.

Only a variable with a Blob attribute can be attached to a Rich Edit control and allows the end-user to edit the text with any color, font, or paragraph setting.

You can define the numeric size for an RTF edit field through a setting called 'RtfBufferSize' that is located in the MAGIC_SPECIALS section of the Magic.ini file.

For example, `RtfBufferSize = 64` displays the RTF edit field as 64Kb.

The `RtfBufferSize` setting is read each time a new screen is built, prior to the Task Prefix. When the `RtfBufferSize` setting has a new value, and the user utilizes the `INIPut` function, the size value of the RTF edit fields is updated and displayed in the form of the task.

Revising the `RtfBufferSize` setting within a task does not affect the size of the RTF edit fields that were already created.

Drag and Drop operations can be performed only for the selected text in an RTF Edit control.

Table Controls

Table controls have two parts: Title area and a Repeated area, in which the lines of the table appear.

When you resize a form grid that has a Table control, the Table control resizes accordingly. For Horizontal placement, the Table control has the Size Placement property that controls the resizing percentage for the table. For example, if the Size Placement property is set to 80%, and the size of the form is increased by 10 pixels, then the size of the Table control is increased by 8 pixels.

For Vertical placement, the table control will display additional or fewer rows according to how the form is resized. The Table control displays empty rows when no more data is available. The parked Table Control row is always displayed.

Resizing in Development Mode

In the Development mode, each column is resized according to the resizing of the table. Each column has a Placement property. When the user sets this property to Yes, then the column is resized proportionally according to the total number of columns. For example, if each of the four columns has Yes set for the Placement property, then each column will be twenty-five percent of the table.

For columns that have Placement property = No, the size will not change. For example, you have a table control that is 8 pixels wide. The Size Placement property is set to 100%. There are four columns. Each column is 2 pixels wide. Two columns have Placement property = Yes. Two Columns have Placement property = No. When the form grid is increased by 8 pixels, then two columns will be 2 pixels wide, and two columns will be 6 pixels wide.

Resizing in Deployment Mode

The end-user can resize a table column in runtime by setting the Column Resizing property to Yes. Resizing the column by its right divider will increase the size of the column and push the remaining columns to the right. The remaining columns will maintain their size, so increasing the size of the column will increase the size of the entire table. Though the table will be the

same size on the form, fewer columns will be displayed. When the table is resized so that the table columns cannot be displayed together, a horizontal scroll bar will appear for the table.

The Drag Operation for a Table Control

Drag and Drop operations can be performed for the selected text in the Table control.

For a Table control with marked records, the Drag Begin handler is raised for every marked record. eDeveloper identifies the Drag operation and does not open the Edit mode when the mouse's left button is clicked and held.

Multi-Marking in a Table

Table controls allow multi-marking that is similar to the multi-marking feature in a Windows application. The multi-marking feature is available in both toolkit and runtime for interactive online forms only, GUI=0.

To enable multi-marking in the table control of a task, click Yes in the *Multi marking* property of the Table Control property sheet. The *Multi marking* property is enabled for interactive online tasks only.

Marked rows are reflected by the reverse of the Windows system color.

The following actions can be implemented for marked records:

- Delete - toolkit and runtime
- Check - toolkit, Check Syntax
- Copy - toolkit

Marking Columns

To mark an entry in a column, click Yes in the *Marking Column* property of the Column Control property sheet. When the *Marking Column* property is enabled, the record cells of the column appears raised. Clicking a record on an enabled column will mark the record. Clicking on a marked record will unmark the record.

Note: In order to mark the column only, hold down the ALT key and left-click on the column (anywhere in the column except for the table caption and the first row).

Event Handlers for Multi-Marking

The user can enable an event handler to process each marked table row by clicking Yes for the Enable Multi-Marking property. Once the action is invoked, the event is implemented for each selected row from the beginning of the data view to the bottom.

For example, the user wants to re-calculate totals for a summary table displayed on the form. The user calls the program that calculates and updates the total. Once the action is implemented, the event handler calls the program for each selected row, and displays the updated total.

When the event handler is complete, then marked rows can be maintained by clicking Yes for the Keep Multi-Marking property.

The multi-marking functions are listed below.

- | | |
|----------------|--|
| MMStop | Stops the multi-mark handler. |
| MMCount | Provides the number of marked rows. |
| MMCurr | Provides the current row of the number of marked rows in the counting process. For example: Row 5 of 15. |

Tree Control

You can use the tree control to display a task dataview in a data hierarchy, a tree structure with parent and child nodes that represent different levels of data.

The different data levels are displayed:

- **Root Value** - The expression value used as the starting point for the runtime engine to build the data tree. The root value is displayed as the top node that connects to the next level of data. When there is no root value, there is no top record that connects to the next level of data. As a

result, the runtime engine builds the data tree with the next level of data displayed as separate parent nodes.

- Parent Node - A variable value representing a parent record in the task dataview. The parent node cannot be defined from an expression.
- Node - A variable value representing a child record in the task dataview. The node cannot be defined from an expression.



The Tree control can only display records from a Main or Link Join table that are defined in a single database table.

Generating a Data Tree

The following runtime behavior occurs when an online task containing a tree control on the task's main form is opened:

1. The dataview range is calculated.
2. If the root value exists, the runtime engine displays the root value as the top parent node and fetches the record from the database.
3. The initial expand level for the tree is determined by the value selected for the Auto Expand or Single Expand properties.
4. When the Node Preload property is set to Yes, the runtime engine automatically retrieves the child records of each parent found in the dataview range. When the Node Preload property is set to No, the runtime engine only retrieves the child records for the expanded levels.
5. For each retrieved record, a new node is added to the tree with the relevant node's GUI data. Steps 4 and 5 will be repeated for all the records specified by the Auto Expand or Single Expand property.

Tree Control Events

You can use internal tree control events to let the end-user move, expand, and collapse nodes in the data tree.

The tree control events are:

- Collapse Node
- Collapse SubTree
- Create Child
- Edit Node
- Expand Node
- Expand SubTree
- Move to First Child
- Move to Next Sibling
- Move to Parent
- Move to Previous Sibling

Expand and Collapse Node Events

The information about triggering a node can be provided through arguments passed by the Expand Node and Collapse Node events. When selecting either the Expand Node or Collapse Node events, eDeveloper prompts you with a confirmation message about creating the Tree Node Level and Tree Node Value virtual variables.

When the Expand Node or Collapse Node events are triggered, the corresponding virtual variable for the user-defined handler is updated with tree node values related to the triggered event. The first Select operation is updated with the Tree Node Level. The second Select operation is updated with the Tree Node Value.

Raising the Expand Node and Collapse Node events outside the scope of the Tree control will not update the Tree Node Value and Tree Node Level fields.

Arguments passed by a Raise Event operation for the Expand Node and Collapse Node events are set in the handler's virtual fields after the first two variables.

Edit Node Toolkit and Runtime events

The Edit Node Toolkit and Runtime events let you edit the Tree control node descriptions in Toolkit or Runtime mode. Triggering the Edit Node Runtime event only lets you edit a mode that is parked upon. Click the left mouse button to select the node. Click the left mouse button a second time to trigger the event.

Adding or Deleting Nodes

You can add a node to the data tree by clicking **F4** or by selecting the Create Child event when the:

- Task has been opened in Modify mode
- Main screen is displayed
- Selected parent node has been expanded

You can delete a node by clicking **F3**. All nodes must be deleted prior to deleting the parent node. After you delete a node, the runtime engine automatically refreshes the data tree.

Navigating in the Data Tree

The runtime engine always points to the current node of the dataview, represented by the current node in focus. When you move to another node, the GUI layer instructs the runtime engine to update the current node.

If the parent node is expanded for the first time, the GUI layer sends a request to the runtime engine to retrieve the records for nodes associated with the expanded parent node. If the parent node was expanded previously, the GUI layer displays the expanded node level, but does not send a request to the runtime engine.

If an expanded parent node is collapsed, the GUI layer displays the closed node, but does not send a request to the runtime engine. If the selected node is part of the collapsed parent node, the GUI layer displays the parent node as selected and sends a request to the runtime engine to change the current record.

Parking on a Node

The runtime engine remains parked on the selected node even when the focus moves from the data tree.

Editing the Node Text

You can change the node text by:

1. Clicking the mouse once to focus on the specified node.
2. Clicking the mouse again to edit the node's string value.
3. Press **ENTER** after the text has been edited. The updated text will be saved and the runtime focus will remain on the selected node. If any other key is pressed, the edited text will not be saved.

Refreshing the Data Tree

The data tree content is refreshed when:

- A parent or child node has been added, modified, or deleted.
- A Screen Refresh event is invoked.

Any other task modification only changes how the data tree is displayed.

Selecting Multiple Nodes

You can select multiple nodes only when the Multi-Marking property is set to **Yes**.

Tree Control Functions

The Tree control functions are:

TreeLevel - Retrieves the current level of the selected node in the data tree.

TreeValue - Retrieves the node identification number determined by the current level of the selected node.

For more information, see the TreeLevel and TreeValue functions in Chapter 8, Expression Rules.

Drag and Drop

Drag and drop functionality is supported within an eDeveloper application and between an eDeveloper application and other applications running on the same machine. Its main purpose is for passing information within an eDeveloper application and between applications by dragging and dropping, which have become a standard in Windows applications.

The Allow Drag control property determines the ability to perform a drag operation, and the Allow Drop control property determines the ability to perform a drop operation.

Drag Begin Event

When the left mouse button is clicked and held down on a control whose Allow Drag property has been set to Yes, the control can be dragged from side to side.

When a drag operation begins, the eDeveloper engine can automatically set the drag operation data according to the type of control on which the drag operation is performed, as described in the table below:

Control Type	Data	Format
Edit	Marked text	Text
Text	No data is set automatically	N/A
Push button	No data is set automatically	N/A
Check box	No data is set automatically	N/A
Radio button	Displayed value of selected item	Text
Tab	Displayed value of selected item	Text

Control Type	Data	Format
List box	Displayed value of selected item	Text
Combo box	Displayed value of selected item	Text
Static controls (Group, Rectangle, etc.	No data is set automatically	N/A
Table	No data is set automatically	N/A
Image	No data is set automatically	N/A
Tree Control	No data is set automatically	N/A

A user-defined handler can be defined to handle the drag begin event so that you can handle the task data and affect the drag operation in the following ways:

- Control the drag operation data by using the DragSetData function
- Control the drag operation icon by using the DragSetCsr function

For more information, see the description of these functions in the Functions section on page 553.

When performing a drag operation on a table control where the records are marked, a handler written for the drag begin event is executed for every marked record.

Drop Event

Releasing the left mouse button over a control where the Allow Drop property has been set to Yes allows the control to be dropped. A user-defined handler can be defined so that you can handle the task data and affect the drop operation in the following ways:

- Check if a certain format is supported by the current drag and drop operation using the DropFormat function.
- Retrieve the data of a defined format using the DropGetData function.
- Retrieve the Mouse coordinates at the time of the DROP operation using the DropMouseX and DropMouseY functions.

For more information, see the description of these functions in the Functions section on page 553.

Drag and Drop Limitations and Environment Settings

A disabled control cannot be dragged, but a non-parkable control can be dragged.

When the Propagate property of the top user-defined drag begin event handler has been set to No, automatic handling of the drag data cannot be performed.

When the Propagate property of the top user-defined drop event handler has been set to No, automatic handling of the drop data cannot be performed.

eDeveloper automatically only handles the drag operation data for Edit controls by updating the control where the dropping occurs.

The Allow Drag and Allow Drop properties are available for the controls listed below:

- Form (for Allow Drop only)
- Edit
- Text
- Push button
- Check box
- Radio button
- Tab
- List box
- Combo box
- Static (Group, Rectangle, etc.)
- Table
- Image
- Tree

The Allow Drag and Allow Drop properties are not available for the controls listed below:

- Form (not available for Allow Drag)
- Slider
- RTF Edit
- RTF
- OLE
- Active-X

You must let the eDeveloper engine know the names of the user-defined formats to be retrieved within the scope of an active drop event as a result of a drag operation.

The Drop Data supported user formats environment setting lets you set the names of the expected user-defined formats. For more information on this

setting, see the Drop Data Supported User Format environment setting in Chapter 2, Settings.

The GUI Display Variable Palette

The Variable palette is accessed by clicking the Variables tab next to the Controls tab on the Control palette.

The Variable palette displays a list of the available variables for the task and its ancestors. Each entry contains the variable's letter identifier, the variable's name, and indicates if it is a real variable, virtual variable, or parameter.

When crossing task boundaries, a separator with the task name appears.

You can drag and drop a variable onto the form in the same way you add controls to the form. You can drop a variable onto a table control to create a corresponding column title.

Press **ESC** or click a task name separator to cancel a variable tool selection.

You can easily resize the Variable palette by dragging the palette's edges. The size of the Variable identifier column will not be affected by resizing, and you will also not be able to reduce the palette's size to less than the minimum size required for the Control palette. Resizing the Variable palette will not affect the Control palette.

GUI Display Control Properties

After you have entered a control into a form, you can assign properties to the control. Select the control with the mouse and press **F5** or double-click to zoom to the Control Properties sheet.

The Control Properties sheet lets you assign variables as Data properties for controls, and to define properties for their format. It is

important to note that the tab order in which eDeveloper accesses the variables at runtime is specified by the order of entries in the Task Operation repositories and not by the sequence of controls on the form.

If a variable is chosen as a Data property in more than one control in the window layout, only the last selection is updated at runtime. If it is necessary to show a variable more than once within a window layout, you must create a virtual variable for each extra occurrence of that variable, and then display the virtual variable.

When you are creating a text-based form (Class > 0), control properties that are not supported in text-based forms are disabled. This is true also for Java forms and HTML forms.

Control properties are organized into separate categories, varying with each type of control. The properties in each list also vary as appropriate to the control. The GUI control properties are listed below.

Radio Button Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Data - Data can be defined either as Variable (Value) or Expression.
- Variable Name - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- Items List - Use this property to enter a string that defines the available options that can be selected from a list box control. The options are concatenated one after the other, where each option is delimited by a comma character.

If the Display List property is not defined, each string of each item is used both for the display and for the returned value. You may set a display list

by using the Display List property to differentiate between the displayed string and the returned string.

This property supports the As Data mode. This mode instructs the control to use the Range property of the associated field as the Item List value.

- **Display List** - You can enter options separated by a comma that are displayed for the control but are not return values. The Display List values must be entered in the same order as the return values entered for the Items List property.

For example, If you enter 1, 2, 3 in the Items List property field, where 1 is red, 2 is blue, and 3 is green, the red, blue, and green string must be entered in the order of 1, 2, 3.

- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CTRLNAME function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Source Table** - You can display the values of a table selected from the Tables list or defined as an expression from the Expression Rules repository. Table values can be combined with Items List values. When a source table is selected, you must add Display List values that represent the table values.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by

defining a handler over the Drag begin event, or automatically by the eDeveloper engine.

- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.
- **Source Table** - The table name as it appears in the Table repository. You can select the table from the Table list or define an expression from the Expression Rules repository. Valid only for List and Combo box controls.
- **Display Field** - The field name from the selected table. You can select the field name for the selected main table from the Variable list. Note that you must first select the Source table.
- **Linked Field** - The field name from the selected table. You can select the field name for the selected linked table from the Variable list. Note that the variables from the main table appear on the Variable list when a linked table is not defined.
- **Index** - An index name from the selected table. You can select an index that is defined for the Source table.
- **Field Ranges** - You reach the Field Range list by zooming from the Field Ranges property in the Control Properties sheet of a Combo Box or List Box control. When a source table is defined for the list box or combo box, you can define a range of values for each field to limit the range of

displayed options. If the Source table does not have a range of values defined for a field, none will appear in the Field Range list.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.
- **Modify in Query** - Lets you toggle from the following choice controls even when the task is in Query mode.
 - Radio button
 - Tab
 - List box
 - Combo box

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken (Default)

- 2-Dimensional
- Border Style - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the control with a thick line. No Border - Leaves the control without a visible border. The default setting varies from one control type to another.
- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Choice Columns - Applies to Radio Button controls. Defines the number of columns displayed in a Radio Button control.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0, the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer- If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent

control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Rectangle Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Detail

- **Text** - Specifies the text that appears on Static controls and Check Box controls.
- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CTRLNAME function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Enable RTF** - A Yes value enables the Rich Text format, which allows for better text resolution during development.
- **Static Type** - You can zoom on Static Type in the Properties sheet to display the Static Control Types Values dialog. The choices in this dialog appear in combo boxes. Explanations of the properties are as follows: NE-SW Line -

Inserts a diagonal line from the upper right hand corner down to the lower left hand corner of the Static control.

- NW-SE Line - Inserts a diagonal line from the upper left hand corner down to the lower right hand corner of the Static control.
- Horizontal Line - Defines the Static control as a horizontal line.
- Vertical Line - Defines the Static control as a vertical line.
- Rectangle - Defines the Static control as a rectangle.
- Rounded Rectangle - Defines the Static control as a rounded rectangle.
- Ellipse - Defines the Static control as an ellipse.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- Allow Drop - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- Multi-line Edit - This property specifies whether the Edit control can contain more than one line of text. When using a Multi-line Edit control on a text-based form, the text can be enlarged to fit the entire text. This control can be enlarged to the size of a page and span multiple pages. Multi-line Edit controls that are over a page length continue on the next page with all of

eDeveloper's header and footer mechanisms. If the control is in a table, the table cell enlarges to fit the edit control size. The title of the table prints on each page. Other controls appear either above or below the edit control.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised (Default)
 - 3-Dimensional Sunken
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the

control with a thick line. No Border - Leaves the control without a visible border. The default setting varies from one control type to another.

- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Vertical Alignment - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.
- Line Style - Defines the appearance of lines in a Static control. The valid values are Regular, Dotted, Dashed, DashedDot, and DashDotDot.
- Line Width - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.
- 3D Line - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Table Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Keep Width - When set to Yes, the table keeps its set width even when columns are hidden or displayed. When set to No, the table does not keep its set width, but will change the width of the table as columns are hidden or displayed. Table behavior for the Keep Width property is:
 - When Keep Width = No and a column is hidden, all columns in the table shift by the width of the hidden column, and the width of the table is decreased by the width of the hidden column.
 - When Keep Width = No and a column is displayed, all columns in the table shift by the width of the displayed column, and the width of the table is increased by the width of the displayed column.

- When Keep Width = Yes and column is hidden, the consecutive column shifts to the position of the hidden column and is enlarged by the size of the hidden column. If the hidden column is the last column, the last divider shifts to the position of the hidden column, leaving blank space between the last column and the table slider.
- When Keep Width = Yes and a column is displayed, the consecutive column shifts by the width of the displayed column. Its width is decreased by the width of the displayed column. If the width of the displayed column is greater than that of the consecutive column, the consecutive column is set to a minimal width of 20 pixels, and the remaining width of the displayed column is taken from the next column. If the displayed column is greater than the width of the next column, this column is also be reduced to 20 pixels, and the column after will also be reduced. If the last column is reduced, but the width of the displayed column has not been distributed, the same procedure is performed for the column that precedes the displayed column up to the first column of the table. If the first column has been reduced, and the displayed column still has excess width, the width of the displayed column will be the original column width minus the remaining width. Note that the column width used for the above calculation is always the width set for the column in the form editor, and not by a runtime evaluated expression. After the calculation, the columns are aligned; only then is their width recomputed by the expression.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by

defining a handler over the Drag begin event, or automatically by the eDeveloper engine.

- Allow Drop - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Appearance

- Color - You can zoom from the Value column to the Color list and select a color. When the Set Table Color property is set **By Column**, The Color property is enabled for the 2D style only. When the Set Table Color property is set **By Table**, the Color property is enabled for all styles.
- Set Table Color - This property lets you set the background color either for each column or set the background color for the entire table.

When **By Column** is selected, the option specified in the Color property affects the column background and the foreground of the column title for the 2D tables.

When **By Table** is selected, the Table control Color property is enabled for all styles, and the color assigned to a column is disabled. When the table option is selected the selected color affects the following table styles:

- 2D - Column header backgrounds, empty space backgrounds, and column backgrounds are affected by the background color set in the Color property. The column titles and table dividers are affected by the foreground color set by the Color property.
- 3D Raised - The column title is affected by the foreground color set by the Color property.
- Windows 3D - The column background and the empty space background are affected by the background color set by the Color

property. The columns title are affected by the foreground color set by the Color property.

- Alternating Background Color - This property lets you set the color of an alternating row for a table. The default value is zero (0) and eDeveloper accepts zero as a valid value. This property does not support a dynamic value used in an expression. The Alternating Background Color property is disabled when the Set Table Color property is set By Column.

In runtime, the row background switches from the table color to the alternating color from one record to another. In the order of appearance, odd records display the table color and even records display the alternate color. When records are added or deleted, the background color of the affected records change depending whether they have become even or odd. When the table is not full, the table color and alternate color are displayed for the remaining rows.

- Tooltip - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- Visible - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- Style - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised (Default)
 - 3-Dimensional Sunken
 - 2-Dimensional
- Border Style - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the control with a thick line.No Border - Leaves the control without a visible border. The default setting varies from one control type to another.
- Scroll Bar - This property determines whether or not to display a horizontal scroll bar and a vertical scroll bar according to the properties Horizontal

Scroll and Vertical Scroll. Select Yes to display the scroll bars defined for an Edit control.

- Line Divider - Applies to a Table control. Defines whether the table displays a divider between its rows.
- Row Highlight Style - This property lets you determine the style by which the table highlights its current row.

The options are:

- None - The current row will not be highlighted.
- Frame - The current row is highlighted by a rectangular box surrounding the row. The color of the rectangle frame color is the foreground color defined in the Highlight Color property. When the Line Divider = No, the rectangle frame is not displayed.
- Background - A current row is highlighted by a background color specified in the Highlight Color property.
- BG & Controls - The current row is highlighted by a background color just like the Background option. This mode also modifies the color of all controls in the current row to display highlighted color.
- Row Highlight Color - This property sets the highlight color according to the Highlight style property.
- Title Height - Defines the title height of a Table control. You can also set this property by clicking and dragging the lower border of the title bar in the Form Editor.
- Row Height - Defines the row height of a Table control. You can also set this property by clicking and dragging the lower edge of the first row of the table.
- Bottom Position Interval - This property sets the interval by which the bottom position of the table is set within the defined height of the table.

When this property is set to:

- Row Height (Default) - The bottom position displays only full rows in the Table control.

- None - Partial rows are displayed and the bottom position matches the full height of the table.
- Columns - Displays the number of columns in a Table control. The number of columns can only be changed from the form layout.
- Column Divider - Defines whether the table displays a divider between its columns.
- Last Divider - When set to No the indication of the last divider (on the table title) will not be displayed. It is recommended to display the line divider in GUI Display forms and to hide it (set to No) in GUI Output forms. The default for this property follows the recommendation: Yes for GUI Display tables, No for GUI Output tables.
- Table in Window - Displays the Table control in an embedded window. This option allows the table to be larger than the display area. When a table in window exceeds the display size, a horizontal scroll bar appears to enable the user to scroll through the columns of the table.
- Allow Column Resize - When set to Yes, the end user will be able to change the size of each column by dragging the adjacent column divider. A column can be resized in two ways: 1. Clicking on a column divider and dragging it to a different location - This action modifies the width of the columns on both sides of the divider. 2. Clicking on a column divider, pressing the SHIFT key and dragging it to a different location. This action modifies the width of the column positioned on the left side of the column divider (in RTL applications it will be for the column on right) and the table width will be affected accordingly.
- Multi-Marking - When set to Yes, the end user will be able to multi mark various rows displayed in the table, perform user defined logic on all the

marked rows, and use the eDeveloper internal delete operation on all the marked rows.

Navigation

- Placement - The Placement property determines whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions.
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears

disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Edit Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Data** - Data can be defined either as Variable (Value) or Expression. For an OLE control, however, Data must be defined as Blob.
- **Variable Name** - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Format** - Present for Edit and Push Button controls. Specifies the format associated with this variable. The format can be modified here for the current form display only. The initial setting is the Picture inherited from the item's Column repository setting. You can specify the value of this property at runtime by zooming to the Expression Rules repository and

entering an expression that evaluates to the coordinates of the control's format.

- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when

the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.

- Multi-Line Edit - This property specifies whether the Edit control can contain more than one line of text. When using a Multi-line Edit control on a text-based form, the text can be enlarged to fit the entire text. This control can be enlarged to the size of a page and span multiple pages. Multi-line Edit controls that are over a page length continue on the next page with all of eDeveloper's header and footer mechanisms. If the control is in a table, the table cell enlarges to fit the edit control size. The title of the table prints on each page. Other controls appear either above or below the edit control.
- Horizontal Scroll - Applies to controls with Multi-line Edit set to Yes. Allows horizontal scrolling.
- Vertical Scroll - Applies to controls with Multi-line Edit set to Yes. Allows vertical scrolling between the lines of the control.
- Show Scroll Bars - This property determines whether or not to display a horizontal scroll bar and a vertical scroll bar according to the properties Horizontal Scroll and Vertical Scroll. Select Yes to display the scroll bars defined for an Edit control.
- Allow CR in Data - Applies to controls with Multi-line Edit set to Yes. Specifies whether eDeveloper will exit the control or move down one line when the end-user presses ENTER.
- Expansion Window - Applies only to the Edit control. Allows you to define a window for text that extends beyond the visible limits of the control. Zoom

to the Form list to choose a form for the expansion window associated with the control.

- Password Edit - Specifies whether access to this control is password-protected. The options are: No (Default) - Access is not password-protected. Yes - Access is password-protected.
- Viewed Currency - Lets you access a currency value from the European Currency Conversion table. Zoom from the Viewed Currency property to select a currency value from the Currency list. The Viewed Currency property is active only for the Edit control with a numeric attribute.

Appearance

- Font - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- Color - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- Help Screen - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- Tooltip - Specifies whether a Tooltip is associated with this control. A number specifies the number of the Tooltip Help in the Help list.
- Help Prompt - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- Visible - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical

value makes the control visible. A 'FALSE' logical value makes the control invisible.

- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - Windows 3-D (Default)
 - 3-Dimensional Raised
 - 3-Dimensional Sunken
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the control with a thick line. No Border - Leaves the control without a visible border. The default setting varies from one control type to another.
- **Horizontal Alignment** - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- **Vertical Alignment** - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.

Navigation

- **Placement** - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the

Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Column Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Column Title** - Enter the column title in a table control.
- **Sortable** - If this property is set to Yes, the values in the table column can be sorted.
- **Marking Column** - If this property is set to Yes, the values in the table column can be marked.

Appearance

- **Top Border** - If this property is set to Yes, a top border is displayed for the table column.
- **Right Border** - If this property is set to Yes, a right border is displayed for the table column.
- **Horizontal Alignment** - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- **Vertical Alignment** - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.
- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical

value makes the control visible. A 'FALSE' logical value makes the control invisible.

Navigation

- **Placement** - The Placement property determines whether the column is resized with the resizing of the Table control. The property options are Yes or No. You can also insert an expression with a Boolean value that is evaluated at runtime.
- **Width** - The initial width of the control in units of measurement.
- **Control's Layer** - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Tab Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Data** - Data can be defined either as Variable (Value) or Expression.
- **Variable Name** - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- **Items List** - Use this property to enter a string that defines the available options that can be selected from a list box control. The options are

concatenated one after the other, where each option is delimited by a comma character.

If the Display List property is not defined, each string of each item is used both for the display and for the returned value. You may set a display list by using the Display List property to differentiate between the displayed string and the returned string.

This property supports the As Data mode. This mode instructs the control to use the Range property of the associated field as the item's list value.

- **Display List** - You can enter options separated by a comma that are displayed for the control but are not return values. The Display List values must be entered in the same order as the return values entered from the Items List property.

For example, If you enter 1, 2, 3 in the Items List property field, where 1 is red, 2 is blue, and 3 is green, the red, blue, and green string must be entered in the order of 1, 2, 3.

- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other

applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

- **Source Table** - You can display the values of a table selected from the Tables list or defined as an expression from the Expression Rules repository. Table values can be combined with Items list values. When a source table is selected, you must add Display List values that represent the table values.
- **Display Field** - The field name from the selected table. You can select the field name for the selected main table from the Variable list. Note that you must first select the Source table.
- **Linked Field** - The field name from the selected table. You can select the field name for the selected linked table from the Variable list. Note that the variables from the main table appear on the Variable list when a linked table is not defined.
- **Index** - An index name from the selected table. You can select an index that is defined for the Source table.
- **Field Ranges** - You reach the Field Range list by zooming from the Field Ranges property in the Control Properties sheet of a Combo Box or List Box control. When a source table is defined for the list box or combo box, you can define a range of values for each field to limit the range of

displayed options. If the Source table does not have a range of values defined for a field, none will appear in the Field Range list.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are:
 - **After** - specifies that when the called select program terminates, the cursor moves to the next property.
 - **Before (Default)** - specifies that when the called select program terminates, the cursor returns to the property.
 - **Prompt** - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.
- **Modify in Query** - Lets you toggle from the following choice controls even when the task is in Query mode.
 - Radio button
 - Tab
 - List box
 - Combo box

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised (Default)
 - 2-Dimensional

- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Tab Control Side - Specifies where the tabs appear for a Tab control: Top, Right, Bottom, or Left.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When the Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears

disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Ellipse Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Text - Specifies the text that appears on Static controls and Check Box controls.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- Enable RTF - A Yes value enables the Rich Text format, which allows for better text resolution during development.
- Static Type - You can zoom on Static Type in the Properties sheet to display the Static Control Types Values dialog. The choices in this dialog appear in combo boxes. Explanations of the properties are as follows: NE-SW Line - Inserts a diagonal line from the upper right hand corner down to the lower left hand corner of the Static control.

- NW-SE Line - Inserts a diagonal line from the upper left hand corner down to the lower right hand corner of the Static control.
- Horizontal Line - Defines the Static control as a horizontal line.
- Vertical Line - Defines the Static control as a vertical line.
- Rectangle - Defines the Static control as a rectangle.
- Rounded Rectangle - Defines the Static control as a rounded rectangle.
- Ellipse - Defines the Static control as an ellipse.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- Allow Drop - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- Multi-line Edit - This property specifies whether the Edit control can contain more than one line of text. When using a Multi-line Edit control on a text-based form, the text can be enlarged to fit the entire text. This control can be enlarged to the size of a page and span multiple pages. Multi-line Edit controls that are over a page length continue on the next page with all of eDeveloper's header and footer mechanisms. If the control is in a table, the

table cell enlarges to fit the edit control size. The title of the table prints on each page. Other controls appear either above or below the edit control.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are: Windows
 - 3-Dimensional Raised (Default)
 - 3-Dimensional Sunken
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the

control with a thick line. No Border - Leaves the control without a visible border. The default setting varies from one control type to another.

- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Vertical Alignment - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.
- Line Style - Defines the appearance of lines in a Static control. The valid values are Regular, Dotted, Dashed, DashedDot, and DashDotDot.
- Line Width - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.
- 3D Line - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Image Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Data - Data can be defined either as Variable (Value) or Expression.
- Variable Name - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing

blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.

- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Default Image File** - The Default Image File Name property specifies the name of the bitmap file for display on an image or push button.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by

defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are:
 - **After** - specifies that when the called select program terminates, the cursor moves to the next property.
 - **Before (Default)** - specifies that when the called select program terminates, the cursor returns to the property.
 - **Prompt** - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.

Appearance

- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the

Help list. Zoom again to create a new Help screen entry in the Help repository.

- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are:
 - Thin - Surrounds the control with a thin line.
 - Thick - Surrounds the control with a thick line.
 - No Border - Leaves the control without a visible border. The default setting varies from one control type to another.
- **Image Style** - Controls the way an image is fitted into an Image control. The valid values are:
 - Tiled - The image is copied onto the control as many times as needed to fill its entire area.
 - Copied - The image is copied onto the control as is. If the image is larger than the control, it is cropped. If it is smaller than the control, the uncovered part of the control is painted with the background color.

- Scaled to Fit - The entire image is scaled onto the control while maintaining the original aspect ratio of the image. This may result in part of the control remaining uncovered, unless the aspect ratios of the image and the control are identical.
- Scaled to Fill - The image is scaled to fill the entire control while maintaining the original aspect ratio of the image. This may result in only part of the image being seen on the control, unless the aspect ratios of the image and the control are identical.
- Distorted Scaling - The image is shrunk or stretched to fill the entire area of the control. The resulting image will be distorted, unless the aspect ratios of the image and the control are identical.
- Image Effects - Optionally add special video display effects to an image on an Image control. Click on the combo box to select an effect from the list of options. The default is Normal (no special effect).

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Text Control Properties

Details

- Text - Specifies the text that appears on Static controls and Check Box controls.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- Enable RTF - A Yes value enables the Rich Text format, which allows for better text resolution during development.
- Static Type - You can zoom on Static Type in the Properties sheet to display the Static Control Types Values dialog. The choices in this dialog appear in combo boxes. Explanations of the properties are as follows:
 - NE-SW Line - Inserts a diagonal line from the upper right hand

corner down to the lower left hand corner of the Static control.

- NW-SE Line - Inserts a diagonal line from the upper left hand corner down to the lower right hand corner of the Static control.
 - Horizontal Line - Defines the Static control as a horizontal line.
 - Vertical Line - Defines the Static control as a vertical line.
 - Rectangle - Defines the Static control as a rectangle.
 - Rounded Rectangle - Defines the Static control as a rounded rectangle.
 - Ellipse - Defines the Static control as an ellipse.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
 - Allow Drop - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- Multi-line Edit - This property specifies whether the Edit control can contain more than one line of text. When using a Multi-line Edit control on a text-based form, the text can be enlarged to fit the entire text. This control can be enlarged to the size of a page and span multiple pages. Multi-line Edit controls that are over a page length continue on the next page with all of

eDeveloper's header and footer mechanisms. If the control is in a table, the table cell enlarges to fit the edit control size. The title of the table prints on each page. Other controls appear either above or below the edit control.

Appearance

- Font - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- Color - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- Help Screen - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- Tooltip - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- Visible - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- Enabled - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- Style - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised (Default)
 - 3-Dimensional Sunken
 - 2-Dimensional

- **Border Style** - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the control with a thick line. No Border - Leaves the control without a visible border. The default setting varies from one control type to another.
- **Horizontal Alignment** - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- **Vertical Alignment** - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.
- **Line Style** - Defines the appearance of lines in a Static control. The valid values are Regular, Dotted, Dashed, DashedDot, and DashDotDot.
- **Line Width** - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.
- **3D Line** - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.

Navigation

- **Placement** - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

List Box Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Data - Use this property to define data information as either Variable (Value) or Expression.
- Variable Name - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- Items List - Use this property to enter a string that defines the available options that can be selected from a list box control. The options are

concatenated one after the other, where each option is delimited by a comma character.

If the Display List property is not defined, each string of each item is used both for the display and for the returned value. You may set a display list by using the Display List property to differentiate between the displayed string and the returned string.

This property supports the As Data mode. This mode instructs the control to use the Range property of the associated field as the item's list value.

- **Display List** - You can enter options separated by a comma that are displayed for the control but are not return values. The Display List values must be entered in the same order as the return values entered for the Items List property.

For example, If you enter 1, 2, 3 in the Items List property field, where 1 is red, 2 is blue, and 3 is green, the red, blue, and green string must be entered in the order of 1, 2, 3.

- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other

applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

- **Source Table** - You can display the values of a table selected from the Tables list or defined as an expression from the Expression Rules repository. Table values can be combined with Items list values. When a source table is selected, you must add Display List values that represent the table values.
- **Display Field** - The field name from the selected table. You can select the field name for the selected main table from the Variable list. Note that you must first select the Source table.
- **Linked Field** - The field name from the selected table. You can select the field name for the selected linked table from the Variable list. Note that the variables from the main table appear on the Variable list when a linked table is not defined.
- **Index** - An index name from the selected table. You can select an index that is defined for the Source table.
- **Field Ranges** - You reach the Field Range list by zooming from the Field Ranges property in the Control Properties sheet of a Combo Box or List Box control. When a source table is defined for the list box or combo box, you can define a range of values for each field to limit the range of displayed options. If the Source table does not have a range of values defined for a field, none will appear in the Field Range list.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.
- **Selection Mode** - Determines whether the List Box control supports Single or Multiple item selections for Browser and GUI Display forms:
 - **Single** - Only a single item can be selected.
 - **Multiple** - More than a single item can be selected. The key combinations below let the end-user select single or multiple items.

The key combinations below let the end-user select single or multiple items.

Click	Selects the clicked item and deselects all other items.
Ctrl+Click	Selects or deselects the clicked item without affecting the other selected items.
Shift+Click	Selects all the items from the last selected item to the clicked item.

Shift+Up Arrow	Selects the clicked item and all items below it.
Shift+Down Arrow	Selects the clicked item and all items above it.
Up Arrow	Deselects all items and selects the item before the currently selected item.
Down Arrow	Deselects all the items and selects the item after the currently selected item.

The Multiple selection option is available for List Box controls that have an Alpha, Memo, RTF Blob, or Vector Attribute property value. The default value for this property is Single.

When the Selection Mode = Single, the selected item is updated by the specified data attribute value determined by the label or the linked field in the data-bound list box.

If the specified data attribute value is Vector, the selected value is stored in the vector as the value of the first cell. The size of an updated vector reflects the total number of selected items. For a Single selection, the List box data property, defined as a Vector, returns 1.

When the Selection Mode = Multiple, the specified attribute value is displayed as series of selected values determined by the label or linked field, separated by commas. The order of the concatenated result string is by the order of the items as they appear in the list and not by the order of selection.

Layering controls in a List Box is not supported for a List Box set as a Multiple selection. However, the form editor does not prevent the developer from attaching another control to a List Box layer when the List box is set as a Multiple selection.

- Modify in Query - Lets you toggle from the following choice controls even when the task is in Query mode.
 - Radio button
 - Tab
 - List box
 - Combo box

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken (Default)

- 2-Dimensional
- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Choice Columns - Defines the number of choice columns displayed in a list or radio button control.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears

disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Line Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- Allow Drop - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by

defining a handler over the Drop event, or automatically by the eDeveloper engine.

Appearance

- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken
 - 2-Dimensional
- **Line Style** - Defines the appearance of lines in a Static control. The valid values are Regular, Dotted, Dashed, DashedDot, and DashDotDot.
- **Line Width** - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.

Navigation

- **Placement** - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement

- Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- X1 - The x-coordinate of the left point of a selected line control.
- Y1 - The y-coordinate of the left point of a selected line control.
- X2 - The x-coordinate of the right point of a selected line control.
- Y2 - The y-coordinate of the right point of a selected line control.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

OLE Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Data** - Data can be defined either as Variable (Value) or Expression. For an OLE control, however, Data must be defined as Blob.
- **Variable Name** - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression. Note: OLE controls do not support Allow Drag and Drop properties.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical

value makes the control visible. A 'FALSE' logical value makes the control invisible.

- Enabled - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears

disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

OLE

- **OLE Class** - The object class that is to be inserted into the property. Leave the OLE Class value field empty to allow the end-user to select any available object type from the Windows Insert Object dialog during runtime. To specify an object type for the end-user, zoom from the OLE Class property and click No in the Active X dialog, which opens the Windows Select Object Type dialog. When the end-user selects Edit/Insert Object, the application serving the developer-specified class is opened. Note that if the end-user does not have an application available that supports the specified object class, a Windows error message appears.
- **Display OLE As** - There are three choices available in the combo box list:
 - Icon - When the OLE object is installed but not edited, an icon describing the type of information that is stored in the container is displayed.
 - Content - When the OLE object is installed but not edited, a snapshot of the file's contents is displayed in the container.
 - Any - Gives the end-user the option to display the object as an icon, or to display the object's contents as a snapshot. The choice is made in the Insert Object dialog by selecting the Display As Icon check box.
- **Store OLE As** - There are three choices available in the combo box list:
 - Linked - The OLE object will be kept as a linked object in runtime.
 - Embedded - The OLE object will be kept as an embedded object in runtime.
 - Any - The end-user has the option to keep the object as a linked object or as an embedded object. This option is exercised in the Windows Insert Object dialog by selecting the Link check box.
- **Auto Link Update** - This property enables automatic updating of a linked document's content bitmap. This option is relevant when Display OLE As is set to Content, Icon, or Any, and when Store OLE As is set to Linked or Any. Yes - eDeveloper verifies that the content image is the most current before displaying it on the form. No - The last saved content image is displayed.

- Use OLE Frame Type - When set to Yes, eDeveloper displays a different frame style around the object, depending on the object's storage type.

Push Button Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Data - Data can be defined either as Variable (Value) or Expression.
- Variable Name - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- Format - Present for Edit and Push Button controls. Specifies the format associated with this variable. The format can be modified here for the current form display only. The initial setting is the Picture inherited from the item's Column repository setting. You can specify the value of this property at runtime by zooming to the Expression Rules repository and entering an expression that evaluates to the coordinates of the control's format.
- Attribute - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other

applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.

- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

- **Button Style**

Push button - When the Button Style property is set to Push Button, the Push Button control uses the default Windows Push Button design. The conditions for defining a variable name for a Push button style button are limited to:

- A thirty character name
- Leading and trailing blanks
- Blanks within the name
- Numbers and special characters
- Specific language characters as defined in the Language repository

Image Button - When the Button Style property is set to Image button, a bitmap (.bmp) file can be selected. The bitmap must have images that correspond to the four different states of a push button: Parked on, Pressed, Disabled, and Default. During runtime, eDeveloper decides the image to display. The size of the bitmap file is not relevant. You must specify the file name for the bitmap file either in the Data property as a variable or as an expression. The Default Image File Name property displays the specified bitmap only in toolkit.

Text on Image - This toolkit button lets you set an image and a text label as it would appear in runtime according to the static values of the Format and Default Image File properties. The purpose of this option is to let you update the text label without recreating the image. The text format is determined by the Font and Color properties. In runtime, the control data determines the button image, such as Alpha or BLOB, and the format determines the text label. This button type behaves exactly like an image button. The only difference is that the label text is in the center of the button.

Hypertext - When the Push button style is set to Hypertext, an eDeveloper event is linked to the Push Button control. The Push Button control appears as a 2D image with an underline. Zoom from the Return Action property to access the Action list. The conditions for defining a variable name for a Hypertext style button are limited to:

- A 30 character name
- Leading and trailing blanks
- Blanks within the name
- Numbers and special characters
- Specific language characters as defined in the Language repository

- **Default Image File** - The Default Image File Name property specifies the name of the bitmap file for display on a Push button.
- **Raise Event** - Lets you define the type of event that will be raised during the flow of the eDeveloper engine when the button is clicked in runtime.

Input

- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.
- **Park on Click** - This property determines whether the engine can park on the Push Button control.

When set to Yes, the direct activation, clicking the button or pressing the push button accelerator of the push button moves the focus to that control by executing the flow logic from the engine's current location to the push button. Only after the engine has parked on the push button will it raise the event set for that control. The default value is Yes.

When set to No, the direct activation triggers the event set for the push button control without leaving the engine's current location.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - This property is enabled when the button style is set to Text on Image. You can either set the foreground color of the text or define an expression that determines the foreground color at runtime.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical

value makes the control visible. A 'FALSE' logical value makes the control invisible.

- Enabled - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears

disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Combo Box Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Data** - Data can be defined either as Variable (Value) or Expression.
- **Variable Name** - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- **Items List** - Use this property to enter a string that defines the available options that can be selected from a list box control. The options are concatenated one after the other, where each option is delimited by a comma character.

If the Display List property is not defined, each string of each item is used both for the display and for the returned value. You may set a display list by using the Display List property to differentiate between the displayed string and the returned string.

This property supports the As Data mode. This mode instructs the control to use the Range property of the associated field as the item's list value.

- **Display List** - You can enter options separated by a comma that are displayed for the control but are not return values. The Display List values must be entered in the same order as the return values entered from the Items List property.

For example, If you enter 1, 2, 3 in the Items List property field, where 1 is red, 2 is blue, and 3 is green, the red, blue, and green string must be entered in the order of 1, 2, 3.

- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.
- **Source Table** - You can display the values of a table selected from the Tables list or defined as an expression from the Expression Rules repository. Table values can be combined with Items list values. When a

source table is selected, you must add Display List values that represent the table values.

- **Display Field** - The field name from the selected table. You can select the field name for the selected main table from the Variable list. Note that you must first select the Source table.
- **Linked Field** - The field name from the selected table. You can select the field name for the selected linked table from the Variable list. Note that the variables from the main table appear on the Variable list when a linked table is not defined.
- **Index** - An index name from the selected table. You can select an index that is defined for the Source table.
- **Field Ranges** - You reach the Field Range list by zooming from the Field Ranges property in the Control Properties sheet of a Combo Box or List Box control. When a source table is defined for the list box or combo box, you can define a range of values for each field to limit the range of displayed options. If the Source table does not have a range of values defined for a field, none will appear in the Field Range list.

Input

- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow

the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.

- Modify in Query - Lets you toggle from the following choice controls even when the task is in Query mode.
 - Radio button
 - Tab
 - List box
 - Combo box

Appearance

- Font - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- Color - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- Help Screen - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- Tooltip - Specifies whether a Tooltip is associated with this control. A number specifies the number of the Tooltip Help in the Help list.
- Help Prompt - Specifies whether a Help prompt is associated with this control. The default setting is Yes. No indicates that no prompt line is associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen

associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.

- Visible - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- Enabled - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- Style - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken (Default)
 - 2-Dimensional
- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Choice Columns - Applies to Radio Button controls. Defines the number of columns displayed in a Radio Button control.
- Visible Lines - Specifies the number of lines presented when the Combo Box control is accessed.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the

control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Slider Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Data - Data can be defined either as Variable (Value) or Expression.
- Variable Name - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case

sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.

- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Range** - Applies to Slider controls. Defines the range of values for the slider.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.

Appearance

- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the

Help list. Zoom again to create a new Help screen entry in the Help repository.

- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes.No indicates that no prompt line is associated with this variable.If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Slider Style** - Defines the direction of a Slider control. The valid values are Horizontal or Vertical.
- **Slider Step** - Applies to Slider controls. Defines the step size within the Slider Range.

Navigation

- **Placement** - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the

control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Rich Text Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Text - Specifies the text that appears on Static controls and Check Box controls.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing

blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.

- Enable RTF - A Yes value enables the Rich Text format, which allows for better text resolution during development.
- Static Type - You can zoom on Static Type in the Properties sheet to display the Static Control Types Values dialog. The choices in this dialog appear in combo boxes. Explanations of the properties are as follows:
 - NE-SW Line - Inserts a diagonal line from the upper right hand corner down to the lower left hand corner of the Static control.
 - NW-SE Line - Inserts a diagonal line from the upper left hand corner down to the lower right hand corner of the Static control.
 - Horizontal Line - Defines the Static control as a horizontal line.
 - Vertical Line - Defines the Static control as a vertical line.
 - Rectangle - Defines the Static control as a rectangle.
 - Rounded Rectangle - Defines the Static control as a rounded rectangle.
 - Ellipse - Defines the Static control as an ellipse.
- Allow Drag - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- Allow Drop - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by

defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- Multi-line Edit - This property specifies whether the Edit control can contain more than one line of text. When using a Multi-line Edit control on a text-based form, the text can be enlarged to fit the entire text. This control can be enlarged to the size of a page and span multiple pages. Multi-line Edit controls that are over a page length continue on the next page with all of eDeveloper's header and footer mechanisms. If the control is in a table, the table cell enlarges to fit the edit control size. The title of the table prints on each page. Other controls appear either above or below the edit control.

Appearance

- Font - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- Color - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- Tooltip - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- Visible - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- Enabled - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- Style - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised (Default)
 - 3-Dimensional Sunken

- 2-Dimensional
- Border Style - Defines the style of the control's border. The valid values are:
 - Thin - Surrounds the control with a thin line.
 - Thick - Surrounds the control with a thick line.
 - No Border - Leaves the control without a visible border. The default setting varies from one control type to another.
- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Vertical Alignment - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.
- Line Style - Defines the appearance of lines in a Static control. The valid values are Regular, Dotted, Dashed, DashedDot, and DashDotDot.
- Line Width - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.
- 3D Line - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the

Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Check Box Control Properties

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Data** - Data can be defined either as Variable (Value) or Expression.
- **Variable Name** - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- **Text** - Specifies the text that appears on Static controls and Check Box controls.
- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following

controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is

Yes.If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.

- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether the Help Prompt is associated with this control. The default setting is Yes.No indicates that there is not a Help Prompt line associated with this variable.If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken (Default)
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are:
 - Thin - Surrounds the control with a thin line.
 - Thick - Surrounds the control with a thick line.
 - No Border - Leaves the control without a visible border. The default setting varies from one control type to another.

- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears

disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Group Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Text - Specifies the text that appears on Static controls and Check Box controls.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the `CtrlName` function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- Enable RTF - A Yes value enables the Rich Text format, which allows for better text resolution during development.
- Static Type - You can zoom on Static Type in the Properties sheet to display the Static Control Types Values dialog. The choices in this dialog appear in combo boxes. Explanations of the properties are as follows:
 - NE-SW Line - Inserts a diagonal line from the upper right hand corner down to the lower left hand corner of the Static control.
 - NW-SE Line - Inserts a diagonal line from the upper left hand corner down to the lower right hand corner of the Static control.
 - Horizontal Line - Defines the Static control as a horizontal line.
 - Vertical Line - Defines the Static control as a vertical line.
 - Rectangle - Defines the Static control as a rectangle.

- **Rounded Rectangle** - Defines the Static control as a rounded rectangle.
- **Ellipse** - Defines the Static control as an ellipse.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Allow Drop** - The Allow Drop property, which enables the dropping of data onto the control from other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls, (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied onto this control can be set manually by defining a handler over the Drop event, or automatically by the eDeveloper engine.

Input

- **Multi-line Edit** - This property specifies whether the Edit control can contain more than one line of text. When using a Multi-line Edit control on a text-based form, the text can be enlarged to fit the entire text. This control can be enlarged to the size of a page and span multiple pages. Multi-line Edit controls that are over a page length continue on the next page with all of eDeveloper's header and footer mechanisms. If the control is in a table, the

table cell enlarges to fit the edit control size. The title of the table prints on each page. Other controls appear either above or below the edit control.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a Tooltip is associated with this control. A number specifies the number of the Tooltip Help in the Help list.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Style** - Defines the appearance of controls. The valid values are:
 - 3-Dimensional Raised
 - 3-Dimensional Sunken (Default)
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are: Thin - Surrounds the control with a thin line. Thick - Surrounds the

control with a thick line. No Border - Leaves the control without a visible border. The default setting varies from one control type to another.

- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.
- Vertical Alignment - Defines the vertical alignment of text in the control. The valid values are Top, Center, and Bottom.
- Line Style - Defines the appearance of lines in a Static control. The valid values are Regular, Dotted, Dashed, DashedDot, and DashDotDot.
- Line Width - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.
- 3D Line - Defines the width of the line in a Static control. Specify a numeric value based on the measurement units you have defined for the form.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Rich Edit Detail Control Properties

Model

- Model - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- Data - Data can be defined either as Variable (Value) or Expression. For an OLE control, however, Data must be defined as Blob.
- Variable Name - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- Control Name - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing

blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.

- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.

Input

- **Must Input** - Specifies whether the end-user must enter a value into a control. You can set the value of the control at runtime with an expression in the Expression Rules repository.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Select Program** - Specifies whether the end-user can invoke a program at runtime using Edit/Zoom. The default setting is Yes. Specify a Program ID in the Program property or zoom to the Program list to choose a program.
- **Select Mode** - Specifies when eDeveloper calls the program defined in the Select Program property. The valid values are: After - specifies that when the called select program terminates, the cursor moves to the next property. Before (Default) - specifies that when the called select program terminates, the cursor returns to the property. Prompt - specifies that the program is automatically executed prior to entering the property, to allow the user to immediately select the lookup value. In this case, after exiting the Select program, the cursor returns to the property as in 'Before' mode.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the Help list. Zoom again to create a new Help screen entry in the Help repository.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Help Prompt** - Specifies whether the Help Prompt is associated with this control. The default setting is Yes. No indicates that there is not a Help Prompt line associated with this variable. If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Style** - Defines the appearance of controls. The valid values are:
 - Windows 3-D (Default)
 - 3-Dimensional Raised
 - 3-Dimensional Sunken
 - 2-Dimensional

- Horizontal Alignment - Defines the horizontal alignment of text in the control. The valid values are Left, Center, and Right.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

Tree Control Properties

Changing the class number from 0 to 1 causes the Tree control to be deleted.

Model

- **Model** - You can reinherit any broken properties or disinherit all properties for the specified control model. You can also select a different model.

Details

- **Node ID** - A variable value that defines a record in the task dataview. The node cannot be defined from an expression.
- **Parent ID** - A variable value that defines a parent record in the task dataview. The parent node cannot be defined from an expression.
- **Root Value** - The expression value used as the starting point for the runtime engine to build the tree.
- **Show Root** - When Yes is selected, the root value is displayed as the top node. If No is selected, the root value is not displayed but is still considered the top node. If the root value is not specified, this property will not be active.
- **Control Name** - This property enables a robust method to name controls found in a Class 0 form, and to refer to them in functions. See the CtrlName function for more information. The Control name is case sensitive. The Control Name property is limited to 30 characters. Trailing blanks are not allowed. The Copy operation for controls in the Form Editor does not copy this property.
- **Image List File Name** - Click the Ellipse button to select a bitmap (.bmp) image that has images that correspond to each of the following default states of a tree node: Expanded, Collapsed, Parked Expanded, and Parked Collapsed. You can have as many images as you want. The Tree control displays the specified image dynamically. Due to size limitations, each image within the bitmap should be 16 x 16 pixels.
- **Expanded Image Index** - If you have specified a bitmap file in the Image List File Name property, click the Ellipse button to select an from the

images contained in this file. You can also select an expression that dynamically displays one of these images during runtime. If you do not specify an image, the runtime engine assigns a default image.

- **Collapsed Image Index** - If you have specified a bitmap file in the Image List File Name property, click the Ellipse button to select an from the images contained in this file. You can also select an expression that dynamically displays one of these images during runtime. If you do not specify an image, the runtime engine assigns a default image.
- **Parked Expanded Image Index** - If you have specified a bitmap file in the Image List File Name property, click the Ellipse button to select an from the images contained in this file. You can also select an expression that dynamically displays one of these images during runtime. If you do not specify an image, the runtime engine assigns a default image. When a node is not parked on and expanded, the image is not displayed.
- **Parked Collapsed Image Index** - If you have specified a bitmap file in the Image List File Name property, click the Ellipse button to select an from the images contained in this file. You can also select an expression that dynamically displays one of these images during runtime. If you do not specify an image, the runtime engine assigns a default image. The image is not displayed when the engine is not parked on the node and the node is not collapsed.
- **Auto Expand** - If Yes is selected, the data tree is displayed as expanded for every parent node in runtime.
- **Node Preload** - If Yes is selected, the runtime engine loads the data of all the records in the task dataview. If No is selected, the runtime engine only loads the data of expanded records. Note that this property specifies how the data is loaded at runtime. The data tree display is specified by the Auto Expand and Single Expand properties.
- **Allow Drag** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by

defining a handler over the Drag begin event, or automatically by the eDeveloper engine.

- **Allow Drop** - The Allow Drag property, which enables the dragging of data from the control to other controls in the same application or other applications that run on the same machine, applies to the following controls: Edit, Text, Push button, Check box, Radio button, Tab, List box, Combo box, Static controls (Group, Rectangle, etc.), Table, Image, and Tree control. The data copied from this control can be set manually by defining a handler over the Drag begin event, or automatically by the eDeveloper engine.
- **Keep Tree View** - This property instruct the Tree to keep its view and state on reentrance to the tree's task. Setting this property to Yes is usually required for tree controls that are displayed in phantom tasks.

Input

- **Description Variable** - Click the ellipsis button to select variable from the Variable list. Your data can be defined as either as a variable or as an expression.
- **Variable Name** - Defines the variable selected in the current position. To select a variable, enter the variable's identification in the Value column or zoom to choose a variable from the Variable list.
- **Format** - Present for Edit and Push Button controls. Specifies the format associated with this variable. The format can be modified here for the current form display only. The initial setting is the Picture inherited from the item's Column repository setting. You can specify the value of this property at runtime by zooming to the Expression Rules repository and

entering an expression that evaluates to the coordinates of the control's format.

- **Attribute** - The attribute of the selected variable or expression. The Attribute property is fixed for variables. You can change the setting for items based on an expression.
- **Modifiable** - Specifies whether the user can change the value in the control during runtime. A No setting denies the user access to the control. A Yes setting allows the user to modify the value in the control.
- **Enabled** - Defines whether or not the control will be active. Use an expression to set the value at runtime. A 'TRUE' logical value enables the control. A 'FALSE' logical value disables the control.
- **Multi-Marking** - When set to Yes, the end user will be able to multi mark various rows displayed in the table, perform user defined logic on all the marked rows, and use the eDeveloper internal delete operation on all the marked rows.

Appearance

- **Font** - Defines the font for text that appears in a control. If you do not enter a value in this property, the control takes the form's font as defined in the Font property of the Form Properties dialog.
- **Color** - The number of a color in the Color list. This property is valid for 2-Dimensional Style only. You can zoom from the Value column to the Color list and select a color.
- **Visible** - Defines whether or not the control will be visible to the user. Use an expression to set the value dynamically at runtime. A 'TRUE' logical value makes the control visible. A 'FALSE' logical value makes the control invisible.
- **Help Screen** - Specifies whether a Help screen is associated with this control. This property is used for online forms only. The default setting is Yes. If the setting is Yes, tab to the Help property to enter a number that specifies the number of the Help screen. Zoom from this property to the

Help list. Zoom again to create a new Help screen entry in the Help repository.

- **Help Prompt** - Specifies whether a Help prompt is associated with this control. The default setting is Yes.No indicates that no prompt line is associated with this variable.If the setting is Yes, tab to the Prompt's property to enter a number that specifies the number of a Help screen associated with this variable that will automatically appear at runtime. Zoom to the Help list to choose a Prompt line entry.
- **Tooltip** - Specifies whether a ToolTip is associated with this control. A number specifies the number of the ToolTip Help in the Help list.
- **Style** - Defines the appearance of controls. The valid values are:
 - Windows 3-D (Default)
 - 3-Dimensional Raised
 - 2-Dimensional
- **Border Style** - Defines the style of the control's border. The valid values are:
 - Thin - Surrounds the control with a thin line.
 - Thick - Surrounds the control with a thick line.
 - No Border - Leaves the control without a visible border.The default setting varies from one control type to another.
- **Show Buttons** - If Yes is selected, the button image next to a parent node is displayed. If No is selected, the button image is not displayed and the end-user must double-click the parent node to expand the level.
- **Show Lines** - If Yes is selected, a line is displayed identifying the relationship between parent and child nodes. If No is selected, the line is

not displayed. When this property is set to Yes, the Row Highlight property will not be enabled.

- Lines at Root - If Yes is selected, a line is displayed identifying the relationship between the root value and the next level of nodes in the data tree at runtime. If No is selected, the line is not displayed.
- Mouseover Indication - If Yes is selected, a line appears under the node text when the cursor rests on or passes over the node. If No is selected, eDeveloper does not indicate that the cursor rests on or passes over a node.
- Row Highlight - If Yes is selected, the node row is highlighted in blue when the cursor rests on or passes over it. If this property is set to Yes, the Show Line property is not enabled.

Navigation

- Placement - The Placement property controls whether or not controls are resized when a form is resized. You can set the placement coordinates for the following positions:
 - Left placement
 - Right placement
 - Top placement
 - Bottom placement

When a control's Placement property equals 0, the relative size of the control does not change when the form size changes in runtime. When the Placement property is > 0 , the relative size changes when the form size changes in runtime. Note that a control with placement properties will not resize to a size smaller than its original size.

- Left - The initial x-coordinate of the upper left corner of a selected control.
- Top - The initial y-coordinate of the upper right corner of a selected control.
- Width - The initial width of the control in units of measurement.
- Height - The initial height of the control in units of measurement. The Table control has only Left, Right, and Top properties.
- Control's Layer - If the selected Choice control is linked as a child to a parent, the Control's Layer property indicates to which of the parent control's choice layers the selected control is linked. For example, assume that a combo box that has three selections in its menu is on the form layout. If you link an edit control to the third selection, then the number 3 displays in the Edit Control's Layer property. This property always appears disabled, and can only be changed from the form layout. This property is relevant only for controls that are linked to Choice controls.

There are two kinds of Output forms: Internet forms and Report forms. Internet forms can use any of the four Internet interface types: HTML, Frame Set, HTML Merge, and Web Online Response. Report forms can use either a GUI Output or a Text-based interface type. Output forms, which are always based on batch tasks, must have their class defined as >0.

In this chapter:

• HTML Forms
• Frame Set Forms
• HTML Merge Forms
• Web Online Response
• GUI Output and Text-based Report Forms

HTML Forms

HTML forms must be defined as Class > 0. When an HTML form is displayed, all forms of the same Class are displayed on the same form.

Because HTML forms are row-oriented, the GUI functionality provided by the Form Editor may not always be implemented during runtime.

HTML forms let the end-user submit variables to the calling application, or to where the hyperlink settings specify, through the Web server.

HTML Form Display

The eDeveloper HTML Form Editor attempts to display forms in the same way the forms will actually be displayed on the Web browser. However, some factors which affect form display, such as the actual font used or the exact placement of controls, are beyond eDeveloper's control and are determined by the specific Web browser used.

HTML Control Placement

HTML imposes limitations on how controls can be placed on the form:

- If a control's height is more than one row, all the rows are treated as one logical row. The highest control defines the size of the logic row. The logical row's upper and lower borders are displayed when a control is dropped on the form. All controls in a logical row are automatically aligned to the bottom of the row.
- Pre-formatted controls can be placed anywhere on the form. The appropriate padding is automatically added when the form is converted to HTML.
- Other controls can only be placed at the beginning of each row, or immediately following the last control on a row that already has a control on it. A control dropped at the upper left corner of a form is automatically placed at the beginning of the row, or creates its own row. For example, the Table, Static Table, Rich Text, and Line controls take up a complete

row. For a logic row of controls, you can specify where to place that control by dropping it either before or after another control.

- Controls cannot be placed on top of each other with the exception of Table, Static Table, and Image controls.

HTML Form Properties

The HTML Form properties listed in the Form Properties sheet are described below:

Details

- Header File Name - The Header property lets you define an external file that can be added to the Head section of the generated HTML in runtime. The end-user then has the option to include the external Head file to the Head section of the HTML form generated in eDeveloper. You can zoom from the Header property to the Open dialog to select the desired Head file.

The name of the Head file will be the result of the expression evaluation. The conversion process adds the contents of the external file to the Head section of the generated HTML form after the title tag. If the external file is inaccessible for any reason, the conversion process will result in no expression evaluation.

- Vertical Factor - Defines the vertical placement of a control on a form grid.
- Horizontal Factor - Defines the horizontal placement of a control on a form grid.
- Show Grid - Specifies whether or not to show the grid lines on the form.
- Form Name - The Name in the Form repository. This name appears in the browser's title bar when the form is displayed. The corresponding HTML code is <TITLE>Form Name</TITLE>.

Input

- Hyperlink - You can zoom from the Hyperlink Value property to the Hyperlink dialog to specify what eDeveloper Program, URL or Control

name should be accessed when the form is submitted to the Web server. The corresponding HTML code is `<FORM action="Hyperlink"...>` For more information on defining hyperlinks, see the section on Hyperlink Settings.

- Input Form - You can define an HTML document as a form, where you can link a push button control to a specific action for submitting information to a web server.
- Context Variables - See the section on Context Variables on page 927.
- Average Palette - For forms with many graphic images with different color palettes, you can set Use Best Palette to Yes to enable eDeveloper to create a combined color palette that provides the best color for each image (only for images of 256 colors).

The default setting for this property is No. It is not recommended to change the default value unless necessary.

Appearance

- Wallpaper - Zoom from the Wallpaper Value property to select the name of the file to be displayed as the wallpaper image of the current form. The corresponding HTML code is:
`<BODY BACKGROUND="Wallpaper">`
- Color - Zoom from the Color Value property to select the background and default text colors of the form from the Color repository. The corresponding HTML code is `<BODY bgcolor="#xxxxxx" text="#xxxxxx">`
- HTML Internal Attribute - You can select an attribute from the HTML Style repository, which will be added to the control's tag. Alternatively, you can define an expression to specify the HTML style by zooming from the Exp property to the Expression Rule repository.

Allows for the standardization of elements on the form by defining the attribute in the HTML Style repository and referencing it in the HTML Internal Attribute property of each control.

Zoom to choose a style from the HTML Style repository. For more information, see the HTML Style Repository section.

Navigation

- Width - specifies the width of the window that displays the form during development.
- Height - specifies the height of the window that displays the form during development.

HTML Style Repository

To open the HTML Style repository, click *Settings/HTML Styles*.

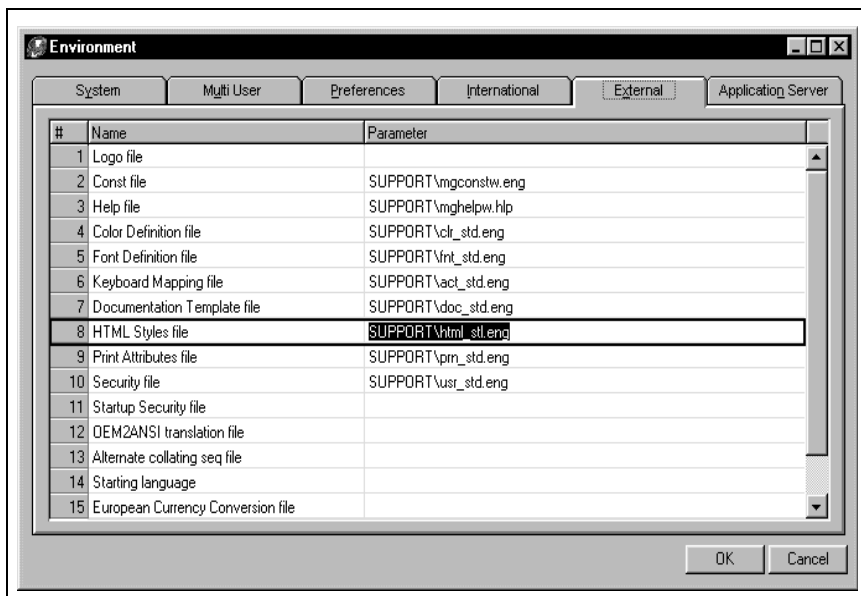


Figure 10-1 HTML Styles File Settings

The HTML Style repository contains the following fields:

- Name - A name representing an HTML style.
- HTML Tag - HTML tags containing the HTML style to be merged into an HTML tag generated by eDeveloper.
Example: BORDER=0 to eliminate a border when displaying Hyperlink objects.

Hyperlink Settings

You can define the URLs that are used when submitting forms to the Web server or when clicking on hyperlinks to call an eDeveloper program.

You can access the Hyperlink dialog by zooming from the Hyperlink property from the Input section in the HTML Form Properties sheet or from the Frame Set Control Properties sheet.

An eDeveloper hyperlink can be designated for one of the following types:

- Magic program
- URL
- Control Name

eDeveloper Program

Select Magic program to create a hyperlink to another eDeveloper program in any application.

When converting the form to HTML, eDeveloper constructs the URL from the following parts:

- The HTTP Requester setting, as defined in the Enterprise Server tab of the Environment repository.
- The eDeveloper system and Public name supplied by the user.
- Any parameters defined in the Arguments setting.

When an eDeveloper Program is selected as the Hyperlink type, you can define the following hyperlink settings:

- eDeveloper system - Defines the name of the eDeveloper system (application) in which the called program is located. You can zoom from the left-hand field to choose an eDeveloper system from the Application repository. You can also zoom from the right-hand field to select an expression.
- Public name - Defines the public name of the program that is to be called. This is the same public name defined in the Program repository. You can

zoom from the left-hand field to choose a program from the Program list for the current application. You can also zoom from the right-hand field to select an expression.

- Arguments - Defines the parameters that are to be passed to the eDeveloper program. You can zoom from the Arguments field to select the arguments from the Argument repository.

To specify a call to an eDeveloper program that will use the input fields of the HTML Merge form, zoom on the Arguments setting of the Hyperlink Settings dialog to open the Argument repository.

- Var: The eDeveloper variable name in the Variable list
 - Exp: Used to pass a constant value
 - Description: Provide a variable description
- Destination frame - Defines the browser frame into which the application will return the called program's HTML output. You can zoom from the right-hand field to select an expression. This field is not available when the Hyperlink Settings dialog box is accessed from the Frame Set Control Properties sheet.

URL

Select URL to create a hyperlink to any legal URL. When converting the form to HTML, eDeveloper copies the URL as it is.

When URL is selected as the Hyperlink type, you can define the following hyperlink settings:

- URL - Defines the complete URL string to be used as the hyperlink. The URL string can be up to 1024 characters.
- Expression - Defines an expression which should evaluate to a complete URL string at runtime.
- Destination frame - Defines the browser frame into which the application will return the called program HTML output. You can zoom from the right-hand field to select an expression. This field is not available when the Hyperlink Settings dialog box is accessed from the Frame Set Control Properties sheet.

Control Name

You can select this option to create a hyperlink to a specified control.

Context Variables

Context variables maintain the context throughout an application when moving from one HTML form to another.

Use the eDeveloper GetParam function to check for the value of a context variable set in a previously called program.

Context Variables are implemented using either Cookies or Hidden Fields.

Cookies

Context Variables can be implemented as HTTP objects called cookies. The data in a cookie is stored on the client (browser) machine in a special file. After a cookie has been stored, the browser automatically sends the data stored in the cookie when the same URL or site is accessed again.

eDeveloper automatically sends the data as cookies in the HTTP process. From then on, whenever the same browser accesses a URL that falls within the range defined by the cookies, the browser automatically sends the data in the cookies as part of the HTTP process. In this way the data becomes accessible to any subsequent eDeveloper program.

When using cookies, the context variables are available to every program called afterwards by the Internet requester.

Remember that not all browsers support cookies, and those that do may offer the user the option to disable this feature. Therefore using cookies for your application's context variables may not always be effective.

Hidden Fields

Context Variables can be implemented as hidden fields that are added to the form.

```
<INPUT "type=hidden" NAME="MGCONXTVAR" value ...>
```

"MGCONTEXTVAR" is the value of this hidden field.

This implementation is available only for HTML forms.

When using hidden fields, the context variables are only available to the program called by the Internet requester directly from the form where the context variables were defined. If context must be maintained in subsequent programs, each program's form must include the appropriate context variables.

Context Variable Settings

You can access the Context Variables dialog by zooming from the Context Variables Value property under the Input section of the HTML Form Properties sheet.

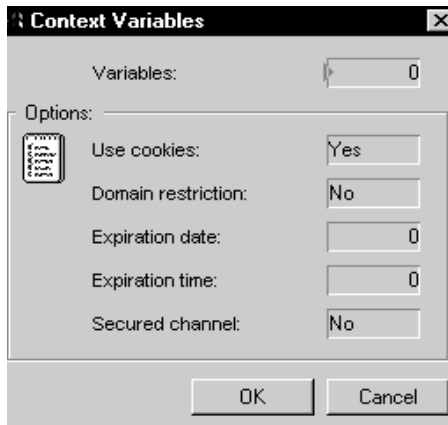


Figure 10-2 The Context Variables Dialog

The fields of the Context Variable dialog are described below:

- Variables
 - Defines the list of context variables to be set by his form.
 - Zoom on the field to define the parameters in the Parameters dialog
- Use Cookies
 - Specifies whether the context variables are implemented as cookies

or hidden variables.










- Default setting is Yes and can only be changed when the HTML Form type is selected.
- Domain Restriction
 - Applies only to context variables implemented as cookies.
 - Specifies whether the cookie, and hence the context variables, will be sent by the browser to all URLs or just to the URLs that are in the same domain as that of the current form.
- Expiration Date and Expiration Time
 - Applies only to context variables implemented as cookies.
 - Defines the data after which this cookie expires and is not sent back by the browser.
 - If no date is specified, the browser deletes the cookie and the end of the session.
 - Zoom from this field to the Expression Rule repository to select the appropriate expression.
- Secured Channel
 - Applies only to context variables implemented as cookies.
 - Form was first displayed. The corresponding HTML code is:
`<INPUT type=submit ...>` or `<INPUT type=reset ...>`








The HTML Command Palette






You can edit controls in the HTML form using pulldown menus, context menus, or the HTML Command palette. Command buttons are enabled or disabled according to their context, as shown below.

The HTML Form Editor Command palette has an HTML Tab containing commands specific for HTML control formatting and form editing. The General Tab and Colors Tab commands are also accessible in the HTML Form Editor.

A description of the buttons that activate functions in the HTML Command palette follows.

Command	Description	HTML Code
	Sets the bold attribute of the selected control(s) or selected	RTF text. ...
	Sets the italic attribute of the selected control(s) or selected RTF text.	<I>...</I>
	Sets the pre-formatted attribute of selected control(s).	<PRE>...</PRE>
	Hide wallpaper during editing.	
	Display a list of fonts from the font repository which may be applied to the selected control(s).	...
	Default browser font	
	H1 - Sets the heading style attributes of the row of the selected control(s) to level 1.	<H1>...</H1>
	H2 - Sets the heading style attributes of the row of the selected control(s) to level 2.	<H2>...</H2>
	H3 - Sets the heading style attributes of the row of the selected control(s) to level 3.	<H3>...</H3>

Command	Description	HTML Code
	H4 - Sets the heading style attributes of the row of the selected control(s) to level 4	<H4>...</H4>
	H5 - Sets the heading style attributes of the row of the selected control(s) to level 5.	<H5>...</H5>
	H6 - Sets the heading style attributes of the row of the selected control(s) to level 6 (smallest or least important).	<H6>...</H6>
	Align Paragraph Left - Align rows of all controls selected to the left.	<DIV align=left>...</DIV>
	Align Paragraph Center - Align rows of all controls selected to the center.	<CENTER>...</CENTER>
	Align Paragraph Right - Align rows of all controls selected to the right.	<DIV align=right>...</DIV>
	Indent - Indent the row of the selected control. Multiple indents are allowed. Active only when the Indent Characters setting, in the Environment dialog, is set to a value other than 0. The Indent command is active for all HTML Controls except for Hot Spots.	...











Command	Description	HTML Code
	Remove Indent - Remove a single indent from the row of the selected control. Active only when the Indent Characters setting, in the Environment dialog, is set to a value other than 0. The Remove Indent command is active for all HTML Controls except for Hot Spots.	
	Numbered List - Sets the numbered list attribute of the row(s) of the selected controls. Active for the Edit control only.	<code></code>
	Bulleted List - Sets the bulleted list attribute of the row(s) of the selected controls. Active for the Edit control only.	<code></code>
	Save HTML - Save the form as HTML into a file. Opens the Save As dialog when first selected.	
	View in Browser - Loads the form into the Web browser for viewing.	














HTML Static Table Command Palette


The HTML Control Palette does not let you independently format the title and edit components of a variable control. The Static Table control, however, lets

you precisely place and independently format all HTML controls. Static Table cells and rows can be easily modified for custom designing.

The Static Table Command palette contains commands specific for the HTML formatting of a static table. The Static Table Command Palette buttons are described below.

Command	Description
	Cell Properties - Accesses the Control Properties for a Static Table cell.
	Row Properties - Accesses the Control Properties for a Static Table row.
	Table Properties - Accesses the Control Properties for a Static table.
	Increase Padding - Increases the cell padding of all cells in a selected table. Access the Padding Properties parameter to determine the padding width.
	Increase Border Width - Increases the static table border width.
	Increase Cell Span - doubles the width of the selected cell.
	Decrease Cell Span - Decreases the selected cell by half its width. This command is accessible only if the selected cell size has been increased.
	Select Cell - Selects the cell of a static table.
	Select Row - Selects the row of a static table.
	Select Table - selects the entire static table.



Command	Description
	Decrease Spacing - Decreases the cell padding of the table. This is accessible only if the cell padding has been increased by using the Increase Spacing command.
	Decrease Border Width - Decreases the static table border width.
	Increase Row Span - Doubles the vertical size of the selected cell.
	Decrease Row Span - Decreases the vertical size of the selected row. You can access this command only if the selected cell had been vertically increased.
	Add Cell - Adds a static table cell to the current row.
	Add Row - Adds a static table row.
	Add Column - Adds a static table column.
	Increase Padding - Increases the space between the cell border and the control placed in that cell.
	No Border - Removes the static table border.
	Fixed Width Cell - Places specific dimensions on a single cell.
	Delete Cell - Deletes a single static table cell.
	Delete Row - Deletes a static table row.
	Delete Table - Deletes an entire static table.






Command	Description
	Decrease Padding - Decreases the space between the cell border and the control placed in that cell.

HTML Control Palette

The Control palette in the HTML Form Editor displays the various types of controls that can be inserted into an HTML form.

A description of the buttons that are unique only to the HTML Control palette follows below. For all other control buttons, refer to The GUI Control Palette section.

Control	Description
	Inserts a Square Hot Spot control used to define a rectangular area on an image that is linked to a specific program or URL. The Square Hot Spot control can only be placed on top of an Image control. Corresponding HTML code: <code><MAP name=...><AREA shape=rect...>...</MAP></code>
	Inserts a Circle Hot Spot control used to define a circular or elliptic area on an image that is linked to a specific program or URL. The Circle Hot Spot control can only be placed on top of an Image control. Corresponding HTML code: <code><MAP name=...><AREA shape=circle...>...</MAP></code>

Control	Description
	<p>Inserts a Java control, used to display a Java applet. The applet can be specified directly or based on a variable or expression evaluated at runtime. The source file name for the Java applet appears on the Java control.</p> <p>Corresponding HTML code: <code><APPLET code="..."><PARAM name="..." value="...">...</APPLET></code></p>
	<p>Inserts an ActiveX control used to display an ActiveX object selected from the ActiveX Class dialog. This control can be specified directly or based on a variable or expression evaluated at runtime. The source file name for the ActiveX object appears on the ActiveX control.</p> <p>Corresponding HTML code: <code><OBJECT classid="..." code="..."> <PARAM name="..." value="...">... </APPLET></code></p>
	<p>Inserts an HTML control used to include another HTML formatted document at the control's position. The document can be specified directly or based on a variable or expression evaluated at runtime. The source file name for the HTML object appears on the HTML control.</p>
	<p>Inserts an HTML static table control.</p>
	<p>Inserts a hyperlink to a sound file in the HTML form. The source file name for the sound object appears on the Sound control.</p>

The Variables palette in the HTML Form Editor displays all variables except for OLE variables.

Fonts and the HTML Controls

You can use any font in the Font repository for the control's Font property. However, remember that although the font you select is displayed in the Form Editor, the actual font displayed in the browser is subject to the conversion rules below:

Font Conversion for HTML Output

- The actual font used in the browser is specified by the browser configuration. This does not apply to font attributes such as size, bold, and italic.
- When using fonts 50-77, the font attributes are converted to the <Hn>, , and <I> HTML formatting tags. If, for example, a text control is defined with font #72 (Header 1 Bold Italic), the control will be converted to HTML as <H1><I> text</I></H1>.
- When using font #7, the control is assumed to be a pre-formatted HTML control and is converted using the <PRE>...</PRE> tag.
- When using other fonts, the font attributes are converted to the , , and <I> HTML formatting tags, where the size attribute is relative to the size of the font used.

Font Formatting Commands

Selecting formatting commands such as Bold, Italic, Font, Default, Headings, and Pre-formatted from the Command palette changes the font property of the selected control. If, for example, a text control has font #50 (HTML Default) in the font property, and the Bold and Heading 3 commands are selected, the font property changes to 60 (Header 3 Bold).

HTML Control Properties

HTML Control properties can be accessed by zooming from the appropriate controls on your form. Different properties can be defined for different types of controls. The HTML control properties are described below:

Model

- Model - You can re-inherit any broken properties or disinherit all properties for the form model. You can also select a different model.

Details

- ActiveX File Name - Applies to ActiveX controls. Defines the name of the ActiveX .ocx file to be used. You can zoom on the property to choose an ActiveX file. The corresponding HTML code is :

<OBJECT code=...>

- Alternate Text - Applies to Image, Sound, and Java controls. Defines the text that appears instead of the control if the browser cannot display the control contents. The corresponding HTML code is: <... alt=...>
- Control Name - Used to define a destination for a local hyperlink. When the control name is used as part of a hyperlink that calls this form, the browser scrolls the page and displays it starting with this control. Corresponding HTML code: ...

The Control Name is not available when the Hyperlink Settings dialog box is accessed from the Frame Set Control Property sheet.

- Default Image File - Applies to Image and Sound controls, and specifies the image file name to be displayed. Any value is overridden when the data property is used. You can zoom on the property to select an image file from the open file dialog. The corresponding HTML code is:
- Enable RTF - Applies to static (text) controls. Converts text and static controls to rich text format.
- Format - Applies to the Push Button control. specifies the label that appears on the button, or formats the label text of an existing push button label.
- HTML File Name - Applies to an HTML control. Defines the name of the HTML file to be included in the control's position. You can zoom on the property to choose an HTML file.

- Java File Name - Applies Java controls. Specifies the name of the Java Class file to be used. You can zoom on the property to choose a Java Class file. The corresponding HTML code is: `<APPLET code=...>`
- Label - Applies to choice controls (radio buttons, combo boxes, and list boxes), and specifies the label that appears for each option in the control.
- Parameters - Applies to Java and ActiveX controls. Used to define parameters based on variables and expressions that will be passed as arguments to the Java Applet/ActiveX Control. You can zoom on the parameter to select the parameters from the Parameter repository. Note that the Parameter repository has an extra column, Name, which is required for the name attribute of the `<PARAM>` tag. The corresponding HTML code is: `<PARAM name=...value=...>`

Input

- Button Type - Applies to Push buttons. specifies the action taken by the form when this button is pushed. The valid values are:
 - Submit - To submit the form.
 - Clear - To clear (reset) all parameters to the values displayed when the form was first displayed. The corresponding HTML code is:

`<INPUT type=submit ...>` or `<INPUT type=reset ...>`

- Hidden Variable - Applies to the Edit control. Specifies whether this control is implemented as a hidden parameter that is not displayed to the end-user, but is submitted. This lets you submit data without the end-user's intervention. The corresponding HTML code is:
`<INPUT type=hidden ...>`
- Hyperlink - Applies to the following controls: Edit (non-modifiable), Text, Image, Square Hot Spot, and Circle Hot Spot. specifies the URL called when the control is clicked. A text control that has a hyperlink defined is underlined. The corresponding HTML code is :
`` or `<AREA href=...>` (for Hot Spots)

See also the section on Hyperlinks.

- **Modifiable** - Applies to Edit controls. Specifies whether upon conversion to HTML, this control will be implemented as an HTML edit parameter `<INPUT type=text...>` whose value is submitted with the form, or whether the control will be implemented as plain HTML text with optional formatting.
- **Multi-line Edit** - Applies to Edit controls. Specifies whether this control is implemented as an HTML multi-line edit parameter. The corresponding HTML code: `<TextAREA...>...</TextAREA>`
- **Password Edit** - Applies to Edit controls. Specifies whether this control is implemented as an HTML password parameter where entered data is not displayed to the end-user. The corresponding HTML code is :
`<INPUT type=password...>`

Appearance

- **Align follow text** - Applies to Image, Java, and ActiveX controls. Specifies the alignment of text immediately following the control. The valid values are: Top, Center, Bottom, Left, Right. The corresponding HTML code is: `<... align=...>`
- **Border Width** - Applies to the Image control. Defines the border width of the Image control. You can submit any numeric value to define the border dimension. Alternatively, you can zoom from the Border Expressions (Exp) parameter to the Expression Rule repository to create an expression that will define a border value. The default value is 0.

The border value will be used for the Border tag of the image in the generated HTML. If the Image control is copied and pasted to another form, its border properties will be retained. If, however, the Image control is included as part of a template, the template format will override its border properties.

The Border Property values will be defined by the IMG_BORDER_VAL and IMG_BORDER_EXP keywords for template documentation.

- **Color** - Defines the color of text that appears in a control. You can also zoom from the Exp field to create an expression that will specify the color selection of the control. If you do not enter a value in this field, the control takes the default color.

- **Font** - Defines the font for text that appears in a control. You can also zoom from the Exp field to create an expression that will specify the font format. If you do not enter a value in this field, the control takes the font defined as HTML Default in the Font repository.
- Remember that the font used to display the form in the browser may be different than the font displayed in the Form Editor. The corresponding HTML code can be any combination of:
`<Hn>`, ``, `<I>`, ``, etc.
- **Horizontal Spacing** - Applies to Image, Java, and ActiveX controls. Defines the horizontal spacing to the left and right of the control. The value is specified in pixels. The corresponding HTML code is:
`<... hspace=...>`
- **Fix Size Table** - Determines whether the table size depends on the size set in the form (=Yes) or depends on the number of records printed on the page (=No).
- **HTML External Attribute** - Selects an attribute from the HTML Style repository that is added around the control's tag. The Exp field lets you define an expression that will specify the HTML style to be added to the control's tag.

Allows for the standardization of elements on the form by defining the attribute in the HTML Style repository and referencing it in every control's HTML External Attribute property.

You can zoom to choose a style from the HTML Style repository. For more information, see the section on the HTML Style repository.

- **HTML Internal Attribute** - Selects an attribute from the HTML Style repository, which will be added inside the control's tag; or an expression, by zooming from the Exp field to the Expression Rule repository that specifies the HTML style.

Allows for the standardization of elements on the form by defining the attribute in the HTML Style repository and referencing it in every control's HTML internal attribute property.

You can zoom to choose a style from the HTML Style repository. For more information, see the HTML Style Repository section.

- **Indent Level** - Defines the indentation level of the control's row in the form. All controls in the selected control's row are affected and have the same indentation. The corresponding HTML code: `...`
- **Paragraph Alignment** - Defines the alignment of the control's row in the form. All controls in the selected control's row are affected and have the same alignment. The valid values are: Left, Center, Right.
- **Text Type** - Applies to a Text control. specifies whether the selected control's row has bullets or numbering preceding it. All controls in the selected control's row are affected and have the same indentation. The valid values are: None, Bullet, Numbered. The corresponding HTML code: numbered bullet `............`
- **Vertical Spacing** - Applies to Image, Java, and ActiveX controls. Defines the vertical spacing above and below the control. The value is specified in pixels. The corresponding HTML code is: `<... vspace=...>`
- **Visible** - An expression that specifies the visibility of the selected control.

Navigation

- **Height** - Applies to Rich Edit Table, List Box, HTML Control, Table, Image, ActiveX, Java, and Sound controls. Defines the vertical size of the control. The corresponding HTML code is: `<HR height=nnn>`
- **Width** - Applies to Edit, Table, Line, Java, ActiveX, HTML, Image, and Sound controls. Defines the horizontal size of the control by a set value or an expression. Note that for Line controls, The Width property is active only when %Width = 0.
- **% Width** - Applies to Line controls. Defines the horizontal size of the line in relative terms as a percentage of the height of the window in which the form is displayed. If the value is 0, the Width property of the control is enabled and defines the absolute width. The corresponding HTML code is: `<HR width=nnn%>`

ActiveX

- **ActiveX Class** - Applies to ActiveX controls. Defines the Class of the ActiveX control. You can zoom on the property to choose an ActiveX class name

from the ActiveX Class dialog, as shown in Figure 10-3. The corresponding HTML code is: `<OBJECT classid=CLSID:...>`

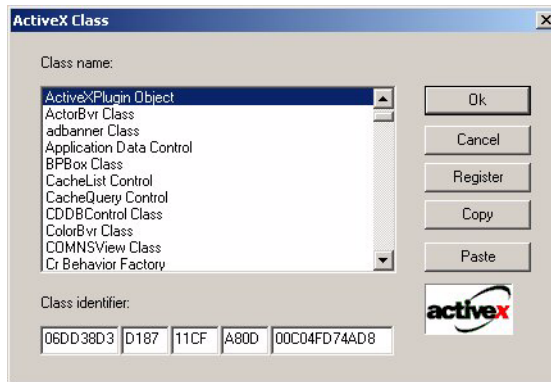


Figure 10-3 ActiveX Class Names

Static Table Control Properties

The Static Table Control Properties can be accessed by zooming from a cell, row, or table. Different properties can be defined for the different parts of the table: cell, row, or the entire static table. All Static Control Properties are described below.

- **Border Width** - Defines the width of a static table.
- **Cell Spacing** - Defines the space between each cell in a static table.
- **Cell Padding** - Defines the space that is between the cell's borders and the control placed in that cell.
- **Color** - Zoom from the Color Value property to select the background and text colors from the Color repository. You can also zoom from the Exp field to the Expressions Rules repository to define an expression that will specify the color value.
- **Column Spanning** - Select the 2 value to increase the size of the selected cell.

- Columns - Select a value from the Columns property to specify the number of columns. The Columns value must be from 1 to 100.
- Fix Width - Specifies if there is a specific default width of a static table cell.
- Horizontal Alignment - Select a value from the Horizontal Alignment property to specify the placement of text in a row cell: Left, Center, or Right. Cell default is Left.
- HTML External Attribute - Selects an attribute from the HTML Style repository that is added to the control tag. You can also zoom from the Exp field to the Expression Rule repository to specify an expression for an HTML Style attribute. For more information, see the HTML External Attribute section.
- HTML Internal Attribute - Selects an attribute from the HTML Style repository which is added to the control tag. You can also zoom from the Exp field to the Expression Rule repository to specify an expression for an HTML Style attribute. For more information, see the HTML Internal Attribute section.
- Paragraph Alignment - Select a value from the Paragraph Alignment property to specify the horizontal alignment of the static table: Left, Center, or Right.
- Row Spanning - Select the 2 value to increase the vertical size of a cell.
- Rows - Select a value from the Rows property to specify the number of rows of a static table control. The Rows value must be from 1 to 100.
- Vertical Alignment - Select a value from the Vertical Alignment property to specify the placement of text in a row cell: Top, Center, or Bottom. Cell default is Center.
- Visible - Specifies whether or not the control will be visible to the user. Zoom from the Exp field to define an expression that will specify when the control will be visible to the user. For more information, refer to Visible property in the GUI Control Properties sheet.
- Wallpaper - Zoom from the Wallpaper Value property to select the name of the file that is displayed as the wallpaper image of the current form.
- Width - Specifies the default width measurements of a static table cell. Note: the Width property is active only when Fix Width is set to Yes.

Frame Set Forms

Frame sets are used to create a complex Internet document where the browser window is divided into multiple frames. Each frame displays a separate HTML page that originates from a different source.

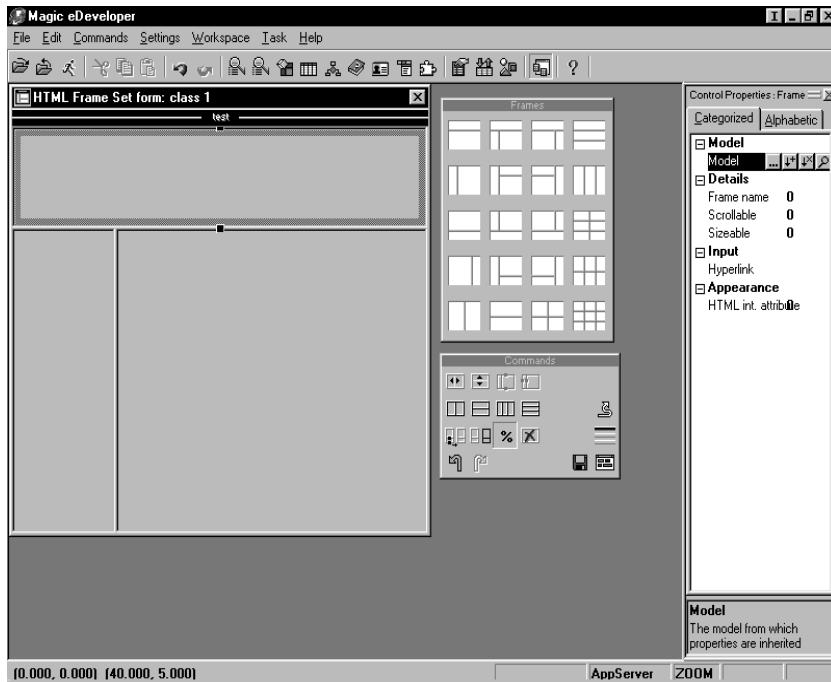


Figure 10-4 HTML Frame Set Forms

The frame set defines a URL and a name for each frame. Each frame content is initially read from the URL defined in the frame set. The frame name allows the result of a hyperlink or submit action sent from one frame to be displayed in another frame. In this way, applications can have parts of their displays, such as a site navigation menu or a company logo, permanently displayed without having to include and reload these parts with every page of the application.

HTML frames sets are hierarchical in nature. Each frame set can contain another frame set, either defined as part of the same frame set or as the page that the frame's URL points to.

Frame Sets must be defined as Class > 0 in the Form repository. You can zoom from the name of a Frame Set form entry in the Form repository to access the Frame Set Form Editor.

Frame Set Form Properties

The Form Set Form properties, which appear in the Form Properties sheet, are listed below.

Model

- Model - The model from which this form can inherit pre-defined properties.

Details

- Form name - The name in the Form repository. This name appears in the browser's title bar when the form is displayed. The corresponding HTML code is: `<TITLE>Form Name</TITLE>`
- Frame Set spacing - Specifies the width in pixels of all of the frame borders in the frame. This property is disabled when With border is set to number. The default setting is 1. The corresponding HTML code is: `<FRAME SET framespacing=n...>`
- Relative Size - If the relative size is set to Yes, the size of all of the frames in the frame set is defined as a percentage of the browser window or the container frame size. When the browser window is resized by the user, all frames are resized relatively.

If the relative size is set to No, the size of all of the frames is defined in pixels. In this situation, resizing the browser window has no effect on the size of the frames. The default setting is Yes.

When the Relative size=Yes, the corresponding HTML code is:

`<FRAME SET ... rows/cols="nn%,nn%,...,*" >`

or when the Relative size=No, the corresponding HTML code is:

`<FRAME SET ... rows/cols="nn,nn,...,*" >`

- With Border - Specifies whether frame borders in this frame set are to be displayed. If No is selected, all frames in the frame set are displayed

without borders in the browser. The default setting is Yes. The corresponding HTML code is: `<FRAME SET frameborder=0/1...>`

Input

- Context variables - Zoom from the Context Variables Value property to specify the context variables for the form in the Context Variables dialog.

Appearance

- HTML Internal Attribute - Selects either an attribute from the HTML Style repository, which is added to the control's tag, or an expression, by zooming from the Exp field to the Expression Rule repository, that specifies the HTML style.

Allows for the standardization of elements on the form by defining the attribute in the HTML Style repository and referenced in the HTML Internal Attribute property of each control.

Zoom to choose a style from the HTML Style repository.

Navigation












- Width - Lets you specify the width of the form.
- Height - Lets you specify the height of the form.








The Frame Set Command Palette

You can edit frames in the Frame Set form using either the Commands pulldown menu, the context menu, or the Frame Set Command palette.

The Frame Set Command palette contains commands for editing a frame or an entire frame set.

The following is a description of the Frame Set Command palette buttons.

Control	Description
	Split vertical - Click this button to change the cursor to a vertical split cursor. Click with the vertical split cursor on any frame to split the frame vertically into two frames.
	Split horizontal - Click this button to change the cursor to a horizontal split cursor. Click with the horizontal split cursor on any frame to split the frame horizontally into two frames.
	Select frame set - Selects the selected frame's container frame.
	Indicate frame set - Displays, but does not select, the selected frame's container frame.
	Equal vertical divide - Splits the selected frame vertically into two identical frames.
	Equal horizontal divide - Splits the selected frame horizontally into two identical frames.
	3 thirds vertical divide - Splits the selected frame vertically into three identical frames.
	3 thirds horizontal divide - Splits the selected frame horizontally into three identical frames.
	Hyperlink - Establishes a hyperlink between the selected frame and the eDeveloper Program or URL.
	Unmark - Deselects the selected frame.
	No border - Toggles between displaying and hiding the frame borders on the browser. The default option displays the borders.

Control	Description
	Relative size - Sets Relative size form property to Yes.
	Delete - Click this button to delete the selected frame or frame set design displayed on the form.
	No Dividers - Removes the dividers between frame set designs.
	Undo - Undoes the most recent command.
	Redo - Redoes the most recent command undone by the Undo command.
	Save HTML - Saves the HTML Frame Set design file.
	View in browser - Invokes the browser to view your frame set design.

All other commands are the same as those in the HTML Command palette, or as those in the GUI Command palette, described in Chapter 9, Display Forms.

Frame Set Control Properties

Each frame in the frame set has the following control properties:

Model

- Model - The model from which the form can inherit pre-defined properties.

Details

- Frame Name - Defines the name of the frame. The frame name is used to reference the frame as the requested target when a hyperlink or submit button is clicked in another frame. The corresponding HTML code is:

```
<FRAME name=" . . . " . . . >
```

- Scrollable - specifies whether the frame has horizontal or vertical scrollbars when the HTML contents displayed are larger than the frame. The default setting is number. The corresponding HTML code is:
`<FRAME scrolling=no...>`
- Sizable - specifies whether the frame can be resized by clicking and dragging on any of the frame borders. All frames adjacent to the border to be dragged must have this property set to Yes. The default setting is number. The corresponding HTML code is: `<FRAME noresize...>`

Input

- Hyperlink - specifies the source for the frame's contents. Zoom or double-click from the Hyperlink Value property to the Hyperlink dialog to define the hyperlink. The selected hyperlink is displayed inside the frame. The corresponding HTML code is: `<FRAME src="..."...>`

Appearance

- HTML Internal Attribute - Selects an attribute from the HTML Style repository to be added to the `<FRAME>` tag. You may also create an expression by zooming from the Exp field to the Expression Rule repository.

HTML Merge Forms

You can use the HTML Merge form to merge data from an eDeveloper task with a predefined template file to create any dynamic HTML page based on a predefined HTML design. The Merge functionality lets you also create an HTML page enhanced with the Web online functionality. The Upload capability of the Internet requester allows you to enable a file to be uploaded from the HTML page and then have it received by an eDeveloper program.

You can zoom from the HTML Merge Name entry in the Form repository to design an HTML Merge form using the Web Authoring Tool specified in the Web Authoring Tool setting of *Settings/Environment/Server*. If the tool you specified supports OLE drag-and-drop, you can drag-and-drop variables from the task's variable palette onto the HTML page.

HTML Merge Form Properties

The HTML Merge Form properties are described below:

Model

- Model - The model from which the form can inherit pre-defined properties.

Details

- HTML File - Specifies the location and name of the HTML file to be used in the HTML Merge form. Logical Names are allowed. You can zoom on this column to choose a file from the Open File dialog.

This property defines an expression that evaluates to the name of the template to be used. You can zoom on this cell to choose an expression from the Expression Rule repository.

- Token Prefix - Defines the prefix for merge tags in the template.
(default: <!\$)
- Token Suffix - Defines the suffix for merge tags in the template.
(default: >)
- Tags Table - Defines the merge tags in the associated HTML template that are matched with and replaced by the appropriate data elements (a variable or an expression). The merge mechanism searches for tags that begin and end with the specified strings and replaces the values according to the matches found in the Tags table but does not search for tags with the specified token. Also, it does not replace their values according to the tags in the HTML file. It replaces their values according with the matches created in the Tags table. For more information, see the section on the HTML Template File Tags.
- Input Fields - You can zoom from this property to the Input Fields list and define the fields that will be used for event handling on the HTML page using the Web online functionality. The columns in the Input Field list are:
 - Field Name - On the HTML Merge form, the Field Name property specifies the name of the HTML input field that can be used for event handling.

- **Exp** - The Expression field holds an expression number, defined in the Expression Rule repository, that specifies the Field Name on the HTML Merge form.
- **Events** - The Events property shows the number of event handlers defined for the Input Field entry. You can zoom from the Event entry to open the Merge Event Handler table, and define the event handlers to be activated when an event occurs on an input field in an HTML page.
- **Form Name** - The Name in the Form repository. This name appears in the browser's title bar when the form is displayed. The corresponding HTML code is `<TITLE>Form Name</TITLE>`
- **XML Output** - When creating an XML file, there are some character values that may not be used in the XML data, for example: `<>`. If XML Output is set to Yes, any merged character value will be converted to a valid XML value. If an expression is specified, the value will only be calculated the first time the merged form is used and will not be recalculated.

Navigation

- **Left** - specifies the position of the left edge of the form's frame. You can specify the value at runtime by zooming to the Expression Rule repository, and entering an expression that evaluates to the coordinates of the left edge of the form's frame.
- **Top** - specifies the position of the top edge of the form's frame. You can specify the value at runtime by zooming to the Expression Rule repository, and entering an expression that evaluates to the coordinates of the top edge of the form's frame.
- **Width** - specifies the width of the window that displays the form.
- **Height** - specifies the height of the window that displays the form.

Web Online Event Handlers

The built-in event handlers that can be defined in the Merge Event Handlers table, accessed from the Events column in the Input Fields table, are

described below:

Event Handler	Description
OnAbort	An Abort event occurs when the user aborts the loading of an image, for example by clicking a link or clicking the Stop button.
OnBlur	A blur event occurs when a form element loses focus or when a window or frame loses focus. The blur event can result from a blur method or from the user clicking the mouse on another object window, or tabbing with the keyboard.
OnChange	A change event occurs when an HTML input field loses focus and its value has been modified. Use the OnChange event handler to validate data after a user modifies it.
OnClick	A click event occurs when an object on a form is clicked. For input fields, links, and images, the OnClick event handler can return False to cancel the action normally associated with a click event.
OnError	An error event occurs when the loading of a document or image causes an error.
OnFocus	A focus event occurs when a window, frame, or frame set receives focus or when a form element receives input focus. The focus event can result from a focus method or from the user clicking the mouse on an object or window or tabbing with the keyboard. Selecting within a field results in a select event, not a focus event.

Event Handler	Description
OnMouseOut	A OnMouseOut event occurs each time the mouse pointer leaves an area (client-side image map) or link from inside that area or link. If the mouse moves from one area into another in a client-side image map, you will get OnMouseOut for the first area, and then OnMouseOver for the second.
OnMouseOver	An OnMouseOver event occurs once each time the mouse pointer moves over an object or area from outside that object or area. If the mouse moves from one area into another in a client-side image map, you'll get OnMouseOut for the first area, then OnMouseOver for the second.
OnReset	A reset event occurs when a user resets a form by clicking the Reset or Reload button.
OnSelect	A select event occurs when a user selects some of the text within a text or text area field.
OnDbClick	A double click event occurs when a user double clicks an object on the form.
OnHelp	A help event occurs when a user presses F1 .
OnLoad	A load event occurs when the form is loaded.
OnUnload	An unload event occurs when the form is removed.
OnKeyPress	A key press event occurs when a user presses any key.

In addition to the built-in event handlers, you can define developer-defined event handlers in the HTML template file according to the following convention:

On <Event Name>=UserCommand ('Magic Command')

where:

- `Event Name` is the name of the event that activates the handler.
- `UserCommand` is the eDeveloper JavaScript function that handles the event.
- `Magic Command` is the content of the command, according to the eDeveloper Web Online Rules.
- `Cmnd` - The Command property displays the number of commands defined in the Event Handler. From the Cmnd column you can zoom to the Merge Command table and specify the commands to be executed when an event occurs on an input field that activates the event handler.

The Merge Command Table

The columns of the Merge Command Table, accessed from the Cmnd column of the Merge Event Handler table, are described below:

- `Command` - The Command field specifies the commands that are executed when an event occurs on an input field activating the event handler. You can zoom from the Command field to the Merge Commands List.

Note that you can add your own commands to this list by specifying the commands in a section of the Magic.ini file called [MAGIC_WEBONLINE] in the following format:

CommandName = Function Name, Parameter Type

where:

`CommandName` is the command to be displayed in the Merge Command list during development,

`Function Name` is the name of the applet function to be called, and

Parameter Type can specify: N for field name parameter, V for value parameter, or H for Hyperlink.

- Field Name - The Field name specifies a field name to be used as an argument for the command.
- Field Name Expression (Exp) - The Expression field specifies a field name argument for the command dynamically with an expression from the Expression Rule repository.
- Value - Value specifies a value argument of the command. For details regarding the arguments for each command, see the Merge Commands List.
- Hyperlink - A Hyperlink command activates an eDeveloper program or a call to a URL from the event handler. After specifying a Hyperlink command in the Merge Command repository, zoom from the Hyperlink column to the Hyperlink dialog to specify the program or URL to call and relevant parameters.
- Condition - The Condition field displays a string expression that contains a condition for execution of the command by the eDeveloper applet in the browser. The string can contain the names of input fields and values to which they can be compared.

The Merge Command List

The commands available on the Merge Command List, accessed from the Command field of the Merge Commands table, are:

Commands	Arguments	Description
Update Field	Field Name, Value	Updates a specific field with a value. Use for setting the value of Selection Lists, Check Boxes, and Radio Buttons.
Disable Field	Field NameValue	Disables writing on the specified field.
Move Focus	Field Name	Moves the cursor to the named field.
Hyperlink	1. Application Name, Program Name, Arguments 2. URL address (Target)	1. Calls the specified eDeveloper program, updating the received arguments respectively. 2. Calls the specified URL.
Alert	Alert string	Displays a dialog containing the alert string.
Status line	Status string	Displays the specified string in the status line.
Update Image	Field Name, Image Source	Updates a specific image with the new image source.
DisableField	Field Name	Disables writing on the specified field.
Set Color	Field Name, Color	Sets the color of the specified field.
Set ReadOnly	Field Name, Logical	Sets the specified field to read only according to the value of the Logical argument.

Commands	Arguments	Description
Set BG Color	Color	Sets the background color of the document to the specified RGB color.
Set CheckBox	Field Name, Logical	Sets the specified check box to checked or unchecked according to the Logical argument.
Set Selection List	Field Name, String1, String2, ...	Sets the values of the specified selection list.
Set Selection	Field Name, Numeric value (starting at 1)	Sets the current value of the selection list.
Set Radio Selection	Field Name, Numeric value (starting at 1)	Sets the current value of the radio button.

HTML Template File Tags

The HTML Tags table is used to define the merge tags in the associated HTML template that are matched with and replaced by the appropriate data elements (a variable or an expression). You can open the HTML Tags table by double clicking the ellipse button next to the Tag table property in the HTML Merge property sheet.

When the Output Form operation is selected in runtime, the tags and displayed values from the variable/expression and picture are sent to the merge mechanism. If the merge mechanism cannot find the tags of the displayed values in the template, it will disregard the tag and its value. If the tag is not embedded in a repeat area it will replace its value with the tag. If a value is sent more than once to a tag that is not embedded into a repeat area, then the second value is not used.

For tags that are embedded, each Output Form operation will duplicate the contents of the repeat area and replace tags with values.

For tags that are at the head of an IF block (part of the MGIF tag), the value of the tag will be regarded as a logical value, which will specify if the IF block will be copied to the output, or the ELSE block (if one exists). An IF block may be inside a repeat area. This means that there are multiple values of this tag, and for each 'TRUE'LOG the IF block will be duplicated.

When a non-logical value is sent by an Output Form operation into a tag that "belongs" to an MGIF tag, the value will be regarded as 'TRUE'LOG.

All trailing blanks of data strings sent as output to the merge mechanism are trimmed.

The columns in the Tags Table are:

- Tag name - Specifies the name of the data element to be matched with a data tag in the HTML template. If a template name was explicitly specified for the selected HTML Merge file, you can zoom to select a tag name from the Tag Name list of tags found in the HTML template file.
- Var - defines a variable whose value will replace the tag. Only a BLOB variable cannot be selected. Zoom to select a variable from the list of variables available.
- Exp (variable) - defines an expression whose value will replace the tag.
- Name - specifies the name of the selected variable or expression.
- Picture - defines the picture to be used when merging the value of either the variable or the expression with the template. The picture will default to either the picture defined for the variable or the default picture of the expression.
- Exp (picture) - defines the picture as an expression evaluated at runtime.

HTML Merge Tags

The template file may be of any type as long as the eDeveloper merge tags are stored as regular ASCII strings.

HTML Merge tags have the following generic form:

{Token Prefix}{Token_[name]}{Token Suffix}

where Token Prefix and Suffix are defined for each I/O repository entry. By default the Token Prefix and Suffix are defined as < !\$ and > respectively.

The {Token_[name]} part of the tag is one of the following:

- < !\$MG_name> - < !\$MG_name> is a data tag that is matched during runtime with a data element defined in the Tags repository of the HTML Merge Properties dialog under the same name. If a match is found, the value replaces the tag during the Output Form operation.
- < !\$MGRepEAT> - The < !\$MGRepEAT> tag defines the beginning of a repeated area. The repeated area is duplicated and processed for each Output Form operation, thereby allowing for an unknown number of data rows. The tag is removed from the output.
- < !\$MGENDRepEAT> - The < !\$MGENDRepEAT> tag defines the end of a repeated area. The tag is removed from the output.
- < !\$MGIF_name> - The < !\$MGIF_name> tag defines the start of an IF block. The name specified is matched with a data element defined in the Tags repository of the HTML Merge Properties dialog. The data is assumed to be logical and is evaluated. If the data is True, the rest of the IF block is processed. Otherwise the ELSE block is processed. If the data does not evaluate to a logical value, the data is assumed to be True. The tag is removed from the output.
- < !\$MGELSE> - The < !\$MGELSE> tag defines the start of an ELSE block and the end of an IF block, which must precede the ELSE block. The ELSE block is processed if the name data value of the IF block evaluates to False. This tag is optional. The tag itself is removed from the output.
- < !\$MGENDIF> - The < !\$MGENDIF> tag defines the end of an IF block, or an ELSE block if one exists. This tag is mandatory if a < !\$MGIF_name> exists. The tag is removed from the output.
- < !\$MGINCLUDE> - The < !\$MGINCLUDE> tag lets you include an entire HTML file during the Merge process. This beginning tag is followed by the

name of the file to be included. The file name can be a tag itself. The Include process takes place after the file is fully merged.

- `<!$MGENDINCLUDE>` - The `<!$MGENDINCLUDE>` tag defines the end of the Include command. This tag is mandatory if the `MGINCLUDE` tag exists.

Examples:

```
<!$MGINCLUDE> \tmp\t1.html <!$MGENDINCLUDE>
```

includes the file `t1.html` in the current HTML template file.

```
<!$MGINCLUDE> <!$MG_T1> <!$MGENDINCLUDE>
```

includes the file referred to by the `<!$MG_T1>` tag in the current HTML template file.

HTML Merge Syntax Rules

The number and order of the `<!$MGRepEAT>` and `<!$MGENDRepEAT>` tags must match.

The number and order of the `<!$MGIF_name>`, `<!$MGELSE>`, and `<!$MGENDIF>` tags must match.

`<!$MGELSE>` tags may only be placed between a pair of `<!$MGIF_name>` and `<!$MGENDIF>`.

RepEAT and IF-ELSE-ENDIF blocks can be nested.

HTML Merge Runtime Behavior

When the Output Form operation is selected in runtime, the tags and displayed values from the variable or expression and picture are sent to the merge utility. If the merge utility cannot find the tags of the displayed values in the template, it will disregard the tag and its value. If the tag is not embedded in a repeat area, the merge utility will replace the tag's value by the tag. If a value is sent more than once to a tag that is not embedded into a repeat area, then the second value is not used.

For tags that are embedded, each Output Form operation duplicates the contents of the repeat area and replace tags with values.

For tags that are at the head of an IF block, which are part of the MGIF tag, the value of the tag will be regarded as a logical value that specifies if the IF block, or the ELSE block, if one exists, will be copied to the output. An IF block may be inside a repeat area. This means that there are multiple values of this tag, and for each `TRUE'LOG the IF block will be duplicated.

When a non-logical value is sent by an Output Form operation into a tag that belongs to an MGIF tag, the value will be regarded as `TRUE'LOG.

HTML File Merge Example

1. Create a HTML file with the following merge tags and save it in a location easy-to-find:

```
<!$MGREPEAT>
```

```
<!$MG_number><!$MGIF_IsEven>Even<!$MGELSE>Odd  
<!$MGENDIF><!$MGENDREPEAT>
```

2. Create an eDeveloper program with the following characteristics:

- Task Type - Batch
- End Task Condition - Counter(0)=10
- Evaluate Condition - After updating record
- Create a new I/O file with the following characteristics:
 - Name - Merged File. You can enter any name.
 - Media - File
 - Exp/Var - Location and name for the resulting file, for example, c:\out_merge.text
- One form with the following characteristics:
 - Name - HTML template. You can enter any name.

- Class - 1
 - Interface Type - HTML Merge
 - HTML File - Location and name for the HTML file that you created
 - Tags Table - Creates two entries that will match the two HTML tag in our Template file:
 - Tag Number - Number replacee with the expression, Counter(0)
 - Tag Name - IsEven replaced with the expression, Counter(0) MOD 2=0
 - In the Record Suffix level, you can create a new line with the following characteristics:
 - Operation - Output form
 - From - HTML template
 - I/O - Merged File
3. Save the program, check and execute it. If you open the resulting file, you should see the following lines:
- 1 Odd
 - 2 Even
 - 3 Odd
 - 4 Even
 - 5 Odd
 - 6 Even
 - 7 Odd
 - 8 Even
 - 9 Odd
 - 10 Even

Web Online Page

eDeveloper provides the web online feature as a way of creating and handling interactive web pages. An eDeveloper Web Online page is an interactive page in an eDeveloper application that runs in a standard Web browser and has the appearance to the end-user of an online application. For example, an HTML data entry application using eDeveloper Web Online is able to update the input data fields without submitting and retrieving an HTML page for each event or operation.

The eDeveloper Web Online facility is similar to the eDeveloper online data entry process, where eDeveloper logic is being executed per field on the form. This allows fields and other elements to be updated and validated on the fly without any explicit requests. Event handling on a field level avoids the commonly used heavy page mode. The “submit” button so prevalent on HTML pages is not necessary.

Web Online Supports the UNIX Web Server

Web Online supports the UNIX Web Server by using the Web Document Alias environment property as the location for the Web Online files (Jar, JS) that are referenced on the Web Online page. This lets the end-user access a readable directory other than the CGIBIN directory.

These files must be placed in the directory specified in the Web Document Path environment setting. The alias must also be set in the Web server. If no Web Document Path is set, the path for the Web Online files is specified from the HTTP Requester property.

Web Online Response

The Web Online Response is used to specify the commands to be executed in response to an eDeveloper program called by a Web Online page.

Web Online Response Form Properties

Use the Web Online Response Form Properties sheet to specify the HTML Template file name or expression to be used by an Output Form operation as a

Web Online Response. Press CTRL+P on the Web Online Response Name entry in the Form repository to open the Web Online Response Form Properties sheet.

Upload Capability of the Requester

The Internet requester lets you design an HTML page on which you can enable a file to be uploaded and then received by an eDeveloper program.

To enable this you should define your HTML file as follows:

1. The form method attribute type must be POST. For example:
`<form action=/scripts/mgrqispi.dll method=post>`
2. The form tag should include the following attribute:
ENCTYPE="multipart/form-data"
for example:
`<form action=/scripts/mgrqispi.dll method=post
ENCTYPE="multipart/form-data">`
3. You should add to the form an Input tag of type: FILE. This tag creates both an edit box in which the user can enter the local path and file name, and a push button allowing the user to browse to select the file.
This input tag should be identified by a clearly defined name.
for example:
`<input type=file name="MyFile">`
4. The given name of the file input tag should be added to the hidden input of the arguments.
for example:
`<input type=hidden name="ARGUMENTS"
value="var1,var2,...,MyFile,...varn">`
5. A corresponding BLOB virtual variable should be created in the called eDeveloper program. This variable is updated with the uploaded file content.
The variable's content can be converted to a physical file on the server side using the Blb2File function.

Report Forms

Report Forms have Class > 0 and either Text-based or GUI Output interface types.

The Form Editor displays all the forms of the same class in the table.

Therefore, in the case of reports, all the associated header, detail and footer forms are displayed together with a labeled divider.

GUI Table Control Functionality

The GUI printing mechanism allows printing of the Table control. This enables easier and faster creation of reports in multi-line format.

A Table control can be added to GUI Output Forms. The following Table control properties facilitate graphical printing:

- Fix Size Table - Valid values: Yes, No. No is the default.
 - No - The table will have only one record line. This line repeats for each record. The actual size of the table depends on the number of records that are printed on the page.
 - Yes - The table prints the size that is declared in the form.
- Title on Every Page - Valid values: Yes, No. Yes is the default. Specify Yes to print a table header on every printed page. Specify No to print a Table header prints only at the beginning of the table.

When the first record of a table is printed, the table header and all other controls on the form are printed at the same time.

Controls that are not part of the table are printed once when the first record and the header are printed. The location of these controls sets the minimum size of the currently printed table. The table length will be long enough to encompass the bottom-most control on the form.

- When records after the first record are printed, they are printed adjacent to the previously printed record. Other controls in the form are no longer printed. The table is resized accordingly.
- When a form with a table is printed, the height of the page footer is

enlarged by the height of the closing line(s) of the table for calculation of end of page condition. This happens when the record to be printed overflows the original size of the form.

- When an end of page condition occurs while printing a record on a table, the following actions are performed:

Closing line of the table - the header of the table (and all other controls in the form) are repeated on the next page as if it is the first record in the table that is being printed.

The header of the table is output after all other headers (if any) that are repeated in an end of page condition.

When a different form is printed and the previously printed form contained a table, the previous table is closed, and the closing line or lines of the table are printed.

When a different form is printed and the previously printed form contained a table, the new form is printed after the table's form, even if the table's form is not completely filled with records. In this case, the closing line(s) of the table are printed at the area of the previous table form.

Multi-Line Edit Printing

You can print a Class > 0 form with variable length textual data with the dynamic sizing of a Multi-Line Edit control during printing.

For a GUI Output form, set the Expand Form property to One Page or Multi-Page. When printing, the control will resize to contain all of the text in the control's frame. The controls located below the Multi-line Edit control are pushed down to keep the same distance from the Multi-line Edit control. The form increases its size to make room for the Multi-line Edit control's new lines. One Page specifies that the control will expand the form up to a single page. Multi-Page specifies that the form will expand across multiple pages to fit all the lines of the Edit control.

Note: Expanded controls in a table will be printed on multiple pages only if the form does not contain expanded controls outside the table. In that case, Expand Form will behave as if set to One Page.

Only the single page option is available for text-based forms.

Printer Attribute Support

The graphical printing feature supports WYSIWYG (what you see is what you get) printing of GUI Output forms using fonts and colors. Colors are translated to shades of gray on non-color printers.

The media column in the I/O file is defined as Graphic Printer, which directs the output of an I/O file to a Windows printer driver, enabling standard Windows printing.

eDeveloper's printer attribute support lets you easily control the appearance of printed output. Printer attribute support is relevant to text-based I/O only. Your output can include whichever fonts (such as Times, Roman, Helvetica, etc.) and type faces (for example, bold and italic) your printer supports.

If you print to a graphical printer using a font other than True Type font, and the font is not available to the printer, there may be situations where some characters will be altered in the printed output.

Printer Settings

For each printer defined in the Printer repository, eDeveloper must know which printer control codes are employed for each typeface/font combination, and which colors are to represent which print styles. This information is contained in two printer support files, the Print Attributes file and the Printer Commands.

The Print Attributes File

The Print Attributes file contains a listing of logical eDeveloper print styles that may be used to enhance the appearance of a report form. The logical print style is resolved in runtime to actual printer control codes. Because the resolution of the logical style is done in runtime, the application can be written without any prior knowledge of the physical printers in a specific installation. All the necessary information is provided in the eDeveloper environment settings while installing the application.

For each logical print style the Print Attributes file contains its association to actual printer control codes by means of print styles. The print styles are translated to actual printer control codes in the Printer Commands file. Several print styles can be combined to create one logical eDeveloper print style.

The Print Attributes file also contains a color number that is used in the form layout editor to represent the logical print style.

A style description provides the name by which the logical print style will be recognized in the application.

This table can be modified with an eDeveloper editing tool, just like any other eDeveloper table or repository.

The Printer Commands File

The Printer Commands file lists the actual printer control codes (escape sequences) for each print style. Each entry is either of the form:

```
style-name=start-sequence:end-sequence or  
style-name=start-sequence
```

where *style-name* is a user-defined style name that appears in the Printer Command file, *start-sequence* is a printer control (escape sequence) string that is to be sent to the printer at the beginning of a text section that is to appear in the given print style, and *end-sequence* is the escape sequence string that is to be sent to the printer at the end of such a text section. The *start-sequence* and *end-sequence* must be separated by a colon.

The second form is for printer commands that do not require a terminating sequence, such as a restart command.

If you want to modify the Printer Command file, use a text editor.

Two sample Printer Command files are supplied with eDeveloper. These are:

1. LJ3.ATR for an HP Laserjet printer
2. EPSON.ATR for an Epson printer and compatible

Print Styles

The Print Attribute repository is used to construct logical print styles that combine print styles with eDeveloper colors. The colors are later used to paint the form areas that should be output using the print attribute represented by the color.

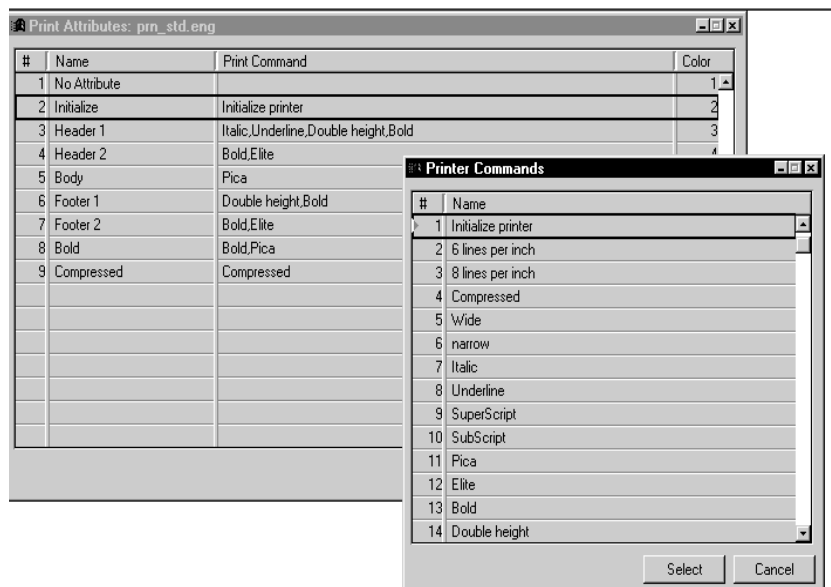
Each entry in this repository corresponds to a particular eDeveloper logical print style. The columns in this repository are:

- # - A sequential line number, assigned by eDeveloper. You cannot edit this column.

- Name - A description of the logical print attribute. This description will be displayed in the Print Attribute list when selecting a logical Print Attribute for a designated form area.
- Print Command - The Print Command is a collection of one or more print styles that make up the eDeveloper logical print attribute. Each print style represents a font or a print typeface of a physical printer. The values for this field are selected by zooming to the Printer Command list. The Printer Command list contains a collection of style-names read from the Printer Command files assigned to each eDeveloper Printer in the Printer repository. Style names that appear in more than one command file appear only once in the Printer Command list.

In runtime, when the logical print attribute is resolved to the actual printer control codes, eDeveloper will send to the printer, the prefix part of all the printer commands that are defined for the print attribute according to their order of appearance in the Print Command field. Then the form control will be sent to the printer. After the form item, eDeveloper will send all the suffix parts of all the Printer Commands that are defined for the print attribute according to their order of appearance in the Print Command field.

By combining several Print Commands into one logical print attribute, it is possible to create custom print styles for use in the application.



MAGIC for Windows

Figure 10-5 The Frame Control Properties Dialog

Color Print Attribute

Specifies the number of the color associated with the logical print attribute. Zoom from this column into the Color repository specified in the Environment dialog to choose the color.

The color is used to assign the print attribute to form areas. Color is used this way because it occupies no form area, and it serves as a visual indicator of the usage of print attributes.

Print Attribute Components

Figure 10-6 below shows the relationships among the various print attribute components. Marking and painting a block on a form layout has the effect of specifying a print attribute for that block. The print attributes are contained in the Print Attribute list. You can open the Print attributes list by double-clicking

the ellipse button related to the Color property for a selected text-based control.

The Print Attribute repository relates each named print attribute to an actual series of print commands, and, for reference, to specific colors.

The print commands identified in the Print Attribute repository are made up of various components contained in the Printer Command list. This list is generated from the total set of printer commands contained in all of the various commands files within the eDeveloper system.

The Printer repository relates eDeveloper Printers or logical print names to Printer Command files and to queue names representing physical printers.

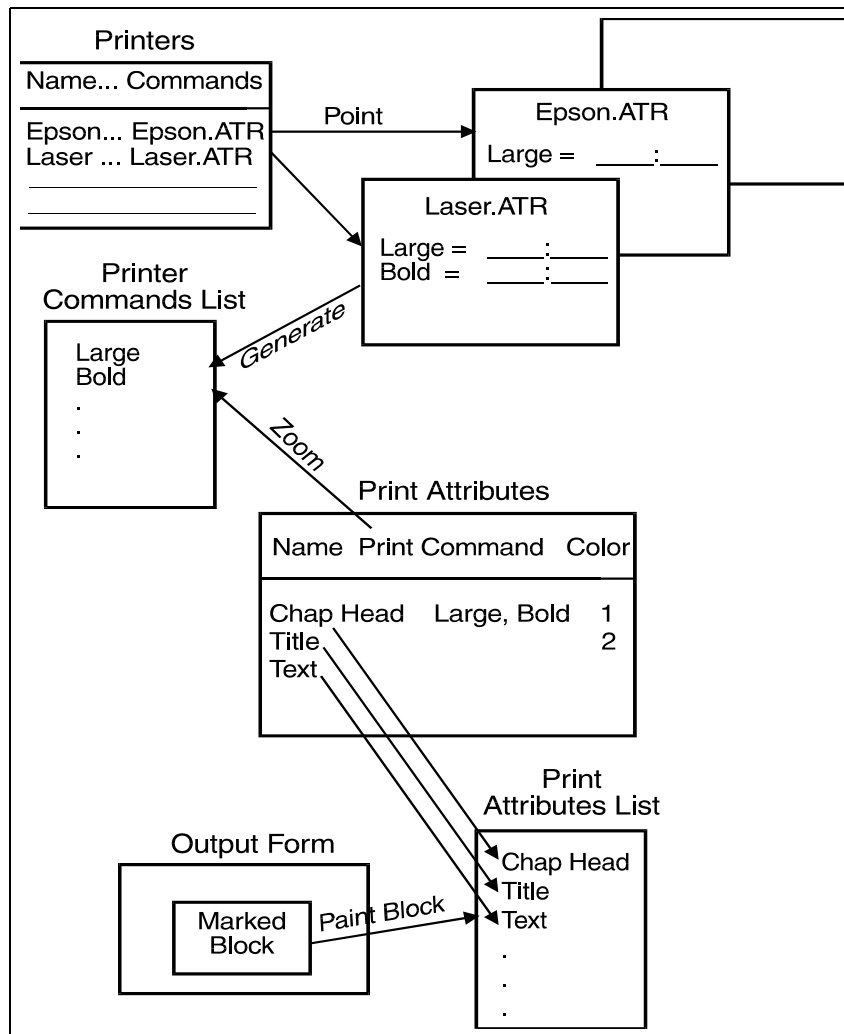


Figure 10-6 Relationships Among Print Components

Runtime Resolution of Print Attributes

At runtime the print attributes specified in the form layout are resolved according to the eDeveloper (logical) printer and the commands file specified,

and to the settings in the I/O File repository and in the Print Attribute repository, as shown in the figure below.

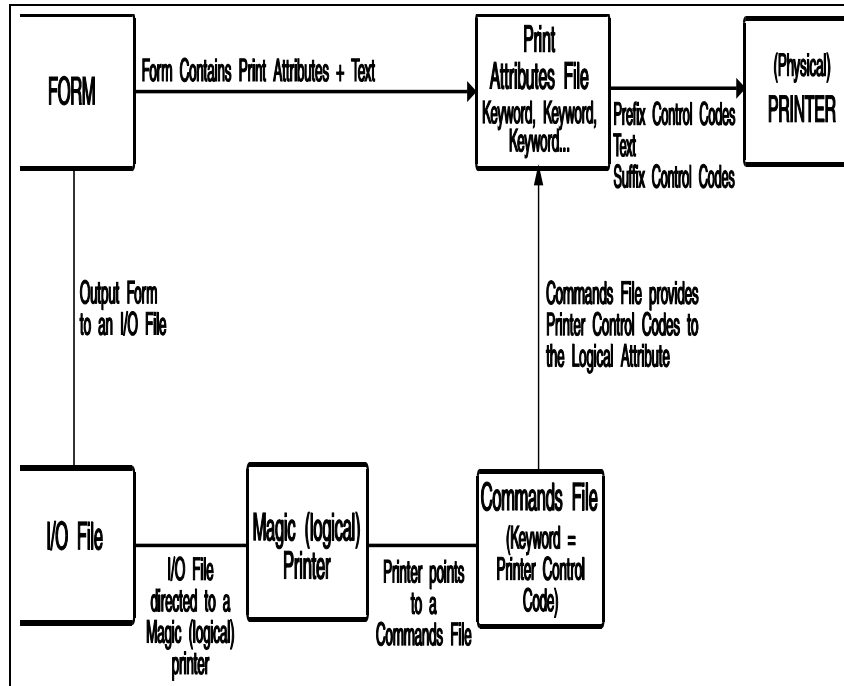


Figure 10-7 The Printing Process

Data management lets you organize the data that is stored in the physical database by determining when and how the stored data is updated.

In this chapter:

• Transaction Processing
• Transactions and Execution Levels
• Deferred Transactions
• Transaction Processing Recovery
• eDeveloper's Internal Transactions
• Deadlocks and Transaction Processing
• Rollback
• eDeveloper Cache
• Error Handling

Transaction Processing

Transaction processing is a data processing technique used to preserve database integrity. It can be defined as the execution of a set of logically related data modifications, which must be *committed* (completed and written to disk) or aborted as a single unit. Reservation and credit-checking operations are typical examples of such transactions.

Transactions can also be used to secure Read operations, as opposed to Read/Write operations. A Read transaction ensures that the data read within the transaction is not modified by other users.

The transaction processing technique automatically logs all of the updates of a transaction to a temporary transaction file. The updates in this file are cleared only when the transaction is complete; that is, the updates to all the regular database tables have been completed successfully. If a problem is detected in any of the tables affected by the transaction, the entire transaction is canceled and the database is *rolled back*; that is, restored to its original state before the transaction occurred. The rollback uses the information stored in the Transaction Log file to restore the application.

The major benefit of transaction processing is the protection from computer system crashes. If the computer crashes in the middle of online data entry or batch processing, the entire transaction is aborted. This prevents incomplete updates to the database, and maintains database consistency.

Transactions and Execution Levels

You can activate transaction processing at any of the task execution levels by setting the following Transaction properties in the Task Properties dialog:

Transaction Mode

- **Deferred** - Data Manipulation (DM) statements are stored in a cache. During the task flow, the DM statements are not sent to the physical database. The statements are only implemented in the database at the expected commit time. All the DM statements accumulated in the cache are implemented at once, and the transaction is closed.

- Nested Deferred - Same as Deferred, except that when such a task is called from another deferred task, it opens a new deferred transaction nested within the hosting one.
- Within Active iTransaction - The task is implemented within the parent transaction. A physical transaction that is a parent will have a physical transaction as a child. A parent that is a deferred transaction will have a deferred transaction as a child.
- Physical - DM statements are implemented after the Record Suffix (same as in previous versions).
- None - eDeveloper does not open a transaction for the task. This option is available for browser tasks only.

Transaction Begin

- Before Task Prefix - The transaction is opened before the Task Prefix and closed after the Task Suffix.
- Group - The transaction is opened before the Group Prefix. The end-user must specify the variable that triggers the Group transaction. This variable must appear in one of the task's group events. You can access the Group transaction only when the Task Type property is set to Batch.
- On Record Lock - The transaction is opened before a record is to be locked and closed after the record update.
- Before Record Prefix - The transaction is opened before the Record Prefix and closed after the record update.
- Before Record Suffix - The transaction is opened before the Record Suffix and closed after the record update.
- Before Record Update - The transaction is opened before the physical updates are sent to the database.
- None - A transaction is not opened for the browser task. When eDeveloper calls a task with the Transaction Begin set to None from a task with an open transaction, the task behaves as if it is assigned to Within Active Transaction.

When changing the task type in the Task Properties dialog (for example, from Online to Batch), all the transaction properties (that is, Transaction Mode, Transaction Begin, Locking Strategy, and Cache Strategy) for that task revert to their original default values specific to that task type.

Physical Transactions at Task Level

Before Task Prefix

When you define the Transaction Begin property as Before Task Prefix, eDeveloper:

- opens the transaction before fetching the task,
- commits the transaction only after the data file updating, which follows the execution of the Task Suffix, and
- includes all operations between the open and the commit, including subtasks, within the transaction scope.

Task Level Transaction Usage Considerations

Defining a transaction at the Task level causes all updates made during the task execution to be logged as a single transaction. Therefore:

- Use the transaction at the Task level in Batch tasks with a dataview that includes only a few records for better concurrence. Use transactions at the Task level in Batch tasks when performance is important (SQL databases). ISAM type databases may require a lot of disk storage for extended transactions in Batch.
- Using a transaction at Task level for Online tasks may cause problems for other users accessing the same tables, in multi-user applications, because the tables may be locked during the transaction for long periods of time.
- Task level transactions that cover large quantities of records will result in the expansion of the Transaction Log file, thus reducing available space on the disk drive. Be careful not to run out of disk space, because this will abort the transaction.

Physical Transactions for the Group Level

Group levels are available only for Batch tasks.

Group Level Transaction Usage Considerations

You can use a Group Level transaction to split the processing of a large table into processing by record groups. This way you reduce the scope of the transaction and maintain the benefits of transaction processing.

Physical Transactions for the Record Level

The Transaction Begin property in the Task Properties dialog has five different values for record level transactions: On Record Lock, Before Record Prefix, Before Record Suffix, Before Record Update, and None.

On Record Lock

When you define the Transaction Begin property as an On Record Lock, eDeveloper opens the transaction before the record is locked.

Before Record Prefix

When you define the Transaction Begin property as Before Record Prefix, eDeveloper:

- opens the transaction before fetching the record,
- commits the transaction only after the data file update that follows the execution of the Record Suffix, and
- includes all operations between the open and the commit, including subtasks, within the transaction scope.

Before Record Prefix Level Transactions Usage Considerations

Because the transaction is open during the interactive stage (Record Main), you should not usually define a transaction at Prefix level for Online tasks in a multi-user environment. The transaction involves locking and may cause problems for other users trying to access the same records or table.

Before Record Suffix

When you define the Transaction Begin property as Before Record Suffix, eDeveloper:

- opens the transaction before executing the Record Suffix, and
- commits the transaction only after the dataview update that follows the execution of the Record Suffix.

Before Record Suffix Level Transaction Usage Considerations

All the Record Suffix operations, including subtask calls, are included in the transaction.

In case of transaction failure, the engine recovers or aborts execution, depending on the Error Behavior Strategy property setting in the Task properties dialog. Alternatively, you can define your own error handler to intercept the error.

Before Record Update

When you define the Transaction Begin property as Before Record Update, eDeveloper:

- opens the transaction after the Record Suffix execution, just before the execution of data file updating, and
- commits the transaction only after the data file update has been executed.

Before Record Update Level Transaction Usage Considerations

Note that a Record Level transaction defined as Update does not include Record Suffix operations. The transaction defined as Update is the one most highly recommended for multi-user applications, because it reduces potential problems of concurrence among end-users.

None

As for all the other levels, None means that no transaction is required at Record level.

Deferred Transactions

In previous versions, eDeveloper implemented Data Manipulation (DM) statements (insert/update/delete) after the Record Suffix. This process, called physical transactions, updated the content of the physical database after each DM statement. Processing the changes of records in a dataview for an application in runtime mode is time-consuming.

Deferred transactions delay the implementation of the Data Manipulation (DM) statements until the time you must commit the changes made to records of a dataview. eDeveloper stores each DM statement in a cache. When you are ready to commit the changes to the database, all changes are implemented concurrently. This keeps the transaction duration as short as possible

Transaction Begin

When the Transaction mode is set to Deferred, the following actions occur at the different Transaction Begin options:

Before Record Prefix	All updates at the Record level are collected as a group. At record update time, a transaction is opened, the changes are applied, and the transaction is committed.
Before Task Prefix	All updates at the Task level are collected as a group. After the Task Suffix, a transaction is opened, the changes are applied, and the transaction is committed.
Group	The updates are collected at the Group level, and are updated whenever a specific variable changes its value. When using the Group Level transaction, you must define the variable on which the group is based.

Locking Strategy Property

Deferred transactions can have the following lock set values:

- On-Modify - selecting the On-Modify locking strategy in a deferred transaction task does not issue locks to the database. These locks are handled through the eDeveloper locking mechanism. eDeveloper lock settings should be used as required. For more information about eDeveloper locking, refer to Databases in Chapter 2, Settings.
- No Lock

SQL Range Statement

The DB SQL Range does not apply to deferred transaction mode tasks.

The Magic SQL Range is a new eDeveloper feature that lets you send Where clauses directly to the database from a deferred transaction. For more information, refer to the Range/Locate section in Chapter 6, Programs.

Direct SQL

Direct SQL tasks can only be included within a physical transaction mode task.

Numeric Field Updates

This property is for Deferred Transaction Mode tasks only.

Previous versions of eDeveloper allowed only absolute numeric updates. For example, you could only set value X for field FLD1 (FLD1=X).

eDeveloper Version 9 lets you make differential updates for tasks that run in Deferred or Nested Deferred Transaction mode only. For example, you can update FLD1 by using the value FLD1+X (FLD1=FLD1+X).

The Update Style property has been added to the Field Properties sheet. The values for the property are:

- Absolute - a fixed value as allowed by previous versions of eDeveloper (FLD1=X)
- Differential - differential values (FLD1=FLD1+X)

This property is only enabled for the Numeric field type that has a normal storage type and relates to an SQL table. This property is enabled for Deferred and Nested Deferred mode task or to the Within Active Trans transaction mode task that evaluates to deferred in runtime. This is not relevant for ISAM files.

The Update Style property has also been added to the Select Real operation. Besides the Absolute and Differential values, you can choose the As Table option, which takes the Update Style property from the Table properties.

For more information, refer to the Column Repository section in Chapter 4, Tables, or to the Select Operation section in Chapter 7, Operations.

Update/Delete Statements

This property is for Deferred Transaction Mode tasks only.

You may determine the procedure by which eDeveloper creates a WHERE clause for each update or delete statement that is issued. The possibilities are:

- by Position only - Only fields that are part of the position will be included.
- by Position and Updated fields - The original value of the updated fields as well as the position fields.
- by Position and Selected fields - The original value of all the selected fields as well as the position fields.

A new property called Identify Modified Row has been added to the Table Properties. This property may accept any of the above values. Note that this property is only enabled for Deferred or Nested Deferred transaction tasks, or to the Within Active Trans transaction mode, which evaluates to the deferred transaction mode in runtime. The Identify Modified Row property has also been added to the DB Table repository.

For more information, refer to the Table Repository section in Chapter 4, Tables, or to the DB Table Repository section in Chapter 6, Programs.

Record Update Fail Before Call

This deferred transaction property lets you control the order of the execution of Data Manipulation statements.

In previous Magic versions, Data Manipulation statements (such as insert, update, or delete) were sent according to the Record Suffix order.

For example:

A parent task calls to a child task. The child task completes its processing and then returns to the parent task. The parent task must also complete its processing. Therefore, eDeveloper processes the Record Suffix operations of the child task before the Record Suffix operations of the parent task, even though changes to the parent task were made prior to calling the child task.

Nested Transactions

When a parent task calls a child task the following behavior can occur:

Parent Transaction Mode	Child Transaction Mode	Transaction Behavior
Deferred/Nested Deferred	Physical	The child transaction opens a physical transaction. The updates are sent to the database independent of the parent task.

Parent Transaction Mode	Child Transaction Mode	Transaction Behavior
Deferred/Nested Deferred	Deferred	The child transaction runs within the parent's transaction. The Data Manipulation (DM) statements of the child task are sent only when the parent task commits.
Deferred/Nested Deferred	Nested Deferred	The child task opens as a new deferred transaction. Its transactions are committed to the database independent of the parent task.
Physical	Physical	The child task submits database updates with the parent after each record or task. Parent and child tasks share the same transaction
Physical	Deferred or Nested Deferred	Runtime error.

A task that opens as a physical transaction can still have a deferred or nested deferred child providing that the parent does not actually open the physical transaction.

When a task is opened from an event handler that is defined for a different task, the transaction nesting is according to the runtime transaction and not the toolkit definitions.

For example:

Task A, defined as a deferred transaction, opens Task B, which is defined as a physical transaction. Task B updates the database as a physical transaction, and an event is triggered. The handler for this event is defined in Task A, and it calls Task C, which is a deferred transaction. Task C, a deferred transaction, cannot run under Task B, a physical transaction, and runtime error occurs. If, however, Task C was defined as Within Active Transaction, it would have opened as a physical transaction, and no runtime error would have occurred.

Transaction Tree

When running a browser-based program, the user can open several browsers at once. Separate browsers can run different tasks that have no transactional connection. eDeveloper supports separate browsers by separating deferred transactions into logical branches. Each branch is its own transaction tree without any connection to other branches.

eDeveloper lets you open a nested transaction in a modeless task for a deferred transaction. The behavior between parent and child tasks does not change from the previous version.

Open Transaction

When the calling task in the event tree has no open transaction, the called task opens a transaction according to its task properties. If the task is defined to open a transaction, the transaction will open in record or task level.

When the calling task in the event tree has an open transaction, the called task behavior depends on the called task properties listed below:

- Deferred Transaction - The task does not open a new transaction.
- Nested Deferred - The task opens a new transaction.

- No Transaction - The task behaves as a deferred task.

Close Transaction

A transaction is closed in the task where it was opened.

An exception is when a task opens a transaction where there are child programs of the same transaction. When the child programs remain open when the transaction is closed with the calling program, they will be left without a transaction.

The child programs must be part of a transaction. The first called child program will manage the transaction and all of the child programs in the transaction. The management of the transaction moves down the event tree when a parent task is closed. The transaction hierarchy is determined by the order of open tasks. When a transaction is closed and there are still open tasks within that transaction, another transaction is opened immediately and is managed by the first child task.

When the management of the transaction is moved to a new task, the new transaction is always opened in the task level regardless of the values defined in the properties of the task.

Runtime Tree Sample

A sample of a runtime tree is displayed below for a browser-based task with many open browsers.

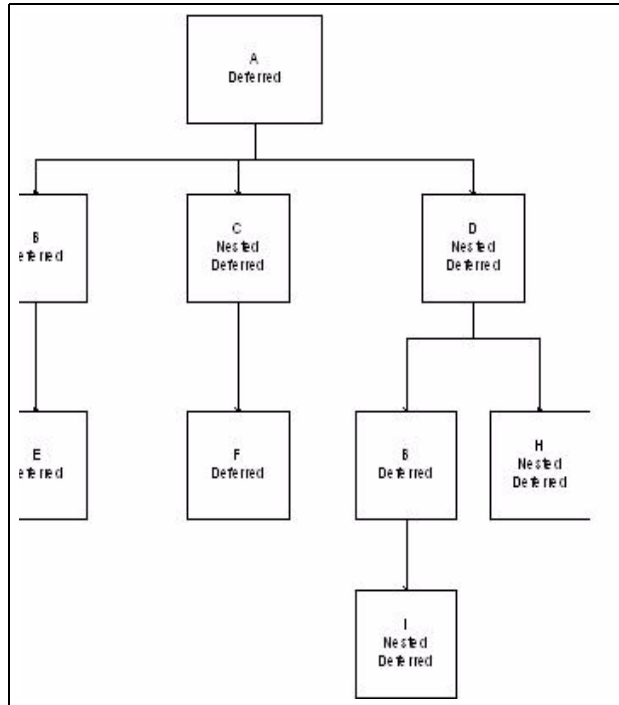


Figure 11-1 Transaction Tree Example

The example displays five concurrent open transactions. The five transactions start in the following tasks:

1. A
2. C
3. D
4. H
5. I

At the end of each transaction, the data is committed or rolled back according to the task logic.

Cache data cannot be shared between different tree branches before the data is committed. Task E cannot view data from Task I. After Task I commits its changes to the database, the updated data is available for Task E from the database.

Transaction Processing Recovery

Transaction processing recovery lets you determine eDeveloper's default behavior for the different database errors. For more information, refer to Chapter 12, Error Handling.

eDeveloper's Internal Transactions

eDeveloper has two types of internal transactions:

- Read - Used for Read Only operations on data files.
- Write - Used for Write operations on data files.



For more information, see DB Table Repositories in Chapter 6, Programs.

Mapping Transactions to Databases

To implement the transaction processing mechanism, eDeveloper utilizes the transaction processing facilities of the underlying databases. Because each database has its own transaction facilities, eDeveloper maps its internal transactions to those available in the various databases.

ISAM Databases

According to the way ISAM databases implement transaction processing, eDeveloper maps its internal transactions in the following manner:

- Read-transactions and Lock-transactions have no implementation in ISAM databases. Therefore, eDeveloper does not transfer them to the ISAM database gateways.
- A Write-transaction is mapped to the ISAM transaction that logs the updates and is able to do rollback. Such a transaction can lock all or part of the table.

SQL Databases

According to the way SQL databases implement transaction processing, eDeveloper maps its internal transactions in the following manner:

- A Read-transaction is mapped to a R/O (read only) transaction. This transaction takes snapshots of the database situation when the transaction is opened.
- If the Access mode of all tables in a Lock or Write transaction is Read, the Lock or Write transaction is mapped to a R/O (read only) transaction. Otherwise, a Lock or Write transaction is mapped to a R/W (read/write) transaction. This transaction locks all the records referred within the transaction scope.

Deadlocks and Transaction Processing

In a multi-user environment, deadlocks are a common cause of transaction failure. A deadlock situation occurs when two users are each waiting for a resource owned (that is, locked) by the other.

The various Database Management Systems (DBMSs) behave differently when a deadlock occurs. Some DBMSs are able to detect the deadlock and issue a warning. In this case, eDeveloper rolls back the transaction and continues execution according to the Error behavior strategy setting for the transaction.

Because some ISAM databases may hang in deadlock situations, eDeveloper provides a deadlock prevention mechanism for ISAM and SQL databases. To activate the deadlock prevention mechanism, set the Deadlock Prevention setting in the Environment dialog to Yes. The deadlock prevention mechanism locks exclusively all tables opened with Write access, in the same order as defined, for the period of the transaction. Temporary tables are not locked.

Rollback

The Rollback function can be used to roll back a transaction to a specified nesting level, or to abort the transaction completely. The Rollback function receives as a parameter the number of nesting levels for the rollback point. The parameter number must be a number of open nested transactions. A parameter value of zero will roll back the entire transaction to the first level. After forcing a rollback, eDeveloper resumes processing according to the rollback level defined in the Error Behavior Strategy property.

The Rollback function lets you display a confirmation dialog to the end-user before the actual rollback operation starts.

Rollback Behavior for Browser-Based Programs

When a browser-based program is rolled back, all the open tasks in the transaction will be closed except for the task that opened the transaction. Nested tasks are not affected.

For example, when a task in a different transaction rolls back an open parent transaction, the child task will stay open and will not be affected.

An exception is when a child program is called by a handler triggered from a higher level. The child program opens its own task level transaction. If there are several child programs, all of them will be in a new transaction and will be managed by the first called child program.

A child program in a task should be closed when it is called by the Call operation.

Sync Data

This property is part of the Call operation. It lets you do the following:

- Sends the parent task's Data Manipulation statements before the child task's Data Manipulation statements by selecting Yes.
- Lets eDeveloper control the order of the execution of Data Manipulation statements by selecting No.
- Defines a logical expression that can be evaluated as Yes or No.

The Sync Data property is enabled for Call Task and Call Program operations at the Record, Control and Handler levels.

For more information, refer to the Call Operation section in Chapter 7, Operations.

eDeveloper Cache

The basic assumption behind the need for caching is that the same data will be needed more than once. Therefore, if data can be re-fetched without performing disk I/O operations, overall performance will be enhanced.

Unlike general purpose disk caches that employ generic algorithms for their caching, the eDeveloper Cache lets you fine-tune the cache according to your knowledge of the physical data and the nature of the application, and even on a user-by-user basis in an application.

The eDeveloper Cache does not call any file manager as other hardware or software caching methods might. The eDeveloper Cache therefore avoids the additional file manager overhead of traditional methods.

What Can Be Cached

The eDeveloper Cache can be used for Main tables and for linked tables. The eDeveloper Cache is implemented on a task and Main table and link basis. This means that if a parent task and a subtask are accessing the same table, and

both tasks enabled caching on the table, then two caches will be used. The same applies for two link operations to the same table in a task. In this case also, two caches will be used.

When is the Cache Used

Data can be read from the cache only when eDeveloper does not need to issue a lock for the row. The need for a lock is determined by the following three factors.

- The Locking strategy, defined under the Enhanced tab of the Task Properties dialog.
- The Cache strategy, defined under the Enhanced tab of the Task Properties dialog.
- The definition (in the DB Table repository) of the Access and Share modes with which the table was opened.

A lock is issued only for tables that are opened with write access and share. The timing of the lock is determined by the locking strategy.

Moving from Query mode to Modify mode, and from Modify mode to Query mode, is done using the cache. The records in view when switching modes are not refreshed.

A Sort operation automatically creates caches for the sort file.

When accessing Create mode, or when User Locate, User Range, or Sort operations are taking place, the cache is released.

Toggling between Modify and Query modes in eDeveloper V9 causes eDeveloper to access the cache and eliminates the need to re-fetch data from the database.

Activating the Cache Size

In the task level DB Table repository it is possible to define the cache of a linked table as disabled or enabled. This definition is set by using a Yes or No

setting. When the Cache column in the DB Table repository is set to No, caching is disabled for the linked table in the task. If the column is set to Yes, the table cache is used.

The default value in the task's DB Table repository depends on the properties set in the Table Properties dialog.

Changes to Program Behavior

Use of the eDeveloper Cache changes the behavior of programs. Whenever data is found in the cache, the data may not be the same as the data in the table. However, before updating the data, eDeveloper will always read the row from the disk. As an example, consider a simple Online program created by the Automatic Program Generator (APG). Whenever the insertion point is moved from one row to another, the dataview of the new row is automatically reread.

In the case of another user who has changed the data, we will not see the change when we park on the row (if the data was in the cache), but if we try to modify the data, the current values data will be read from the disk, and the following message will appear:

Record has been updated, Restart.

There are cases where the application definition does not allow the cache to be used. For example, assume an order entry application where the Prices table is accessed for read with Write Share. If a price had been changed by one end-user, and the new price of an item resides in that user's cache, no other end-user will be able to see the new price.

Cache and Resident Tasks

Whenever an eDeveloper cache is assigned in a resident task, the cache will not be terminated when the task is exited, but only when eDeveloper returns to the menu. Thus, if a task is reading the same data from a linked table, and all of the data is cached, then the second time the task is called, the cache will be reused.

Cache and The Rollback Operation

For the Rollback operation, all the caches that were updated during the transaction are invalidated. The invalidation causes all data to be erased from the cache. Only caches that had Update Delete or Insert operations will be invalidated.

Cache and Client/Server

The eDeveloper Cache resides on the client. Using the eDeveloper Cache in a Client/Server environment will not only reduce disk I/O, but will also generate less network traffic. This is an important factor, as the network is usually the bottleneck for Client/Server applications.

eDeveloper Cache Internal Implementation

Every row that eDeveloper reads is inserted into the cache. In addition, any modification, deletion, or insertion of new rows, is also placed in the cache. Whenever a row that already exists is reread by eDeveloper, it will supersede the previous value. If a row that does not exist in the cache is read, the eDeveloper Cache automatically places it in the cache. If the cache is full, an LRU (Least Recently Used) algorithm is used to determine which element is to be dumped from the cache.

The use of the LRU mechanism ensures that data that is read more frequently will have a lower chance of being swapped out of the cache.

Error Handling

The Error Handling feature lets you override eDeveloper's default behavior for the different errors that can arise during an application's execution.

Error Handling Mechanism

There are two levels in which you are able to control eDeveloper's behavior when an error occurs. The first level consists of two, pre-defined error strategies that you can select from the Error Behavior Strategy property in the Task Properties dialog on the Enhanced tab.

The second is a more sophisticated error handling level that lets you write actual eDeveloper code for the errors, using error handlers. eDeveloper provides a list of known and expected errors that can be intercepted, and lets you handle other errors specific to your DBMS error code. This level also provides you with a list of new engine directives to control the behavior of the eDeveloper engine after implementing the error handler.

The errors that are described in this chapter are for database-related errors only.

Error Behavior Strategies

eDeveloper provides two pre-defined strategies – *Abort* and *Recover*. The strategy option is set in the Task Properties dialog as shown on the next page.

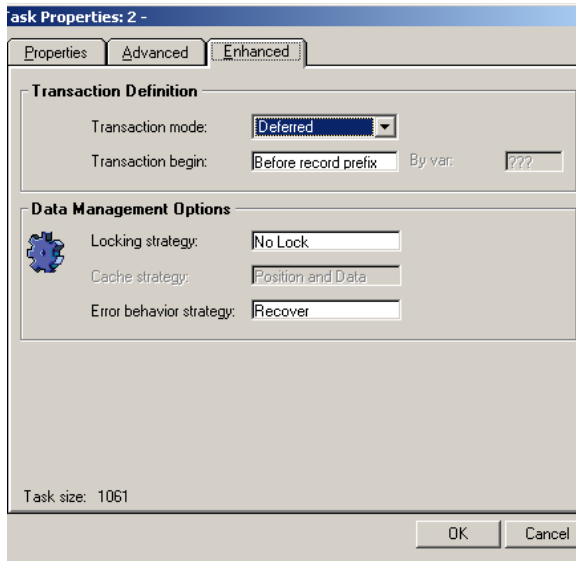


Figure 11-2 Selecting an Error Behavior Strategy

If the strategy is chosen carefully, you will not have to change any of the defaults of the error handling task properties or write handlers for error handling. Of course, you can still change the default settings by writing error handlers in those places where the default strategy does not apply.

Abort Strategy

With the Abort strategy, whenever an error occurs eDeveloper rolls back the current transaction, whether physical or deferred, removes the current dataview, and aborts the task in which the error occurs. The Abort strategy does not roll back the current transaction for errors that the end-user can recover from, such as Locking and Incorrect Login errors.

Recover Strategy

With the Recover strategy, whenever possible, eDeveloper will keep the current dataview, stay on the current task, and allow the end-user to recover from the error. The Recover strategy lets the end-user continue working in the application after an error has occurred.

Error Strategy Behavior

The following is a list of errors and the specific behavior that will be automatically applied depending on the strategy selected. For a detailed description of each strategy behavior, see the Engine Directive section.

Error	Abort Strategy	Recover Strategy
Unable to lock table	Auto Retry	Auto Retry
Locked row	Auto Retry	Auto Retry
Max connections	Abort Task	Auto Retry
Duplicate index	Abort Task	User Retry
Constraint failure	Abort Task	User Retry
Trigger failure	Abort Task	User Retry
Record has been updated	Abort Task	Rollback and Restart
Record has been changed by another user	Abort Task	Rollback and Restart
Table open failed	Abort Task	Abort Task

Error Handlers

You can write an error handler in the Handlers table of the Task Execution repository. This is the same table that lets you define event handlers.

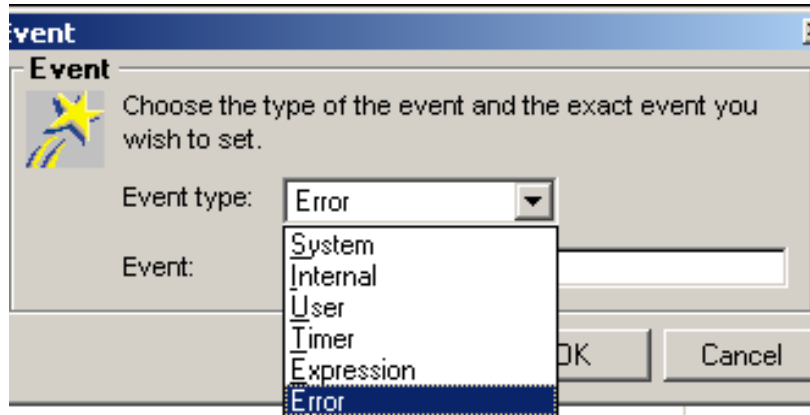


Figure 11-3 Creating an Error Handler

An error handler has the following properties:

- Level
- Event
- Details
 - Engine Directive
 - DBMS Message (Msg.)
- Scope
- Propagate
- Enable
- Operations

Level

The level for an error handler is defined at the handler level. They are similar to other handlers at the task level.

Event

The Event property lets you specify the error type from the following list of error types:

- Any
- Locked row
- Duplicate index
- Constraint failure
- Trigger failure
- Record has been updated
- Record changed by another user
- Insert/Update/Delete failure
- Unmapped errors

If you do not specify an error type, the value Any is assigned to the property. The 'Any' Error type can be executed for any error type that occurs.

Engine Directive

When defining a user-defined error handler, the Engine Directive property determines what action occurs after the execution of the error handler.

As Strategy

eDeveloper verifies the error strategy for the current task, and acts in accordance with it. Note that all other error strategies are exceptions to As Strategy.

Abort Task

The Abort Task aborts the task from the transaction source, rolls back the data, and removes the dataview.

Rollback and Restart

Rollback is the same for all errors and transaction modes, but may vary according to the task's mode (Online, Batch, Browser), and according to where the transaction was opened.

Task Mode	Transaction on	Behavior
Online or Browser	Record	Cursor stays on the record that began the transaction.
	Task	Cursor stays on the task that began the transaction, and re-fetches the data and position of the first record.
Batch	Record	Goes to the next record.
	Task	Aborts the task.

Automatic Retry

On some errors, such as `Locked Row`, `Unable to lock table`, and `Max connections`, recover means that eDeveloper retries the operation automatically, without input from the end-user. Only a few errors can be automatically retried by eDeveloper as displayed in the table below. Most errors require user intervention as explained in the User Retry section.

Error	Behavior
Unable to lock table Locked Row Max Connections	eDeveloper automatically retries the operation. eDeveloper does not rollback the transaction, regardless of whether it is physical or deferred, and does not rollback the dataview. The cursor parks on the same record and retries locking or connecting until the time-out value expires.

Error	Behavior
Duplicate Index	Deferred transactions only. eDeveloper automatically retries the entire physical transaction in the flush phase, which is when eDeveloper writes the data to the database.
Constraint failure	
Trigger failure	
Table open failed	Not applicable. eDeveloper never uses the Automatic Retry option for these errors.
Table Create err	
Table delete failed	
Table copy failed	
Commit transaction failed	
Open transaction failed	
Internal transaction err	
Deadlock	
Connect failed	
Fatal err	
Insert/Update/Delete failure	
SQL execution err	
Invalid SQL command	
Invalid open Query Exp	
Invalid table name	
Table does not exist	
Cannot modify R/O table	
Record has been updated	
Record has been changed by another user	
Record lost	

In the Deferred Transaction mode, Automatic Retry refers to all errors that occur in the flush phase of the transaction, for the commands accumulated during the deferred transaction. The physical transaction is rolled back and then restarted, including all the operations that were already implemented.

User Retry

The User Retry behavior will not automatically rollback the transaction or dataview, but will leave them and return the control to the end-user, positioning the cursor on the record where the error occurred. **User Retry is not applicable for physical tasks.** The table below displays the errors that eDeveloper lets the end-user recover from.

Error	Behavior
Incorrect Login	<p>This is relevant for all task types.</p> <p>In Runtime and Toolkit modes, the login window opens for the end-user to try to login again.</p> <p>In Background mode, the login window does not open.</p>

Error	Behavior
Duplicate Index Constraint failure Trigger failure	<p>Online and Browser Client Deferred Transaction - Only the flush phase is rolled back. The dataview stays with the current values.</p> <p>At Record level, eDeveloper stays on the record.</p> <p>At Task level, eDeveloper stays with the task, and positions the cursor on the first record.</p> <p>Physical The transaction is rolled back.</p> <p>The dataview is recreated.</p> <p>At the Record level, eDeveloper stays on the record and the cursor appears at the beginning of the current record.</p> <p>At Task level, eDeveloper stays with the task, and positions the cursor on the first record.</p> <p>Not applicable for Batch tasks.</p>
Record has been updated	<p>Online The dataview will display the values after the update, and eDeveloper will stay on the current record. If the transaction is on the record, the record will be rolled back. If the transaction is on the task, the task will not be rolled back.</p> <p>Not applicable for Batch tasks.</p>

Error	Behavior
Locked Row	Not applicable. eDeveloper never uses the User Retry option for these errors.
Table open failed	
Table Create err	
Table delete failed	
Table copy failed	
Commit transaction failed	
Open transaction failed	
Internal transaction err	
Unable to lock table	
Deadlock	
Max connections	
Fatal err	
Insert/Update/Delete failure	
SQL execution err	
Invalid SQL command	
Invalid open Query Exp	
Invalid table name	
Table does not exist	
Cannot modify R/O table	
Record lost	
Record has been changed by another user	

Ignore

Ignore causes eDeveloper to skip the error and to continue to the next record as explained below.

- Physical
 - Batch - continues to the next record.
 - Online and Browser Client - continues to the next record according to the operation that is performed. If it is not possible to park on the next record because it is locked, eDeveloper will park on the previous record.
- Deferred - If the error occurs during a non-flush phase, eDeveloper behaves as if it is a physical transaction. If the error occurs in the flush phase, eDeveloper skips the current command, and continues to the next command.



The Ignore option for the Physical mode corresponds to the Skip option in previous Magic versions. It skips the whole record cycle - Record Prefix, Record Main, and Record Suffix. The Ignore option for the Deferred mode only skips the relevant command and not the whole record cycle, because the commands in the Deferred mode are not executed in correlation with the record cycle.

DBMS Errors

You can specify whether eDeveloper displays DBMS messages for fatal or unexpected errors.

- Yes - displays DBMS messages. If this parameter is set to Yes, it overrides the Display Full Messages setting (Settings/Environment/Preferences).
- No - does not display DBMS messages.

Scope

The Scope property allows for the following options:

- Task - executes the error handler triggered in a specific task.
- SubTree - executes the error handler triggered in a specific task and its sub-tasks.
- Global - executes the error handler triggered only in the Main Program.
This option is relevant only for handlers defined in a component, and not in the internal (hosting) application.

Propagate

You can propagate an event to an event handler defined in a higher level. When eDeveloper regards the event as not handled, the dispatcher searches for an event handler at the next level.

Enable

Lets you enable or disable the error handler. This property can be an expression. If it evaluates to **No**, the handler will not intercept the error.

Operations

The number of operations listed in the Operation repository.

Error Information

The information about the error is available to you through a series of functions that can be evaluated even before entering the handler. With this information you can decide to disable or enable a user-defined error handler. The functions are:

- ErrTableName - Returns the physical name of the table on which the error has occurred. If you have main and linked tables, this function can be used to determine where the error occurred. This function is only relevant for errors that are connected to a table.
- ErrDatabaseName - Returns the name of the database on which the error has occurred.

- `ErrDbmsCode` - Returns the DBMS error code. This can be useful for fatal and unexpected errors, assuming that you are familiar with your DBMS internal error codes.
- `ErrDbmsMessage` - Returns the original DBMS error message for fatal or unexpected errors.
- `ErrMagicName` - Relevant if the error handler is set to Any error. Returns the eDeveloper literal of the error. This function can be used to handle several errors that share the same operations.
- `ErrPosition` - Provides a reference to the position of the record on which the error occurred.

For more information, refer to Chapter 8, Expression Rules.

Runtime Error Handling

When eDeveloper encounters an error situation, the eDeveloper error handling mechanism searches for a defined error handler to intercept the error in the same task, according to the error name. If the error handler exists, eDeveloper performs the operations that are defined for the handler, and then performs the corresponding action according to the Engine Directive property that was evaluated by the handler.

The search for an error handler is similar to the search for an event handler. If the handler does not exist in the task, higher task levels are searched for an appropriate handler, according to the error name.

If the required error handler does not exist, and a handler for the Any error is defined, the error handler for the Any error is executed.

Task Range According to a Record's Position

An important feature for error handling is to provide you with a method to display the error record or the range of records according to its position.

In eDeveloper Version 9:

- You can locate a task according to a specific position.
- The CurrPosition function returns the current record's position. For more information, refer to Chapter 8, Expression Rules.
- The position of the record on which the error occurred is returned through the ErrPosition function.

Range/Locate Properties

The following Range/Locate properties are located in the Usage field of the Range/Locate dialog. These properties are enabled only when you define an expression in the Position field.

- Range From - Lets you enter an expression that indicates the position of the record from which the task will range. There is an AND operation between the Range field in the Select operation, the task's range expression, and the Range From property. If the record does not exist, no records are displayed.
- Range On - Lets you enter an expression that indicates the position of a single record to which the task displays. There is an AND operation between the Range field in the Select operation, the task's range expression, and the Range From property. If the record does not exist, no records are displayed.
- Locate Position - Lets you enter an expression that indicates the position of the record to which the task will locate. You can have a logical AND operation between the locate field in the Select operation, the locate expression on the task, and Locate Position.

These properties are relevant only for the Main table of the task. You cannot perform range or locate on linked tables according to their positions.

Applications from Previous Versions

Error Property Values for Online and Batch Tasks

The table below shows how eDeveloper Version 9 sets the error strategy for online programs imported from previous Magic versions.

Previous Magic Version	New
Task (only the highest level is referenced) Transaction - Abort in the On Error	Sets the error strategy of the new task to Abort.
Record (only the highest level is referenced) Transaction - Abort in the On Error	Sets the error strategy of the new task to Abort.
Task (only the highest level is referenced) Transaction - Retry in the On Error	Sets the error strategy of the new task to Recover.
Record (only the highest level is referenced) Transaction - Retry in the On Error	Sets the error strategy of the new task to Recover.
Record level transaction - Skip in the On Error	Sets the error strategy of the new task to Abort. Adds an error handler for the error type Any with the Engine Directive set to Ignore.

The error strategy for batch programs imported from previous versions of Magic is described in the tables below.

Batch Tasks - On Record Locked

Previous Versions	New
On Record Locked = Retry	Default for both error strategies - no change needed.
On Record Locked = Abort	Adds an error handler on error Locked Row with the Engine Directive of Abort Task.
On Record Locked = Skip	Adds an error handler on error Locked Row with the Engine Directive of Ignore.

Batch Tasks - On Access Fail

Previous Version	New
On Access Fail = Skip	No error handlers
On Access Fail = Retry	Error handler for <i>Any</i> error with the Enabled property set to an expression that checks if there is no transaction. The handler has the Engine Directive set to Auto Retry.
On Access Fail = Abort	Error handler set to <i>Any</i> error, if there is no transaction. The handler has the Engine Directive set to Abort Task.

Batch Tasks - On Change	
Previous Version	New
Transaction - Abort in the On Error	Sets the error strategy of the new task to Abort.
Transaction - Retry in the On Error	Sets the error strategy of the new task to Recover.

End-User Menus & Help 12

This chapter explains how to define the menus and help in your application. You can determine menu contents, accessibility, size, and format. The different menu formats include pulldown menus, toolbar buttons that activate menu options, and context menus. eDeveloper provides a facility to design end-user help information in the form of help screens, prompts, and tooltips. You can design help screens that can be viewed when the user requests help from forms, or from individual form controls. In addition, you can design help messages that are automatically displayed for form controls. When you define a data item, a model, a form, a control, or a menu item, you can specify associated help information.

In this chapter:

- | |
|------------------------------|
| • Menu Formats |
| • Menu Repository |
| • Menu Definition Repository |
| • Menu Authorization Options |
| • Menu Properties Dialog |
| • Help Screen Repository |
| • Help Types |

Menu Formats

Pulldown menus cascade down from the menu bar. Toolbar buttons are shortcuts to menu options. Context menus are activated by clicking the right

mouse button, and are displayed where the mouse is right-clicked on the screen.

Pulldown Menus

For every new application, eDeveloper generates a set of default pulldown menus, including File, Edit, Options, and Help menus, with all the basic options a Runtime application needs. You can then change, delete, or extend these default menus using the procedures described in this chapter.

You can design your application to have up to ten levels of pulldown menus, including the menu bar as the top level. The menu bar entries are displayed when the user selects the application in Runtime mode. For most applications, you would probably have at least one menu entry that has a program name and connects the menu to an entry in the Program repository.

You can define a toolbar button as a shortcut to any pulldown menu option.

Context Menus

Context menus appear when the user clicks the right mouse button in Runtime mode.

eDeveloper generates a default context menu for each application. You can also define other context menus that override the default context menu. You can associate context menus with specific tasks to make these menus context-sensitive.

You can design your application to have up to ten levels within a context menu.

Menu Repository

You define pulldown and context menu structures in the Menu repository.

The Menu repository contains one default pulldown menu structure and one default context menu structure. These default menu structures cannot be deleted from the Menu repository.

In the Menu repository you can define additional context menu structures to have as many context menus as you want in an application. However, you cannot define additional pulldown menu structures because you can only have one pulldown menu bar in an application.

You can zoom from each menu structure entry in the Menu repository to a Menu Definition repository and define the specific menus and their options for each menu structure. You can define up to ten menu levels for each menu structure.

The default context menu set is automatically present for applications that run as an eDeveloper client. When you define a context menu set in addition to the default context menu and you associate this set to a specific task, the defined set overrides the default set when the context menu is activated from the context of the associated task. If a task has no context menu associated with it, the default context menu appears when the right mouse button is clicked from the context of that task.

You can associate a menu structure to a specific task in the Attached context property in the Advanced tab of the Task Properties dialog.

Entries in the Menu repository are either default or user-defined menu structures.

Menu Structure	Description
Default	When a task does not have a specific pulldown or context menu attached to it, eDeveloper displays the default pulldown and context menu structures. The pulldown and context menu structures imported from previous Magic versions are imported as default menu structures.
User-defined	User-defined context menu structures can be attached to any online task for an eDeveloper client application.

Menu Name

The Menu Name column is the only accessible column in the Menu repository. This column contains the logical name of the menu structure, which is not the same as the menu name that the end-user sees at runtime.

Menu Type

The cursor cannot park in the Menu Type column. When you create a new menu structure entry, the new Menu Type will always be Context menu.

Menu Definition Repository

You can zoom from each menu structure entry in the Menu repository to a Menu Definition repository and define the specific menus and their options for each menu structure. You can define up to ten menu levels for each menu structure.

Entry Types

The menu entry types in the Menu Definition repository are:

- Program - Contains a program number from the Program repository. The corresponding program name appears on the menu. When selected by the end-user at runtime, this program is executed.
- OS Command - Contains an operating system command. When selected by the end-user at runtime, this operating system command is executed.
- Event - Contains an eDeveloper Action. When selected by the end-user at runtime, the function associated with this eDeveloper Action is executed.
- Separator - Contains a horizontal line at this position. Available for context menus only.
- Menu - Contains an end-user menu.

Entry Text

What you enter in the Entry Text column becomes the text entries that the end-users see displayed in the menu at runtime.

For a pulldown menu at the top level, the name in the Entry Text column will appear on the menu bar. For pulldown submenu levels and for all context menu levels, the name in the Entry Text column becomes a menu option that invokes either a submenu, a related display, or an action.

You can specify an access key for any level pulldown or context menu by putting an ampersand, &, before the letter to be underlined. When the menu is displayed at runtime, typing the underlined letter selects the menu option.

Entry Name

This column contains the logical name of the menu entry itself, which does not appear in the menu that the end-user sees. Note that the cursor can park on the Entry Name parameter for pulldown menus.

Menu Parameters

Each parameter category in the Menu Params column is associated with one menu entry. You can zoom from the Menu Params column to access the menu definition of that entry and see the next submenu level in the menu hierarchy.

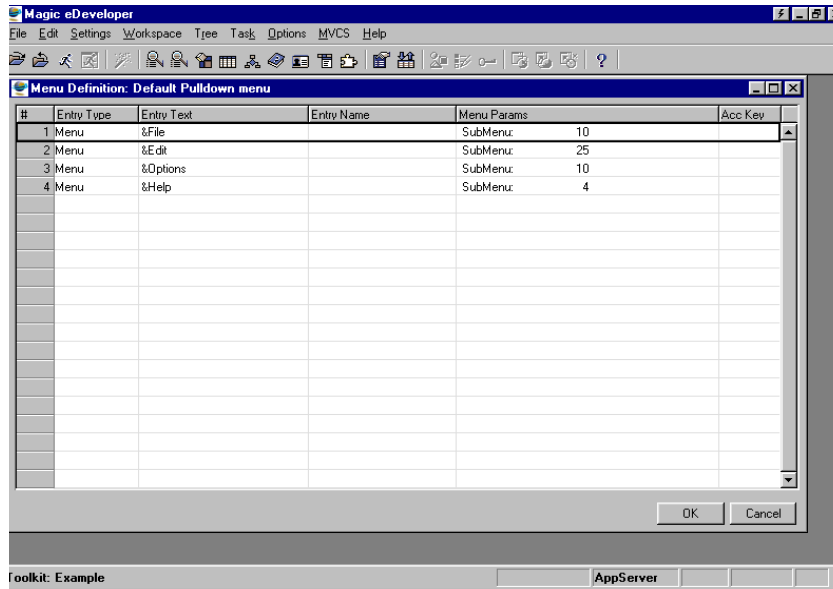


Figure 12-1 The Menu Definition Type Entry

Entry Type	Menu Params	Meaning	Values
Program	Program Number	Zoom to display and select a program from the Program list.	A program number.
OS Command	OS Command	An immediate exit to an operating system command. Type in any executable operating system command, including scripts or batch files, or zoom to select a file from the Windows File dialog.	An alphanumeric string. An expansion box opens automatically. If the OS command is left blank, eDeveloper exits to an operating system shell whenever this menu option is selected at runtime.
Event	Action	Zoom to display and select an action from the Action list, to give the end-user control of the environment.	Any eDeveloper Action.
Separator	N/A	Display a horizontal line in the menu to segregate groups of preceding items from subsequent items.	None.
Menu	Sub-lines	Zoom to display a Menu Definition repository for the next submenu level in the menu hierarchy.	Number of options on this submenu level.

Menu Access Key

The Acc Key column is only enabled in Default Pulldown menus.

When the entry type is not a menu or separator, you can zoom from the Acc Key column to open a dialog and define shortcut keys to the menu options.

Menu Authorization Options

When defining menus, you can click *Authorize* on the *Options* menu to open a Rights Assignment dialog, as shown in Figure 12-2, and set the authorization for developer access rights to the menu systems. You can specify which developers have rights to Query, Modify, Delete, and Create.

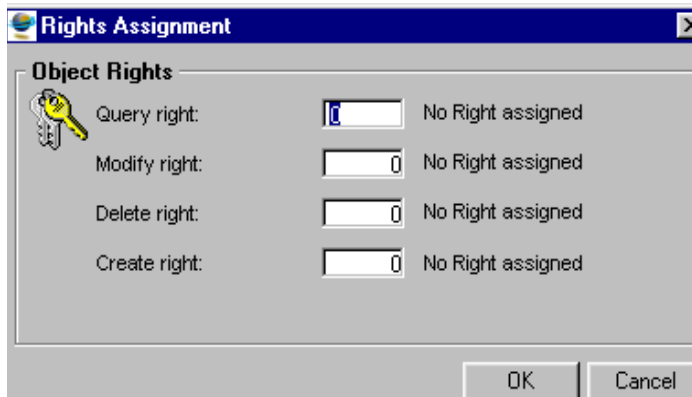


Figure 12-2 The Rights Assignment Dialog

End-user access rights to menus and individual menu items are specified in the Menu Properties dialog, accessed by selecting *Edit/Properties*, and described below.

The Menu Properties Dialog

You can select *Edit/Properties* from a Menu Definition repository to open the Menu Properties dialog, as shown in Figure 12-3. The contents of each Properties dialog varies according to the type of Menu entry to which it applies.

From the Properties tab you can assign an Access Right to the menu item. The Access Right number you enter in the Menu Properties dialog determines whether or not the end-user has access to this menu item. An end-user without the right specified here will not see this entry on the menu.

A special Menu Properties dialog opens for OS Command menu entries. In addition to the Access Right property, you can also specify how to open the OS session in the Show and Wait fields.



Figure 12-3 The Menu's Properties Dialog

The Menu Properties dialog contains three tabs:

- Properties tab
- Toolbox tab
- States tab

Properties Tab

Rights

This property determines the end-user access to menu entries of an application menu at runtime.

Help

This property determines if a help screen is accessible for an application menu. Zoom to the Help list to select the appropriate Help screen to connect to the application menu. For more information, refer to the Help Screen Repository section in this chapter.

Prompt

This property determines if a Prompt Help is connected to an application menu. Zoom to the Help list to select the appropriate Prompt Help to connect to the application menu.

Show

The Show property is only accessible for OS Command menu entries. This property determines the appearance of the external program. The possible values of the Show property are Hide, Normal, Maximize, and Minimize.

- Hide - Specifies that the external program will run behind the visible windows. Certain external programs, such as drvspac.exe, will ignore this setting because they cannot be hidden.
- Normal - The default setting. Specifies that the external program will run in the top visible window.
- Maximize - Specifies that the external program will run in the top visible window and that its window will be maximized to the full screen.
- Minimize - Specifies that the external program will run minimized, and that only its icon will be visible.

Wait

The Wait check box lets you specify if the eDeveloper program will wait for the called program to complete before it continues. The developer can only access this property from the OS Command line.

- Not Selected - This is the default setting. When the Wait check box is not selected, the eDeveloper program will not wait for the called program to complete before continuing.

- **Selected** - When the Wait check box is selected, the eDeveloper program will wait for the called program to complete before the eDeveloper program continues.

Arguments

This property is only available for Program menus.

The Argument repository, accessed by zooming from the Argument (Arg) field, displays the passed arguments.

You access the list of variables by zooming from the Variable (Var) column in the Argument repository. For more information, refer to Call Operations in Chapter 7, Operations.

Toolbox Tab

Image For

This property specifies where the option image appears. The available options are:

- **None** - no menu image is displayed at all
- **Both** - the menu image is displayed both in the toolbar and menu
- **Toolbar** - the menu image is displayed in the toolbar only
- **Menu** - the menu image is displayed in the menu only

Tool Image

A User Image file based on the eDeveloper Image file description. This image will be displayed on the toolbar button.

Tool Number

A number of an Internal eDeveloper image. These images are embedded inside eDeveloper and are available for selection. Zoom from this property and Select an image from the Get Image box.

Tool Group

The number of the group where the tool button is located (0-9999). A separator will be inserted between groups.

Tooltip

A text string, up to 50 characters long, that will be displayed when the mouse cursor pauses on the button tool.

States Tab

Checked

Select this check box to display the menu entry with a check mark. A tool button will appear pressed if this check box is selected.

Visible

Select this check box to display the menu entry or toolbar button, or clear this check box to hide the Menu entry or toolbar button.

Enabled

Select this check box to show the menu entry or toolbar button as enabled, or clear this check box to show the menu entry or toolbar button as disabled.

Notes: When the external application runs hidden or silent, it is advisable to redirect standard output messages of the external program from the screen to either a disk file or a null file.

Click on a toolbar button to activate the same pulldown menu operation as if it was selected from the menu.

The toolbar will display all icons as long as they fit into the toolbar window.

Help Screen Repository

You define end-user Help in the Help Screen repository, which contains the columns described below.

- # - This column contains an automatically generated sequential number used by eDeveloper as a Help identifier. You cannot edit this column.
- Name - A descriptive name for the Help.
- Type - The type of Help.
- Folder - If you want this Help to reside in one of the Help Screen folders defined in the Navigator, click the Folder box and select the appropriate folder for this Help.
- Public Name - Defines the public name of the Help by which it will be called by an Internet requester or a Call Remote operation. The public name must be unique within the application file.

[illegible]

Figure 12-4 The Help Screen Repository

Help Types

eDeveloper Version 9 supports five different Help types.

- Internal - Magic V5.7-compatible Help
- Prompt - informative message to be displayed in the status bar
- Windows - connection to a Windows WinHelp or HTML Help system
- Tooltip - information about buttons on the toolbars
- URL - browser-based Help

You define each of the eDeveloper Help Types in the Help Screen repository, as shown in Figure 12-4. The default value for a Help Type is Internal. You can click the Type box and select another type. Note that the actual contents of Windows Help and URL Help are defined outside of eDeveloper.

Internal Helps

Magic V5.7-compatible Helps are referred to as Internal Help. In this type of Help you zoom to a default edit control and place the text of the Help there.

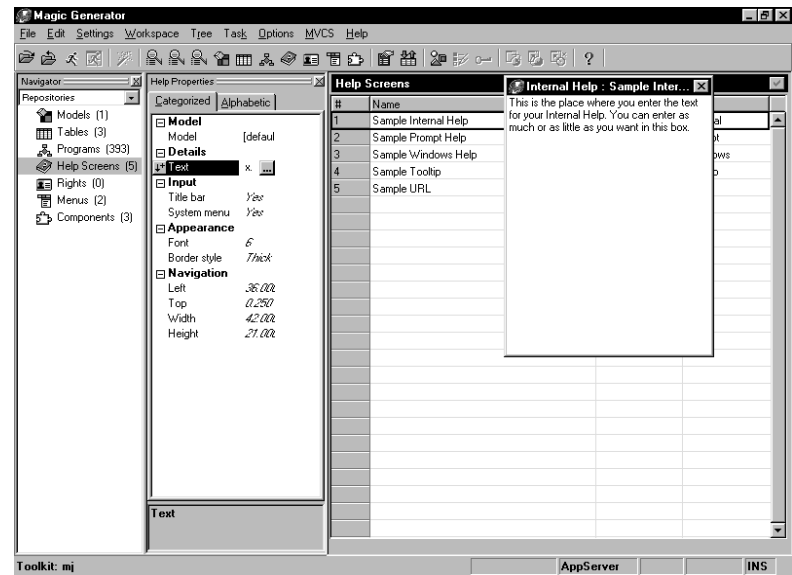


Figure 12-5 Internal Help Properties

From the Internal Help Edit screen you can select *Edit/Properties* or press CTRL+P to open the relevant Help Properties sheet.



In the Text property on the Help Properties sheet, you can enter the message to be displayed on the status bar when the Prompt help is invoked. You can click the Ellipsis button or zoom to open a text box, as shown in Figure 12-5, and see the entire message, if the space provided is not sufficient.

The Internal Help properties that you can set are organized by Input, Appearance, and Navigation.

Input

- Title Bar - Specifies whether the Help form has a title bar. The title bar text is taken from the Name column of the Help Screen repository. Without a

title bar, the end-user cannot move the form nor display Minimize and Maximize buttons.

- System menu - Specifies whether or not the end-user can access the Help screen from the System menu. The System menu is only enabled when Yes is specified.

Appearance

- Font - Specifies the font style for the Help screen. Zoom to open the Font repository.
- Border Style - Specifies the style of the border. The available styles are Thick, Thin, and No Border.

Navigation

- Left - Positions the Help screen at the left of the screen.
- Top - Positions the Help screen at the top of the screen.
- Width - Specifies the width of the Help screen.
- Height - Specifies the height of the Help screen.

Prompts

From a Prompt Help line in the Help Screen repository, you can select *Edit/Properties* or press CTRL+P to open the relevant Help Properties sheet and create a Prompt Help.

In the Text property, you can enter the message that will be displayed on the status bar when the Prompt help is invoked.

Windows WinHelp Connections

A Windows-type help is a hook to the Windows Help system. When a Windows-type help is declared in the Help Screen repository, you can zoom to the Windows Help Properties sheet, as shown in Figure 12-6.

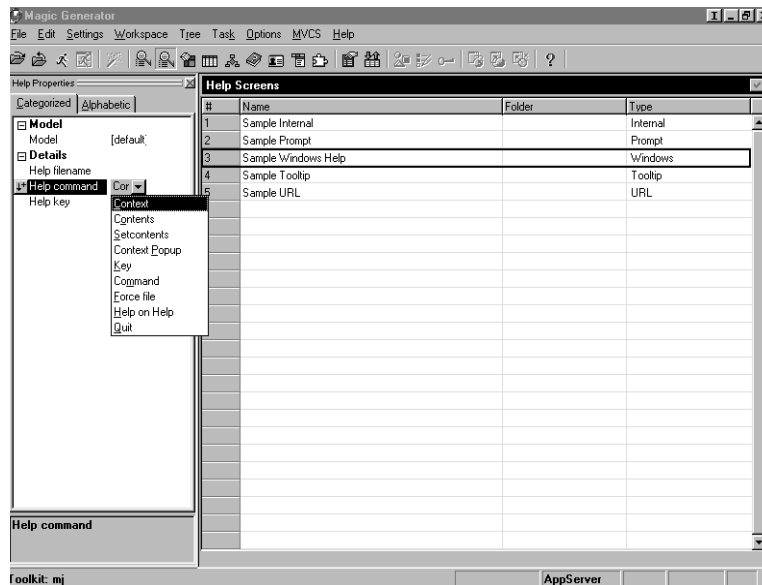


Figure 12-6 Windows Help Properties

The Windows Help properties are:

- Help filename - This can be inherited from the Environment dialog entry or you can zoom to an Open File dialog to select another file name. Once another file name is selected, that file will be the default within the application until a different file name is selected.
- Help command - From the Help command pulldown menu, you can select a command to be passed to the Windows Help Engine. The available commands are:
 - Context - Displays Help for a particular topic identified by a context number defined in the [MAP] section of the .HPJ file.
 - Contents - Displays the Contents topic as defined by the Contents

option in the [OPTIONS] section of the .HPJ file.

- Setcontents - Designates a topic as the Contents topic and determines which Contents topic Help should display when a user presses the F1 key.
- Context Popup - Displays a particular Help topic in a popup window. The topic is identified by a Context number that has been defined in the [MAP] section of the .HPJ file.
- Key - Displays the topic found in the keyword list that matches the keyword passed in the Data property if there is one exact match. If there is more than one match, Key displays the Search dialog box with the topics listed in the Go To list box.
- Command - Executes a Help macro.
- Force file - Ensures that the application is displaying the correct Help file. If the correct Help file is currently displayed, there is no action. If an incorrect Help file is displayed, WinHelp opens the correct file.
- Help on Help - Displays the Contents topic of the designated How To Use Help file.
- Quit- Informs the Help application that Help is no longer needed. If no other applications have asked for Help, Windows closes the Help application.
- Help Key - A number or string that identifies the Help screen. Select a command from the Help Command list that can be passed to the Windows Help Engine. The input type for this property varies by the Help command selected:
 - For Context, Setcontents, or Context Popup, enter the context number for the required help topic according to the selected Help command
 - For Contents, Force file, Help on Help, or Quit commands, the help key is not required and this property is ignored.
 - For Command, define a string containing a Help macro that can be


executed.

- For the Key command, define a string containing a keyword for the selected topic.

See the Microsoft Windows Help Application documentation for further information.

HTML Help

eDeveloper supports HTML Help (CHM) files. Select Windows as the Help Type.

From the Help File Name property, click  and browse to the CHM file. The HTML Help Commands are:

- Context - eDeveloper displays Help for a particular topic identified by a context number. The context number is displayed in the Help Key property.
- Contents - eDeveloper displays the Contents topic as defined by the Contents option in the [OPTIONS] section of the HPJ file.

Tooltips

Tooltips provide information about buttons on the toolbar. Tooltips display in a rectangle when the end-user places the cursor over a button.

To create a Tooltip Help, zoom from the Name setting of a Tooltip Help entry in the Help Screen repository to the appropriate Help Properties sheet.

The contents of the Tooltip Text field will be displayed when the end-user places the cursor on a control that the tooltip is attached to.

To attach a tooltip to a control, use the Tooltip setting in the Input tab of the GUI Control Properties dialog. The display will remain on the user's screen for the period of time defined in the Tooltip Timeout setting in *Settings/Environment/Preferences* or until the cursor is moved.

The font used will be the Tooltip Font as defined in the Font repository. No choice of color is provided. The maximum length of a tooltip is 500 characters.

URLs

The URL entry specifies the browser page that will open when F1 is invoked from the browser. Click on the right of the URL property in the URL Help Properties sheet to enter the URL text.

The Magic Authorization System enables developers and certain users to control access to Magic applications in both runtime and toolkit. This control is achieved through setting access keys to various Magic Application elements, and by assigning specific rights to users and user groups.

In this chapter:

• Rights Repository
• Rights Assignment Dialogs
• Magic Access Keys
• Environment Authorization
• Data Security
• User ID Repository
• User Group Repository

Rights assignment is used to control or restrict access to different elements of toolkit or runtime, program execution, menu selection, and other application elements. The Magic Authorization System also provides a means to utilize underlying database security features for restricted data table access and for data encryption. To simplify the assignment of a collection of rights to a class of similar users, the system supervisor can define groups, and can then define rights and assign them at a group level. Group membership for an individual user is optional. One user may belong to several groups.

The requirement, "Jan has the right to create records in the customer file," will be used in the following sections to illustrate basic features of the Magic Authorization System.

Magic Security and People's Roles

The Magic Authorization System consists of a layer of application-level elements, where pieces of an application are declared accessible for various actions such as creation or modification, and a layer of environment-level elements, where roles are assigned to the people using and developing applications. The bridge between a defined role, such as Manager, and an allowable access to an application component, such as "is allowed the customer-deletion right," is made by the process of Rights Assignment, as described below.

The developer should consider both runtime aspects of a security system: data and processing. The data security aspect of Magic's authorization system combines Magic facilities with those of the underlying database; it can affect both the application file itself and the application's data file. The processing aspect in turn has two layers: the application layer and the environment layer.

- The Application Layer - where application-specific security is defined, by the application's developer. In the application layer, the developer can define a bank of rights and assign them to the application's various dictionaries and elements. Any application element that has an access right assigned to it will be blocked to users who have not been assigned this right. Application-layer security can be defined only during application development.
- The Environment Layer - external to any application, this is where users are defined and application rights are associated to users. The environment layer is initially defined by loading a fresh version of Magic, and can be modified by the Supervisor. This layer contains the bank of users, user groups, and other authorization components. Environment-layer security components are accessible to the Supervisor during both runtime and toolkit.

Getting Started as Supervisor

A new Magic environment opens with a default Supervisor defined with a blank password. The first step in accessing the Magic Authorization System is to log on as a Supervisor, using the Logon dialog (*Settings/Logon*), as shown in Figure 13-1.

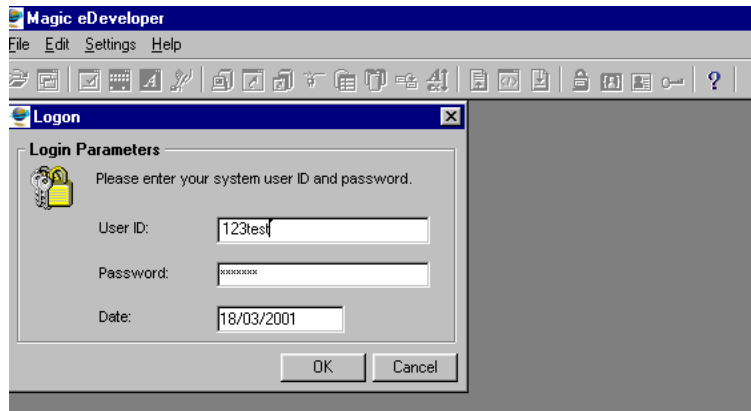


Figure 13-1 The Logon Dialog

Rights Repository

The Rights repository lists the name and keys of all the rights defined for the application. The Rights repository can store 9,999 rights per user.

For our sample requirement, "Jan has the right to create records in the Customer file," you could define a key called XYZ, and name it Create-Customer in an entry in the Rights repository.

Description of the Rights Repository

The Supervisor has exclusive authority to modify the Rights repository, although any user can view it in a limited way. Magic displays only those rights owned by the user who is viewing the repository. Repository entries for rights not assigned to the viewer are filled with asterisks.

To view the Rights repository, select *Workspace/Rights* from an open application.

Right Repository			
#	Name	Folder	Key
1	ENABLE QUERY		ENQY
2	ALLOW DELETE		ALDEL
3	ALLOW MODIFY		ALMOD
4	RUN APG		RAPG
5	VIEW DETAILS		VIEWDET
6	ACCESS PASSWORDS		ACCPAS
7	PRINT RECORDS		PRIREC
8			
9			

Figure 13-2 Rights Repository

The Rights repository has the following properties:

Name

Name is intended to be a description of the right. Create-Customer is an example of a descriptive right name. The name given here will appear as part of the title in Rights Assignment dialogs for this right.

This Name is used in the property of the Rights function.

Key

The Key is the code name used to identify a Right. The example sets XYZ as the key for creating customer records. This key will be used to associate this Right to a user, by placing XYZ in that user's Rights repository.

Public

The valid values for the Public property are:

- Yes - Setting the Public property to Yes means the right is made visible outside of the application, in the Rights repository for the user or group of users. Whenever the Supervisor is working with the User ID or Groups repositories, it is possible to zoom to the Rights repository and see only those rights that are public.

- No - Setting the Public property to No means the right is concealed from the view of the Supervisor when working with the User ID or Groups repositories. A developer may want to conceal rights to protect certain core elements of the application while allowing access to the others.

It is not necessary for the Supervisor to know the Key to a public right, because the right's name will appear as the search argument in a selection list that returns the right's Key. However, the Supervisor can assign a non-public right to users or groups only by directly typing in the Key - the selection list will not, of course, include the names of non-public rights.

In general, the non-public classification is intended for assignment to those rights that are relevant only for developers, and not for end-users. The original developer of an application may assign certain non-public rights to another developer, for modifying that application. This feature allows a software house that has developed an application to sell or license the application to a programmer, without giving the programmer unrestricted access to the source code. The programmer can be authorized to use the software for developing custom applications, while subject to whatever access restrictions the software house considers appropriate.

Because Public rights appear in a rights selection list that is external to the application, and because the rights selection list returns a right's Key, it is important that access to Public rights be protected. This protection is accomplished when a developer defines the Public Rights Access key in the Application Properties dialog; if a Public Rights Access key is specified, Magic requires that this key also be specified in the supervisor's Rights repository. The supervisor cannot learn the Public Rights Access key to an application from Magic.

The Rights function allows the developer to query whether a user has a particular right. To facilitate use of this function, the Expression Rules repository includes a button, labeled Rights, that allows access to the Rights repository.

Folder

Displays the name of the folder in which the right entry is stored. You can create a folder by highlighting the Rights icon on the Navigator pane and

clicking **F4**. Folders let you group rights entries. For more information about Folders, see Chapter 1, Introduction.

Public Name

Defines the public name of the right entry in which it will be called by an Internet requester or a Call Remote operation. The public name must be unique within the application file.

Rights Assignment Dialogs

The Rights Assignment dialogs are used to identify which activities within various application elements are to be permitted. These activities may be relevant to both toolkit and runtime modes. When the developer attempts to perform an activity (for example to create a new line in the Table repository, or to invoke a program), Magic will execute the attempted activity only if the developer owns the right that corresponds to that activity. Similarly, when an end-user attempts to add a new customer to the Customer file, Magic will add the customer only if the end-user owns the right to add new customers.

The Rights Assignment dialogs are accessed by selecting *Options/Authorize* from anywhere in the Model, Table, Program, Help, or Menu repositories, or through the Application Properties dialog.

Only the Supervisor and the application's Super Key holders (explained below) have unrestricted access to Rights Assignment dialogs. For each activity appearing in a Rights Assignment dialog, these people can type in the number of a right or zoom into the Rights repository to select the number and name of the right whose holders are to be authorized to perform the given activity. Once an activity has been authorized, all of the right's holders will see the number and name of its corresponding right.

It is possible to associate one right with more than one Magic activity.

An activity that has not been associated with any right is unprotected. Because access to the activity is unrestricted, it can be invoked by any Magic user.

Any user or developer who has a particular right displayed in one of the Rights Assignment dialogs can view the relevant dialog line, can change the entry to a different right that he or she holds, and can even remove the right.

For example, Jan, as the holder of right number 4, is authorized to use the Automatic Program Generator for a given file. Another user who holds this right as well as right number 6 can change the APG property in the Rights Assignment dialog accessed in the Table repository from 4 to 6, meaning Jan can no longer use APG.

Someone who does not have a particular right, however, cannot modify or even view the corresponding Rights Assignment dialog property. A user or developer who does not have right number 6 will see an asterisk in the rights number entry for the APG property and a line of asterisks where its description would normally appear.

Model Repository Rights Assignment

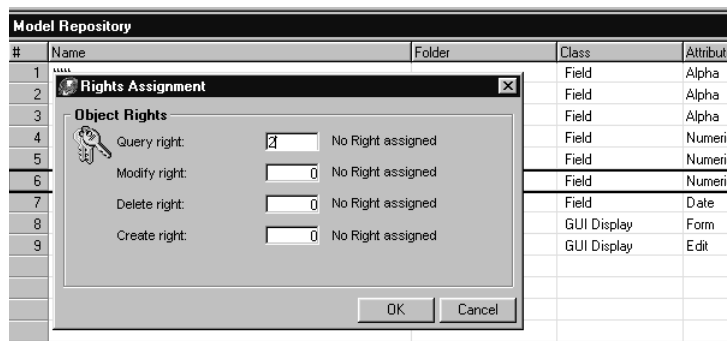


Figure 13-3 Assigning Model Repository Rights

When you access the Rights Assignment dialog from the Model repository, the following property meanings will be applied to its entries:

Query

The Query property is assigned a rights number that determines the level of authorization for the user to view models. Updates will not be prevented, but

they will not be saved. Note that without Query rights, the user cannot open or view any program, help screen, menu, model, table, or object.

Modify

The Modify property is assigned a rights number that determines the level of authorization for the user to update models. This property also controls the ability to export the model definitions.

Delete

The Delete property is assigned a rights number that determines the level of authorization for the user to delete models. Note that for the users to delete models, they must also have the Modify rights.

Create

The Create property is assigned a rights number that determines the level of authorization for the user to create new models for the application. Note that for users to create models, they must also have the Modify right.

Table Repository Rights Assignment

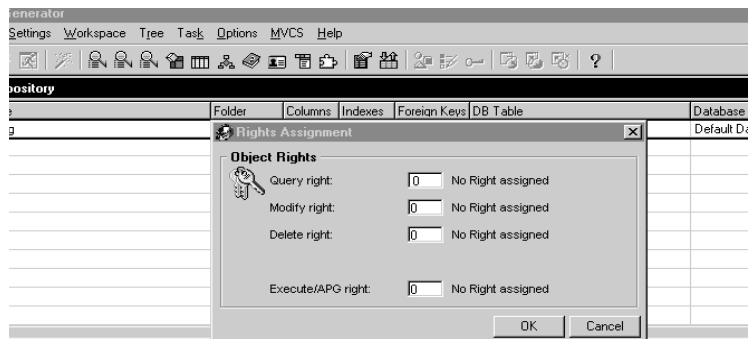


Figure 13-4 Defining Table Repository Rights

When you access a Rights Assignment dialog from the top of the Table repository when no entry is highlighted, you can assign Create, Query, Modify, Delete, and APG rights. These rights effect all entries in the Table repository.

When you access the Rights Assignment dialog from a specific entry in the Table repository, the following property meanings will be applied to that repository:

- Query
- Modify
- Delete
- Create
- Execute/APG - The Execute/APG property holds the number of the right to which the user is authorized to use the Automatic Program Generator facility to generate programs from the entry.

Query, Modify, Delete, and Create properties are explained in the Model Repository Rights Assignment section on page page 1040.

Program Repository Rights Assignment

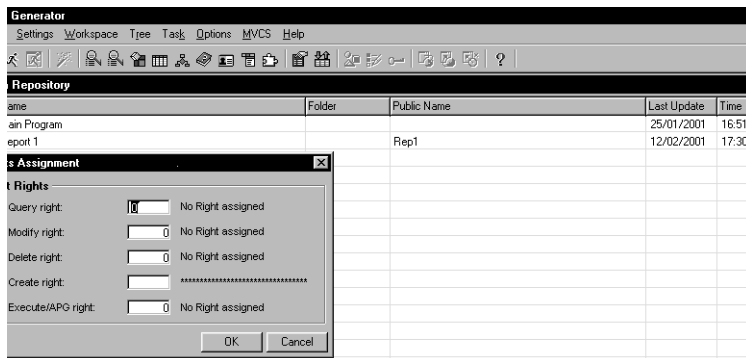


Figure 13-5 Defining Program Repository Rights

When you access a Rights Assignment dialog from the top of the Program repository when no entry is highlighted, you can assign Create, Query, Modify, Delete, and Execute rights. These rights effect all entries in the Program repository.

When you access a Rights Assignment dialog from a specific entry in the Program repository, the following property meanings will be applied to that program:

- Query
- Modify
- Delete

For more information on the above properties, refer to the Model Repository Rights Assignment section on page 1040.

- Execute/APG

The Execute/APG property holds the number of the right to which the user is authorized to execute the program at runtime.

While the Execute property allows only Go/No-Go assignment, a finer-grained security scheme for execution can be implemented by using the functions `Rights()` and `USER()` within the program (these functions are detailed in the chapter). For example, insertion point parking and even user input can be restricted on the basis of user rights. Moreover, by using an expression for the visible property of a control, it is possible to make controls invisible to particular users on the basis of their User IDs and authorization privileges. A conditional Verify Exp operation can similarly restrict certain activities, such as entering values for a control outside a given range.

Note: the Execute right is relevant only in Runtime mode, while the other Program repository rights are relevant only in Toolkit mode.

- Create

The Create property is accessible only from above the first line of the Program repository, when no entry is highlighted. The Create property holds the number of the right whose owners are authorized to create new programs for the application.

Help Screens Repository Rights Assignment

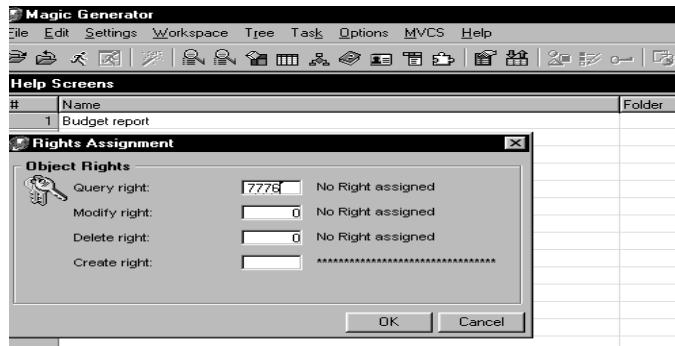


Figure 13-6 Assigning Help Screen Rights

When you access the Rights Assignment dialog from the Help Screens repository, the following property meanings apply:

- Query
- Modify
- Delete
- Create

For more information on these properties, refer to Model Repository Rights Assignment section on page 1040.

Menu Definition Rights Assignment

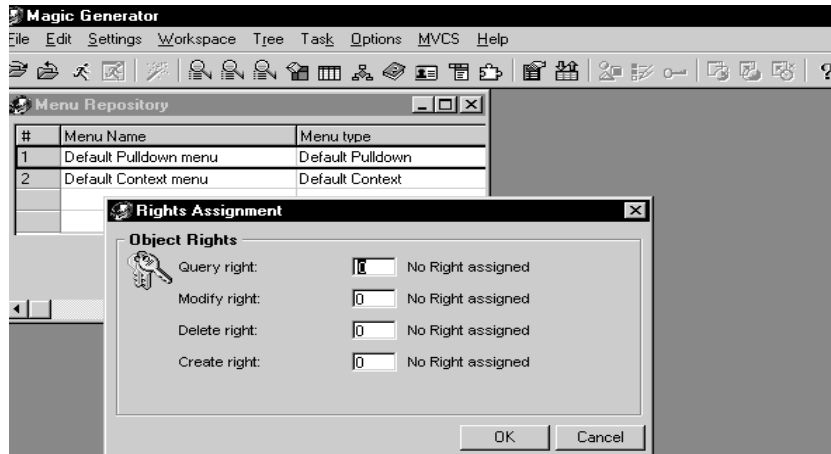


Figure 13-7 Assigning Menu Definition Rights

When you access a Rights Assignment dialog from either the Context or Pull-down Menu Definition repository, the following property meanings will apply:

- Query
- Modify
- Delete
- Create

For more information on these properties, refer to Model Repository Rights Assignment section on page 1040.

Runtime Menu Access Rights

The preceding rights are used to control developer access to menu definitions. In addition, the runtime behavior of each menu item can be controlled by assigning an access right to that item. If at runtime a user does not have the access right, then the menu item, and all of its descendent menus, will not appear.

To assign an access right to a menu item, highlight the desired menu item in the Menu Definition repository, and select *Edit/Properties*. Then, in the menu's Properties dialog, assign the number of the right to the Access Right property.

Component Repository Rights Assignment

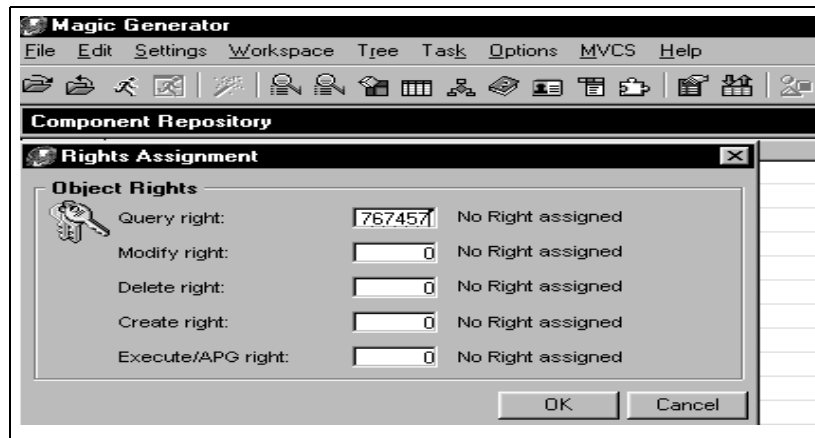


Figure 13-8 Assigning Component Rights

When you access the Rights Assignment dialog from the Component repository, the following property meanings apply:

- Query
- Modify
- Delete

For more information on the properties above, refer to Model Repository Rights Assignment section on page page 1040.

- Create - This property is only accessible from the first line of the component repository, when no entry is highlighted. The Create property holds the number of the right whose owners are authorized to create new components in the application.
- Execute/APG - This property holds the number of the right to which the user is authorized to execute the component items at runtime.

Application Properties Dialog Rights Assignment

The following Access keys are displayed when opening the Application Properties dialog and by selecting *Options/Authorize*.

- Query

The Query property holds the number of the right whose owners are authorized to examine, but not modify, the Application Properties dialog.

- Modify

The Modify property holds the number of the right whose owners are authorized to modify the Application Properties dialog. Without owning this right, a user who wants to export or import a given Magic element, such as the Model repository, will not be authorized to export or import the Rights Holders list associated with the element.

- Flow Monitor Right

You can restrict Flow Monitor access so that only certain users can remotely monitor the application. Only users allocated with this right can remotely monitor the application.

The following Access keys are displayed in the Security tab of the Applications Properties dialog:

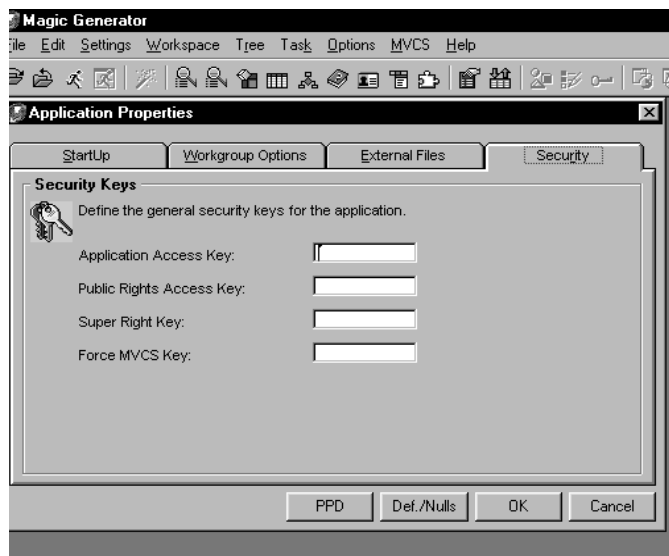


Figure 13-9 Security Tab

Application Access Key

When an Application Access key has been defined, only holders of the key can open the particular application for development or runtime.

Once the Application Access Key property has been entered and the Application Access key assigned, someone who is not an Application Access key holder who accesses *File/Application Properties* will not see the Application Access key property in the dialog.

Public Rights Access Key

By creating a Public Rights Access key, a developer can improve security for a given application. The supervisor's ownership of this key serves to hamper any attempts of an unauthorized user to remove system password protection by

destroying the Security File (usr_std), which contains all user passwords, and by then logging in as the supervisor. An unauthorized user lacking this key, will not be able to find out which rights are associated with which keys in a particular application. After zooming from the Rights property of the User ID repository, a bogus supervisor will not be able to see the Rights Name property, which describes the rights associated with given keys, nor be able to zoom from the Key property into the full Rights repository.

The Super Right Key

The supervisor can create a Super Right key for an application, to give its holder all of the rights to all of the application's activities, for development and runtime purposes. The holder of the Super Right key does not have to hold separate rights to individual application activities.

Once the Super Right property has been entered and the Super Right Key assigned, someone who is not a Super Right Key holder who accesses *File/Application Properties* will not even see the Super Right Key property in the dialog.

Force MVCS Key

The Force MVCS Key property appears only when the Application Properties dialog is accessed by the owner of the Force MVCS Key. This property holds the name of the Force MVCS Key.

The supervisor can create a Force MVCS Key for an application to give its holders the rights of limiting access to one developer in the application.

For a more detailed explanation refer to Chapter 2, Settings.

Restricting Import and Export

Import and Export of Magic Application Elements

To prevent unauthorized users from exporting Magic application elements, and then importing them to a different application, the right to import and export Magic application elements is associated implicitly with the rights to modify and create those same elements.

If a user has the right to modify a repository (Model repository, Table repository, Program repository, Menu Definition, Help Screen, Components, Application Properties) then that user automatically has the right to export the object. If a user has the right to create an entry in a repository, then that user has the right to import the repository.

Importing and Exporting Rights

The global action rights for a particular Magic element, such as the Create right for the Program repository, are exported or imported as part of the application data section. However, only if the user is authorized to modify the Application Properties dialog can these rights be exported or imported.

Environment Authorization

The environment layer of Magic's Authorization System contains all of the Authorization repositories that are not application-specific. Information in these repositories applies to all applications within one Magic installation environment. The repositories at the Environment level of the Magic Authorization System are:

- User ID repository
- User Group repository
- Secret Name repository
- Logon dialog

User ID Repository

The User ID repository is where each individual user is assigned a password, and, optionally, group membership. The supervisor, who has exclusive access to the User ID repository, can edit the repository by selecting *Settings/User IDs*. The supervisor can add or delete User IDs anytime, during development or runtime. The User ID repository can store 9,999 user passwords and group associations.

The user supervisor, however, cannot see the passwords of all users. Passwords are displayed as a string of asterisks (*) if not empty. The user supervisor can alter passwords of users, but cannot see existing passwords. To change a password, it is necessary to zoom into a password box, and to enter the password twice. The password is not displayed while it is being entered. A non-supervisor user can enter the User ID repository and modify his or her own password. Changing a password is the only change to the User ID repository allowed to a non-supervisor user.

Description of the User ID Repository

User ID

An alphanumeric string that will be compared at Logon time to the string the user enters the User ID property of the Logon dialog. If the Input Password field in the Environment dialog is set to No, the Logon window will not be automatically displayed. The User ID property can be queried in programs by the USER (0) function.

Name

The name of the user. This property is not checked during the logon procedure. It identifies the user to the supervisor, and it can be queried in programs by the USER (1) function.

Password

An alphanumeric string that will be compared at Logon time to the string the user enters the Password property of the Logon dialog. If the Input Password

property in the Environment dialog is set to No, the Logon window will not be automatically displayed.

Rights

The Rights column displays a count of the rights keys allocated to the user, and provides a zoom point to the user's Rights repository. Because a Rights key can correspond to more than one right, the count shown may be less than the user's total number of individual rights.

The supervisor can zoom from the Rights property to the user's Rights repository to select additional rights for assignment to the user at anytime during development and runtime. This process is described in the next section, Filling in the User's Rights Repository.

A user need not have any individual rights assigned.

Groups

The Groups property displays a count of groups to which the user belongs. To define membership in a group, the supervisor zooms from the Groups property into the User Groups repository to select a group for the user. A user can belong to more than one group. The User Groups repository is described below.

When the User ID repository is displayed, the supervisor can enter additional free text about the user in a memo property that is accessed by pressing the Properties button located on the lower right-hand side of the screen. Information entered in this memo property can be queried in programs by the USER (2) function.

When the user logs on, Magic compares the User ID and the Password strings of the Logon screen with those of the User ID repository. Even though a given user has a unique User ID and Password at the environment level, each application can apply a different security scheme to that user. Provided there is at least one entry in the User ID repository, whenever a user tries to access data or programs, Magic performs the security check. If the user does not have rights to perform the attempted activity, a message to that effect is displayed on the message line.

The Rights Assignment dialogs control access to the application itself during development, while using the toolkit (development). For programs, end-user runtime access can also be controlled. For finer control of runtime security within programs, you can use the Rights() and USER() functions. By using these functions you can create a custom security system that includes controlled access to elements within the task, while also controlling access at the program level. Refer to the Rights and USER functions in the chapter for more information.

User Group Repository

The User Group repository lists the name and rights of each user group.

Description of the User Group Repository

The repository has the following properties:

Name

Name is a description of a user group.

Rights

The Rights property displays a count of the rights keys allocated to the group, and provides a zoom point to the group's Right repository. Because a Rights key can correspond to more than one right, the count shown may be less than the group's total number of rights.

The supervisor can zoom from the Rights property to open the group's Rights list and examine the rights owned by a particular group.

The set of keys owned by a given group is constant, regardless of the application. The rights associated with these keys, however, can vary from application to application. The list of rights Magic presents for view, when the supervisor first zooms from the User Groups repository Rights property, is the group's set of rights pertaining to the first application that appears in the Application list. The application's name appears at the top of the Rights of

repository. The supervisor can view the group's rights in a different application by clicking on the Application button that appears in the lower right-hand corner of the screen when the Rights of repository is displayed.

The Supervisor Group

The supervisor can assign or add users to the Supervisor Group. All users in the Supervisor Group will automatically inherit all of the rights of the supervisor.

The Secret Name Repository

Secret names are intended for use where there is a need to hide information pertaining to the Authorization System's implementation from unauthorized users. For example, Secret Names should be used for Application File Access keys and Data table Access keys.

The Secret Name repository is identical to the Logical Name repository, except:

1. Access to the Secret Name repository is allowed only to the supervisor.
2. Secret Names are stored in the security file that is specified as an Environment property. This security file is encrypted.

A Secret Name can be used wherever a Logical Name can be used, using the same syntax. Refer to Chapter 2, Settings for an explanation of Logical Names. Magic will attempt to resolve a logical name through the Secret Name repository first, before looking for it in the Logical Name repository.

Data Security

Restricting Access to Application Data Tables

Magic provides the means to restrict access to data tables by an access key. Magic also allows encryption of the data table using the Access key as an

encryption seed. SQL databases do not provide this feature. These features are available only when the underlying database provides it. To implement data table security for an application data table, select *Workspace/Tables*, highlight the Table entry in the Table repository, and select *Edit/Properties*. Two properties in the Table Properties dialog control file security: Access key and Encrypt File.

Access Key

The Access key specified in this property will be required for every access to the data table, by any application, either Magic or external. If the application does not provide the access key specified in this property, access to the data will be barred. For Magic applications, an appropriate message will be displayed.

Note: Always make a table's Access key a Secret Name. Using a Secret Name makes access to the table installation-specific, and the Access Key translation visible only to the system's supervisor. A different Secret Name translation can be used for different installations.

Encrypt File

If the Encrypt File property is set to Yes, the file will be encrypted, using the Access key as the encryption seed. Encryption may affect system performance. When a file is encrypted, it cannot be deciphered by an external access that is not aware of both the encryption method and the encryption seed.

Warning: If encryption or Access keys are used, the user must be sure to keep the Access key in a safe place. There is no way to access the file without the Access key.

Restricting Access to the Application File Itself

The Access key specified in this property will be required for every write access to the application file, by any user, either Magic or external. If the application does not provide the Access key specified in this property, write access to the application will be barred. If an access key is specified, Magic will instruct the

underlying database that holds the application file to encrypt the file using the Access key as encryption seed. The application file will remain accessible for read-only purposes by any Magic system. External applications trying to access the Magic application file will fail. Application file encryption will prevent analysis by external programs. There is no need to supply an application's access key when deploying the application. The application's access key must always be specified when further development is required.

Whenever an access key is added or removed, Magic will prompt for confirmation of Encryption or Removal or File access restrictions. The prompt appears when leaving the Applications Repository editing session. Whenever you leave an Applications Repository editing session, Magic scans all the application entries for changes in access key values. Application file encryption may require some processing time to complete.

A final reminder: be sure to specify the Application Access key as a Secret Name. An application's access key is stored in the MACIG.INI file, which is an unsecured file, and using a Secret Name hides the value of the Access key.

HTTP Authentication

A string that represents an HTTP address that lets you activate and post information. When an eDeveloper client accesses a web server that requires a user name and password, the URL can be defined as

`HTTP://User:Password@[URL]`.

You can use secret names, for example:

`HTTP://%user_secretname%:%pass_secretname%@[URL]`

HTTP Authentication can be used for the HTTPGet function, HTTPPost function, and WSDL Assist.

You can use eDeveloper to define objects for a variety of different component types. The interoperability and reusability of distributed objects lets developers build systems in different programming environments by sharing the same components.

In this chapter:

• Component Frameworks
• Component Repository
• Component Runtime Behavior
• Components and the Main Program
• Web Service Interface Builder
• Enterprise JavaBean Interface Builder

Component Frameworks

You can define application objects as components. Exporting components lets you share resources among eDeveloper applications and also facilitates the distribution of application revisions as shown below.

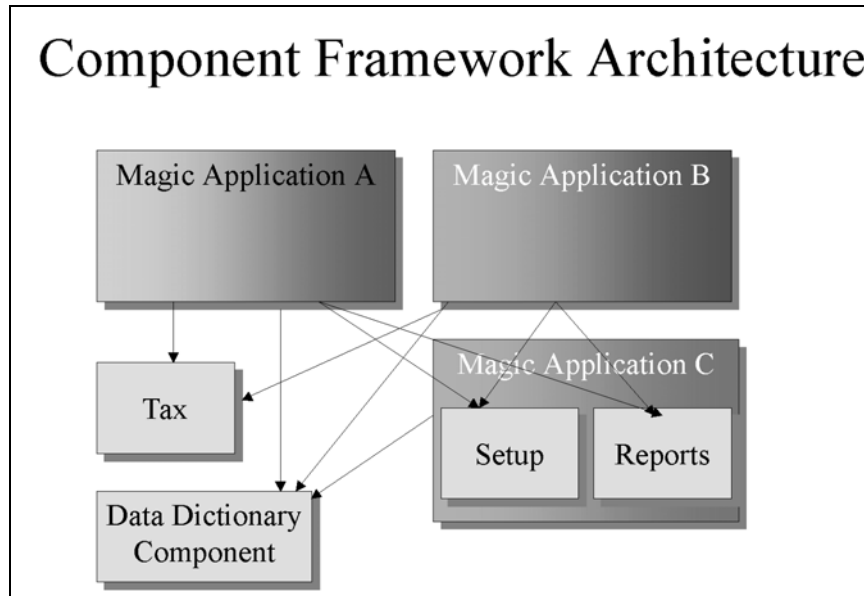


Figure 14-1 The Component Framework Architecture

Components are listed in the **Component** repository. The interface displays the application objects that can be shared with other applications. The objects that can be shared are:

- Models
- Tables
- Programs
- Helps
- Rights

- Events
- Environment settings

The loaded component is integrated into another application and appears in the **Component** repository. The component objects can be selected from their respective repositories.

Component Repository

The **Component** repository lists the objects defined as components in an eDeveloper application. This repository has the following columns:

- Entry # - The row number
- Name - The component name, as listed in the **eDeveloper Component Interface** file (MCI)
- Description - Additional component information provided in the MCI file
- Folder - The folder where the component entry is stored

Component Repository		
#	Name	Description
1	db_tax	DB_Tax
2	db_pricing	DB_Pricing
3	db_admin	DB_Admin

Figure 14-2 Component Repository

From the **Component** repository, you can:

- Reload a component interface or load a new component by selecting **Load/Reload** in the **Options** menu
- Delete a component by pressing **F3**

- Display the properties for a selected component by pressing **CTRL+P**
- Show the component interface by zooming from the Name column
- Assign rights to a component

Loading a New Component

When you zoom on an empty component row, eDeveloper opens a dialog box that lets you select the component type. Once the component type is selected, you must select the **Magic Component Interface (MCI)** file using the **Load/Reload** option in the **Options** menu. eDeveloper loads the component interface and corresponding application file.

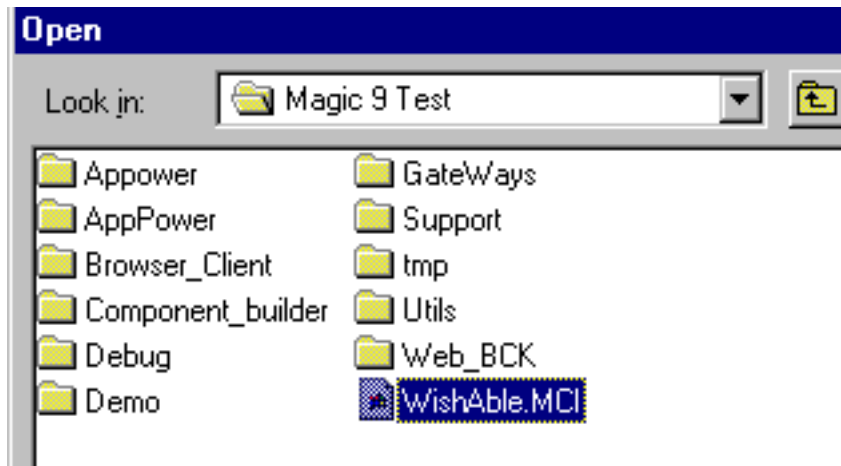


Figure 14-3 Component Interface File Open Dialog Box

Deleting a Component

When you delete a component, the component interface is removed from the application file. but the references to the component in the application are not removed. If you try to access a deleted component, eDeveloper displays an unknown reference indicator, two question marks (??).

Magic Component Properties

Press **CTRL+P** to open the **Component Properties** dialog box from the **Component** repository. The Component properties are displayed in two categories, **General** and **Settings**.

The **General** properties let you enter information about storing, loading and accessing the MCI file, as shown below in Figure 14-4.

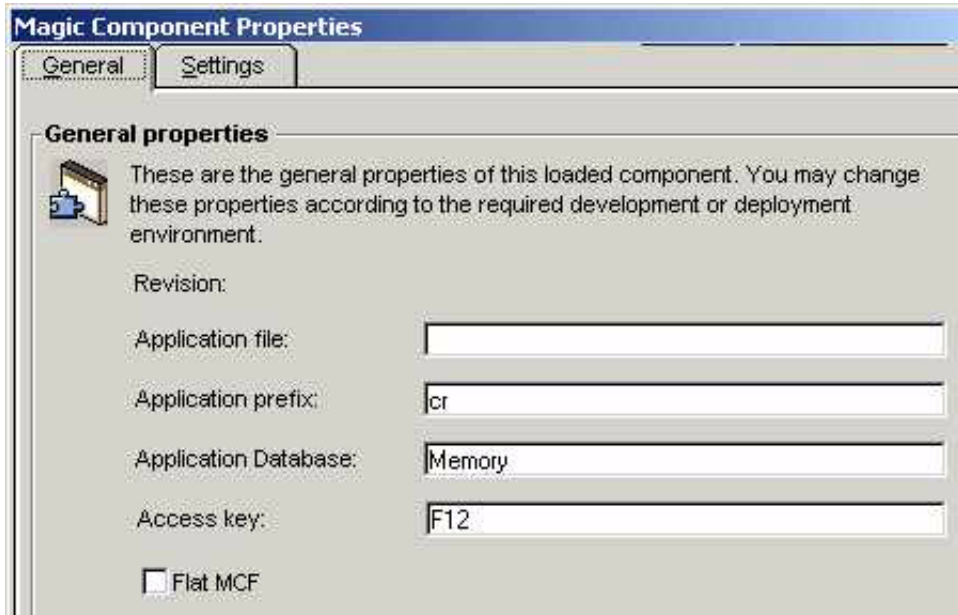


Figure 14-4 Component Properties - General

These properties are:

- Revision - The revision number assigned to a loaded component. This property cannot be modified.
- Application File - The path name used for the source application. The default location is where the **Magic Component Interface** (MCI) file is stored. The application file name is written in the MCI file. The application file value can be modified.
- Application Prefix - The two-letter code used as the application identifier.

- Application Database -The database where the component data is stored. The value is set from the Magic Component Interface Builder, but can be modified.
- Access Key - You can enter the component's access key. The component's access key must match the application's access key.
- Flat MCF - You can determine if the application file is saved as a Magic Flat File.
- Load Immediate - You can select this check box to load the components together with the application load. The default value is read from the MCI file and can be modified.

The **Setting** properties let you define the help information for the MCI file, as shown in Figure 14-5.



Figure 14-5 Component Properties - Settings

These properties are:

- **Help File** - You can enter the path name to the help file. The default location is in the same directory as the MCI file. The help file name is written in the MCI file.
- **Help Key** - You can define keywords to access the help file for a component entry.

Objects Connected to the Magic Component Interface

To access the **Component Items** repository, zoom (**F5**) from the **Name** column in the **Component** repository.

The component interface is based on the objects that are included in the component and is organized by tabs. Each tab represents a component object type, such as Models, Tables, Programs, Helps, Rights, and Events. The individual component objects are listed under the required component object type.

Component Runtime Behavior

The environment settings selected for a component are accessed for any component object that the end user opens in the application. The environment settings for a component appear in the Environment screen, which is accessed by pressing the **Environment** button in the **Component Items** repository.

All environment variables not specified for a component are inherited from the environment settings of the application. The environment variables that can be associated with a component are:

- Century Start
- Batch Event Interval
- Allow Create in Modify Mode
- Allow Update in Query Mode

- ISAM Transactions
- Keyboard Idle Seconds
- Center Screen in Online
- Reposition after Modify
- Date Mode

The following property settings add new lines to the Magic.ini file when the component is loaded to the new application:

- Database properties
- DBMS properties
- Server properties
- Services properties
- Logical Names

Comments appear before these additional lines to identify the properties that are associated with the component.

After the component is loaded to the new application, the properties associated with the component are available to all application objects.

Deleting a component does not remove the component properties that are listed in the Magic.ini file.

INIPut Function Behavior

The **INIPut** function behavior can differ depending on the context in which the end user accesses the component. If the end user accesses a task from a component and uses the **INIPut** function, the following behavior can occur:

- If the *force write* parameter is True, the Magic.ini file is updated and the change affects all active contexts.
- If the *force write* parameter is False and an environment setting is associated with a task from the component, the change affects all the

programs in the current context only. If the end user enters the component task, the environment variable must receive the environment value again from the settings associated with the component.

- If the *force write* parameter is False and the environment variable was not specified in the component, the environment variable is updated only for the current task.

Components and the Main Program

The Main Program cannot be exported as part of a component because this program cannot be executed by a Call operation.

When an object from another component is used (a program, a table, or an event) it must first be preceded by the execution of the Main Program of that object's application. This means that the first time an object from another component is used, the Task Prefix of its Main Program is executed and its variables are initialized.

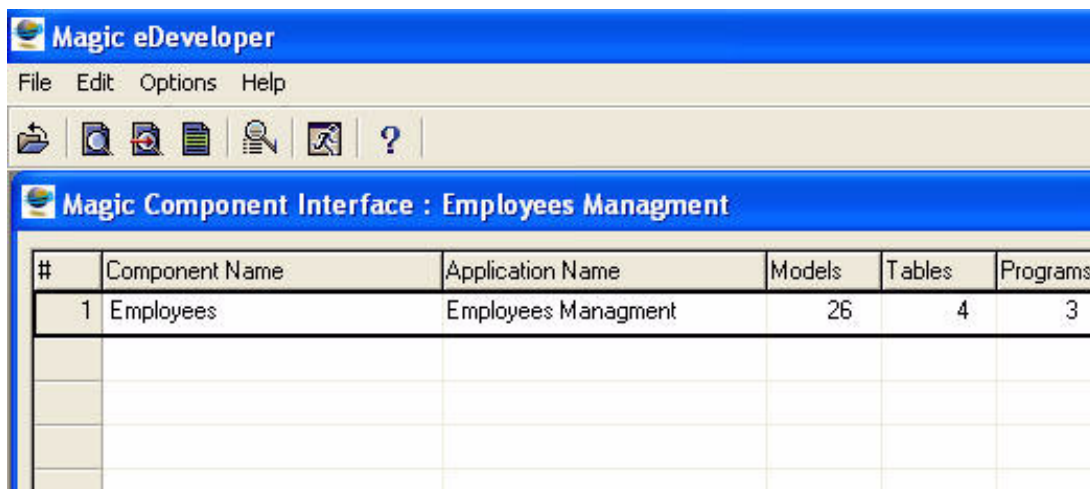
An exception to this rule is for components that are used immediately, which means that the component is loaded when the application is opened, and not when the component is actually used. For these components, the Main Program of the component is executed as soon as they are loaded.

The Task Execution Stack should not be affected by the component's Main Program execution. The program of the component is called by a parent program, and not its Main Program. The Component's Main Program is inserted in the Task Execution Stack just after the application's Main Program and before all other programs.

When an application is terminated before the Task Suffix of the Main Program is executed, the Task Suffix is executed in the order opposite to the Task Prefix execution.

Component Interface Builder Repository

The **Component Interface Builder** repository lets you build an **MCI** file. Click the **Magic Interface** command on the **Components** menu to open the **Component Interface Builder** repository, as shown in Figure 14-6.



#	Component Name	Application Name	Models	Tables	Programs
1	Employees	Employees Managment	26	4	3

Figure 14-6 Magic Component Builder Repository

To create a Magic component, fill in the columns as explained below:

- Component Name - The component name is mandatory and cannot be longer than 30 characters.
- Application Name - eDeveloper automatically displays the name of the current application. The application name cannot be changed.

The item types that can be assigned in an MCI file are:

- Models
- Tables
- Programs
- Helps
- Rights

- Events
- Environment

Zoom from an item column to assign a specified item type. Each item column automatically displays the number of assigned items for that item type.

Magic Component Builder Properties

You can specify the following properties:

- Description - You can enter the MCI file description
- Revision - You can enter the MCI file revision number
- MFF - You can specify that the component is a Magic Flat File application. Note that when this application type is specified, the Application Database is disabled.
- Application File Name - You can specify the current application file.
- Application Database - You can specify the database of the MCF file.
- Help File Name - You can specify The help file path and name.
- Help Key - You can specify the key combination that opens the component help file.

Item Type Repository

The **Item Type** repository displays the items assigned to the selected type. The columns of this repository are described below:

- Name - eDeveloper automatically sets the specified item according to the name selected from the list of items. Click the **Add Items** button to display the items that can be selected. Items are only displayed for programs that have a specified public name. The item name cannot be changed.
- Public Name - eDeveloper Automatically displays the public name of the selected program.

- Remarks - You can add reference information about the item.
- Help Key - You can enter a number that can be set to open a help page from the help file specified in the Component properties. The help key cannot be more than 30 characters.

Environment Repository

In the **Environment** repository, choose from the following categories to see the list of items you can add to an **MCI** file:

- Environment Settings
- Servers
- Services
- Databases
- Logical Names

Click the **Add Items** button to select the environment options for the specified category.

Adding an Item

To add an item:

1. From the **Magic Component Builder** repository, specify the component name.
2. Zoom from the required item columns (for example, Models, Tables, Programs) to enter the **Item Type** repository.
3. Click the **Add Items** button to open that **Item** list box.
4. From the **Item** list box, select the required item. The item appears in the **Item Type** repository. The number of items is updated in the item column of the selected type.

Note: Items are displayed on the selection list only when the Magic item has a public name. The public name creates a connection

between the called item and the host application. In the case of duplicate public names for the same item, the engine accesses the first name encountered.

Generating the Magic Component Interface File

After specifying your MCI items and properties, click the **Build Interface File** option from the **Options** menu or press **CTRL+I**. The **Generate MCI File** dialog box appears with the file name, as shown in Figure 14-7.

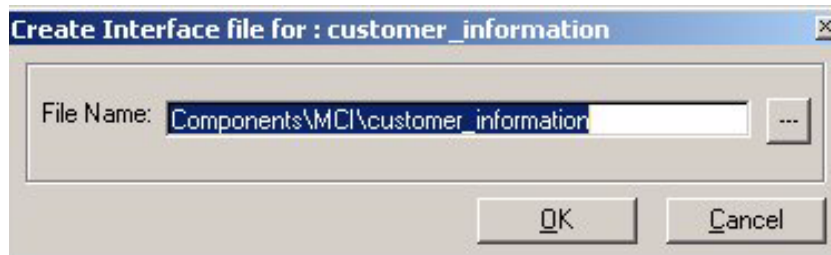


Figure 14-7 Generating an MCI File

If the directory for the MCI file does not exist, the Magic Component Builder creates it.

If the MCI file name already exists in the defined path, eDeveloper displays the following message:

The Component Interface already exists. Overwrite/Cancel

Click **Yes** to overwrite the existing file. Click **No** to define a new name for the **MCI** file.

Sample MCI File

This section displays an example of the keyword parameters in an **MCI** file.

```
//
```

```
COMPID = 76050384676477
```

```
NAME = "Comp1"
```

```
DESC = "Description of Component file"
IMMED = Y
APPLFILE = "d:\magic\newmci\newmci.mcf"
APPLDB = "Default Database"
FLATCTL = N
VRSN = REV5
HLPFILE = "d:\magic\newmci\mgcomp1.hlp"
APPREFIX = mc
MODEL=PUBLIC="MODEL1" RMK=" "HELPKEY="HLPK1"
MODEL=PUBLIC="MODEL5" RMK=" "HELPKEY="HLPK5"
File=PUBLIC="File101" RMK=" "HELPKEY="HLPK101"
File=PUBLIC="File104" RMK=" "HELPKEY="HLPK104"
EVNT=PUBLIC="EVNT502" RMK="gdfxgcvgvfdgdf" HELPKEY=HLPK502
//
```

Web Service Interface Builder

Web Services Description Language (WSDL) is an XML-based language used to describe published web services and their parameters.

Click the **Web Service Interface** command from the **Components** menu to create a WSDL file. The **Web Service** repository opens, as shown in Figure 14-8.

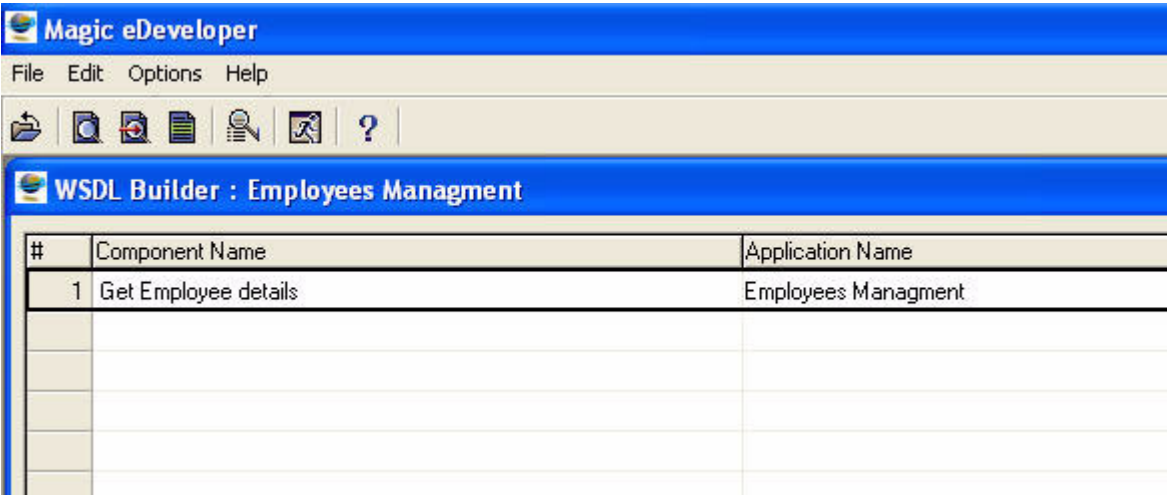


Figure 14-8 Web Service Builder

The Web Service Builder columns are described below:

- Component Name - The component name is mandatory and cannot be longer than 30 characters.
- Application Name - eDeveloper automatically displays the name of the current application. The application name cannot be changed.
- Programs - Automatically displays the number of programs assigned to the WSDL file.

Web Service Programs Repository

The **Web Service Programs** columns are described below:

- Program Name - Automatically sets the specified program by the program name selected. Click the **Select Programs** button to display the list of programs. A program must be a batch task type and have a public name. The program name cannot be changed.
- Public Name - eDeveloper automatically displays the public name of the selected program.
- Arguments and Returned Values - eDeveloper Displays the total number of arguments. Zoom from this column to modify an argument name, XSD type, and direction.

You can expose the Web service in a document style by setting only one Alpha or BLOB parameter for the program, as shown in Figure 14-9.

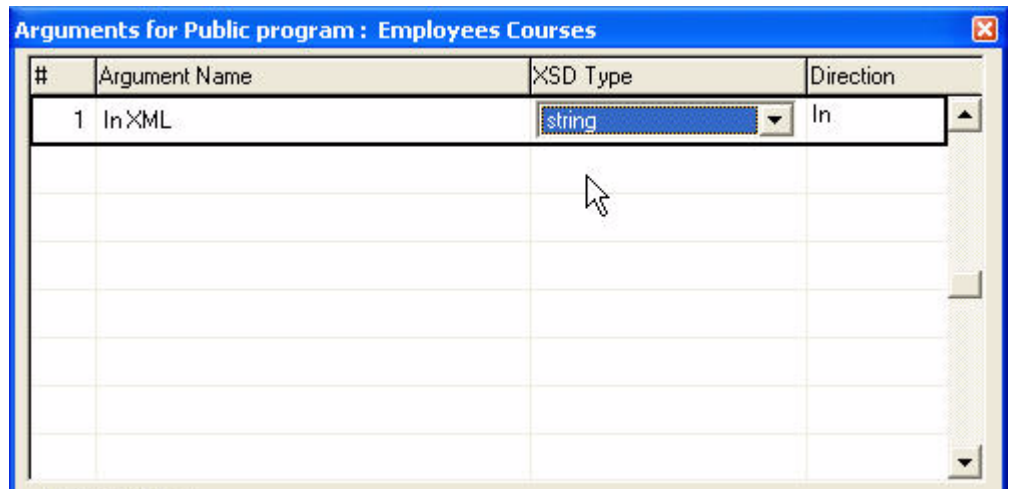
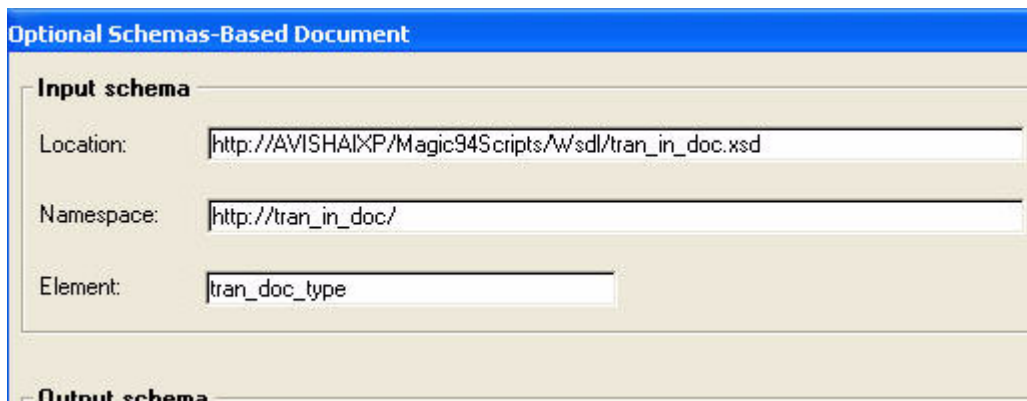



Figure 14-9 Arguments for Public Programs

From the **Web Service Arguments** dialog box, click **Schemas** to specify the location of the XSD file, as shown in Figure 14-10.



The screenshot shows a dialog box titled "Optional Schemas-Based Document". It has two main sections: "Input schema" and "Output schema". The "Input schema" section contains three text input fields. The first field is labeled "Location:" and contains the text "http://AVISHAI\XP\Magic94Scripts\Wsdll\tran_in_doc.xsd". The second field is labeled "Namespace:" and contains the text "http://tran_in_doc/". The third field is labeled "Element:" and contains the text "tran_doc_type". The "Output schema" section is partially visible below the "Input schema" section.

Figure 14-10 Input and Output Schemas

Specify the location and XSD file, and click the  button. The Namespace and Element parameters are filled out automatically.

- Remarks - You can add reference information about the program.

Click the **Verify Structure** button to start the checker. If there are no errors in the program parameters or structure, eDeveloper confirms that the structure is okay. If errors are found, eDeveloper displays a warning.

WSDL File Settings

The WSDL file settings are:

- Description - You can enter the WSDL file description.
- Revision - You can enter the WSDL file revision number.
- WSDL File - You can specify the WSDL file path and name.
- Service End Point - You can specify the URL address where the Web service is deployed.
- WSDL Name Space - You can specify the WSDL Name Space identifier.

- Header Information - You can select the check box when you want to receive the user name and password in the SOAP header.

Generating a WSDL File

After specifying the WSDL file programs and settings, click the **Build WSDL File** option from the **Options** menu or press **CTRL+I**. The **Generate WSDL File** dialog box appears with the file name.

If the directory for the WSDL file does not exist, the WSDL Interface Builder will create it.

If the same WSDL file name already exists in the defined path, eDeveloper displays the following message:

The WSDL already exists. Overwrite/Cancel

Click **Yes** to overwrite the existing file. Click **No** to define a new name for the WSDL file.

The Created WSDL File

eDeveloper creates the WSDL file with all its method definitions and automatically adds a variable for the SOAP header, user name, and password. When a calling request contains a user and password, eDeveloper will log in the user and obtain the user's rights. This feature is not mandatory and is used to restrict access to the exposed methods of the WSDL file.

Enterprise JavaBean Interface Builder

The EJB Interface Builder enables an eDeveloper application to be exposed as an EJB (Enterprise JavaBean) in a J2EE enterprise server.

Click the **EJB Interface** command from the **Components** menu to create a Jar file. The **EJB** repository opens, as shown in Figure 14-11.

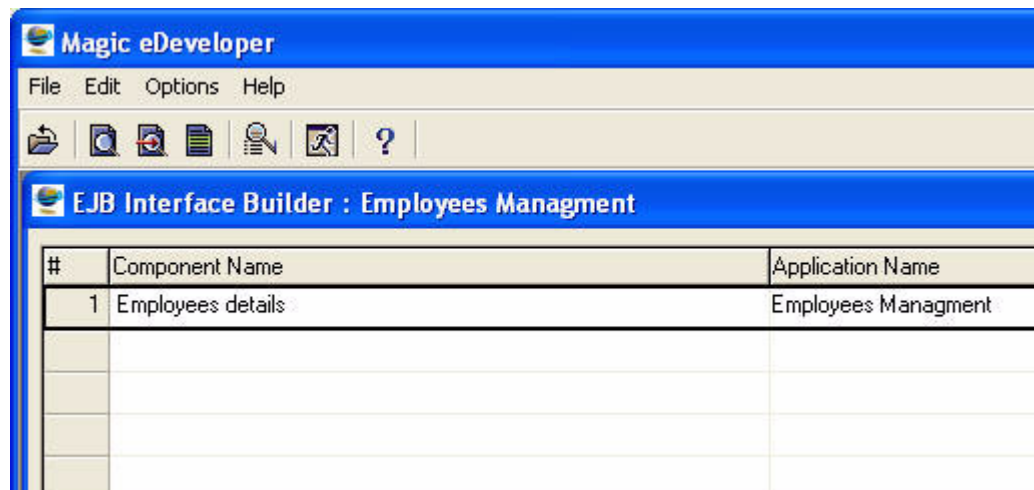


Figure 14-11 The EJB Repository

The **EJB Interface Builder** columns are described below:

- **Component Name** - The component name is mandatory and cannot be longer than 30 characters.
- **Application Name** - eDeveloper automatically displays the name of the current application. You cannot change the application name.
- **Methods** - eDeveloper Automatically displays the number of programs assigned to the EJB interface file.

EJB Programs Repository

The **EJB Program** columns are described below:

- Program Name - eDeveloper automatically sets the specified task according to the program name selected from the Program list. Click the **Select Programs** button to display the programs that can be selected. A program must be a batch task and have a public name. The program name cannot be changed.
- Public Name - eDeveloper automatically displays the public name of the selected program. The public name cannot be changed.
- Arguments and Returned Values - eDeveloper displays the total number of arguments. Zoom from this column to modify the argument name and Java type from the Arguments for **Public Program** dialog box.
- Remarks - You can add reference information about the program.

Click the **Verify Structure** button to start the checker. If there are no errors in the program structure, eDeveloper displays a confirmation message that the structure is okay. If there are errors, eDeveloper displays a warning.

EJB Settings

From the **Options** menu, click **Settings** to select from the following J2EE enterprise servers:

- BEA WebLogic Version 5 or 6
- Sun Reference Version 1.3
- JBOSS
- Oracle Enterprise Server Version 9i
- Sun ONE Enterprise Server Version 7
- Fujitsu Interstage Enterprise Server Version 5
- WebSphere

EJB Environment Variable Path Settings

The path settings for the following EJB environment variables are described below:

- MG_J2EE_HOME should point to the J2EE Server directory.
- MG_JAVA_HOME should point to the JDK directory.
- MG_CLASSPATH should point to directory of where the mgejbgnrc.jar file is stored.

Generating an EJB Component File

After specifying your EJB programs and settings, click **Build Jar File** from the **Options** menu or press **CTRL+I**. The **Generate Jar File** dialog box appears with the file name.

If the directory for the EJB file does not exist, the EJB Interface Builder will create it.

If the Jar file name already exists in the defined path, eDeveloper displays the following message:

```
[file path] [file name].jar already exists!
```

Click **Yes** to overwrite the existing file. Click **No** to define a name.

eDeveloper displays a screen indicating whether the Jar file has been generated successfully. If eDeveloper fails to generate the Jar file, the error details are entered in the error log.

Additional Generated Jar Files

Additional files are created, depending on the selected J2EE enterprise server. One of these files is called ClientTest. ClientTest is a small Java application that provides an easy way for you to check the connection between the Java application and the eDeveloper application.

Java Generator

The Java Component Generator creates a component with programs that call Java class or EJB functions. For more information, see Chapter 16, Java Integration.

XML Generator

The XML Component Generator (XCG) creates all the required eDeveloper objects to integrate with a specific XML type, as determined by the XML schema. For more information, see Chapter 17, XML Component Generator.

e Developer lets you handle any registered COM object in your Magic application by using an ActiveX control in the GUI Display form or an OLE object that implements OLE automation.

A COM object is handled through a Select operation of a variable defined by its attribute and other properties to be an ActiveX or an OLE object. Any activity related to the COM object will be done in relation to the Select operation defined for the object.

In this chapter:

• OLE and ActiveX
• Defining COM Object Fields
• Calling a COM Object
• Handling ActiveX Events
• Runtime Behavior
• Placing an ActiveX Control on a Form
• OLE Variables and BLOB Variables as OLE Content
• COM Interface Builder

OLE and ActiveX

A COM object variable can be defined as either OLE or ActiveX.

An ActiveX object refers to a group of COM objects that provide an embedded user interface and can be placed on a GUI Display form.

Runtime manipulation of an ActiveX object can be done through its GUI representation and through OLE automation commands.

OLE object refers to all registered COM objects, including ActiveX objects. OLE objects cannot be placed on the form and their runtime manipulation can be done only through OLE automation commands.

Defining COM Object Fields

You can define a COM object field either directly as a variable in a task, as a column in a table, or as a field model in the Models repository.

Attribute

The main setting of the COM object field is the attribute property.

The attribute of the field determines if it is a displayed object (ActiveX) or a general COM object (OLE).

Object Name and Type Library Settings

The COM object field must be set with a specific object definition. The Object name property is essential and mandatory for the complete definition of the COM object.

For OLE objects you should first define the type library of the object.

Zoom in from the Type library property to view and scroll through all the COM objects that are registered on the current machine.

Once you located the object's type library entry you should select it and then zoom in from the Object name property to select the actual object.

For ActiveX objects, you should zoom directly from the Object name property to select the ActiveX method you wish to use. After you confirm the Object name. Magic automatically sets the appropriate value in the Type library property.

When these two properties are set and the field definition is confirmed, the Object name cannot be modified. The type library can be modified only to a different version of the already selected type library.

Calling a COM Object

You can handle a COM object defined in a task using the COM option of a Call operation.

You can use the Call COM operation to call a method of the COM object, set a property of the COM object, or get the value of a COM object property.

All the required settings of the Call COM operation are available from the main dialog of the operation. Zoom to the Call COM Object dialog from the third column of the Call COM operation.

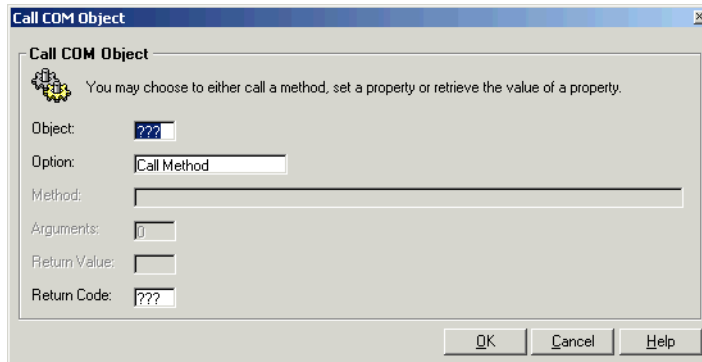


Figure 15-1 Call COM Object

The Call COM object fields are:

- Zoom in from this field to browse through the available properties of the selected object, as shown in Figure 15-2.

Figure 15-2 Call COM Arguments

The Arguments dialog displays the name of the expected arguments, the expected internal type, that is the Magic attribute, the external type of the argument, and whether the argument is optional.

Enumerated arguments are arguments that are set by a collection of values represented by strings. Click the Value button to display the supported enumeration, as shown in Figure 15-3.

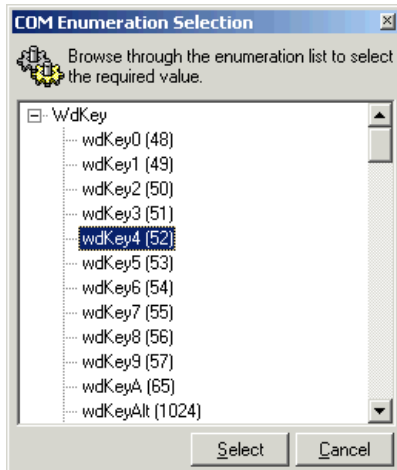


Figure 15-3 COM Enumeration Selection

This list displays the enumeration names and values. You may select a value that adds an expression with the selected value as the passed argument.

- Return Value or Argument Value - This field varies according to the selected option.

For a Call Method option of the method that has a return value, this field enables you to select a variable that will be updated with the return value of the method after the method is completed. Zoom in from this field to see the details of the returned value and to assign a variable to accept the returned value.

For the Get Property operation, this field enables you to select a variable that will be updated with the return value of the property. Zoom in from this field to see the details of the returned value and to assign a variable to accept the returned value.

For the Set Property operation, this field enables you to select a value by which the property will be set. Zoom in from this field to see the details of the expected value and to assign a value to by which to set the property.

- Return Code - The return code of the operation for indication of success or failure.

Handling ActiveX Events

At runtime, Active-X objects can trigger events. The Magic event handlers can handle these events. To handle an Active-X event create a handler, and set its type to be Active-X, as shown in Figure 15-4.

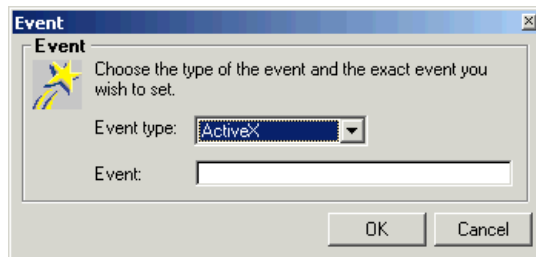


Figure 15-4 Selecting an ActiveX event

Zoom from the Event field to select the object for which the handler is set, as shown in Figure 15-5.

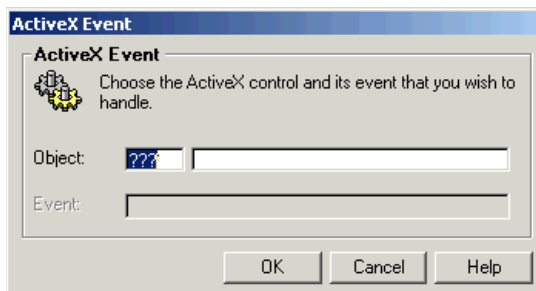


Figure 15-5 Defining an ActiveX event

You may select the object by zooming from the first object field to select an ActiveX variable. This will make the handler a specific handler for the selected variable. You may also select the object by zooming from the second field to select the object name. This will make the handler a general handler for this object for any instance of the object in the task or the runtime task tree.

When you have selected the object, you may zoom from the Event field to view the supported events and to select the event you wish to handle.

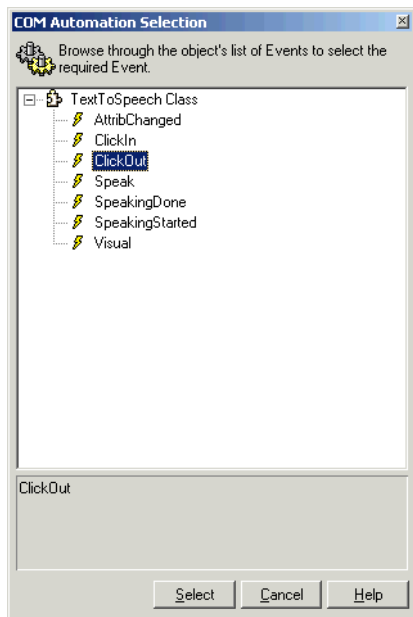


Figure 15-6 COM Automation Selection

When you select the event to be handled, Magic checks if the object passes an argument. If an argument is passed, Magic offers to automatically create Select virtual operations according to the passed arguments of the event.

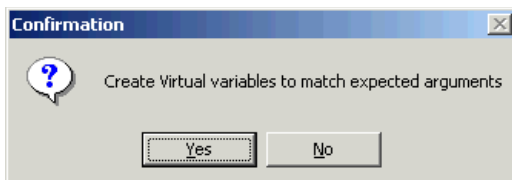


Figure 15-7 Confirmation

If you accept, Magic creates the corresponding Select virtual operations in the handler.



Only ActiveX controls support event-based interaction, OLE objects do not support event-based interaction.

Runtime Behavior

When a task with defined OLE or ActiveX variables is opened, Magic automatically instantiates the defined objects.

Any Call COM operation is executed for the instantiated object.

When an ActiveX object triggers an event that is handled by the Magic application, the appropriate handle defined for the triggered event will be executed.

When the task that instantiated the object is closed, the object is automatically released.

Passing Objects as Arguments

You can pass an object as an argument to another task, usually to have the other task continue and manipulate the created object.

For this purpose, you need to create a Select operation in the called task that will enable you in development time to handle the object using the Call COM operation and the event handler. This Select operation should be a select parameter operation, so the calling task can pass its already instantiated object.

Manual Object Instantiation

The default setting of an OLE or an ActiveX variable is set to be instantiated automatically by the Magic engine when opening the task in which the COM object variable is defined. The object is released when the task is closed.

If you wish to manually instantiate and release an object, you can do so by setting the Instantiation property of the field definition to None, and create and release the object by using the COMObjCreate and COMObjRelease functions.

For more information about these functions, refer to Chapter 8, Expression Rules.

Referring to an Already Created Object

Every COM object is identified by an internal number, also known as a handle.

You may manually instantiate an object in one task, keep its handle, and have another task refer to the created object by using its handle.

This is done by using the COMHandleGet and COMHandleSet functions.

For more information about these functions, refer to Chapter 8, Expression Rules.

Retrieving COM Related Error

Use the COMError function to retrieve information regarding the last COM-related error. For more information about this function, refer to Chapter 8, Expression Rules.

Placing an ActiveX Control on a Form

Variables of ActiveX objects can be placed on a GUI Display form.

You can place the ActiveX control on the form only by dragging and dropping the variable of the object. You cannot place an ActiveX control from the Controls palette.

OLE Variable and BLOB Variable of OLE Content

There is a great distinction between an OLE attribute field and a BLOB attribute field set as an OLE display style.

A BLOB field set as an OLE control appears as an embedded OLE object on the form and to let the OLE application handle its display, content, and manipulation. Magic simply displays the object, enables the end-user to modify it by activating the underlying application and store its binary content.

An OLE field is used for OLE automation capabilities. An OLE field cannot be placed on the form and cannot be displayed by the Magic application.

COM Interface Builder

The Component Object Model (COM) defines an application programming interface (API) that lets you create components for use in integrating custom applications and allows diverse components to interact. The components must adhere to a binary structure specified by Microsoft, which allows components written in different languages to interact.

The Magic COM Builder lets you build a Magic COM Interface (MCI) file that can be integrated in a framework of reusable components supported by Microsoft.

COM Interface Builder Repository

You can create a COM object from the COM Objects repository, as shown in Figure 15-8.

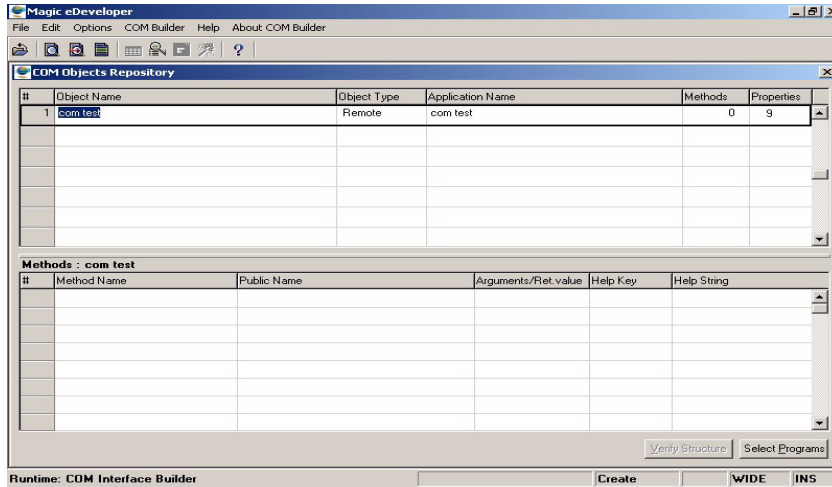


Figure 15-8 COM Interface Builder

The COM Interface Builder columns are described below:

- **Object Name** - The object name is the same as the current application. The object name can be modified. When there is no application name, Magic provides a default name, COMObjectXX, where XX represents the identification number assigned.
- **Engine Type** - Select either Local or Remote engine. A local engine object type requires a local engine loaded on the same machine as the COM object host. A remote engine object type executes Magic application programs as requested by a remote Magic server using a known Magic broker.
- **Application Name** - Magic automatically displays the name of the current application. This field cannot be parked on or modified.
- **Methods** - The number of methods included in the COM interface file. Zoom from the Methods column to display the Methods repository.

- Properties - Displays the number of predefined properties of the object according to its type. Zoom from the Properties column to display the Properties repository.

Methods Repository

The Methods repository lists all the methods defined for the object. These methods are mostly selected public programs of the current application that are available as methods by using the COM interface. For a local engine type, the COM builder creates two predefined methods.

Add or remove the methods by clicking Select Programs to display the list of available public programs. Select or clear the check box of each public program to select or remove it from the Methods repository.

The Methods Repository columns are described below:

- Method Name - This is the method name as specified in the COM interface. The method name can be changed, but it cannot contain illegal characters nor can the name have digits as the prefix.
- Public Name - Automatically displays the public name of the selected program. This field cannot be modified.
- Arguments and Returned Values - Displays the total number of arguments. Zoom from this column to handle the arguments of the public program and its returned value.
- Help Key - You can enter a number that can be set to open a help page. The help key cannot be more than four digits.
- Help String - You can enter a help message up to 512 characters.

Click Verify Structure to start the checker. If there are no errors in the method structure, Magic confirms that the structure is okay. When errors are found, Magic issues a warning.

Predefined Local Methods

When you select the Local engine, Magic automatically displays the predefined methods, MagicEngineLoad and MagicEngineUnload, in the Methods repository. These methods cannot be modified except for the help key and help string.

Properties Repository

Zooming from the Properties column lets you display the supported properties for the object.

The list of properties is predetermined according to the object type. You cannot add or remove a property.

For each property you can set a default value if a default value is applicable, and also the help key and the help string.

Local and Remote Engine Properties

The local engine properties are:

- **ApplicationName** - This property specifies the name of the application to be opened in runtime and which public programs will be executed.
- **WindowHandle** - This property is used to retrieve the handle of the Magic window after it is loaded. This property cannot be assigned with a default value.
- **CommandLineParams** - A series of environment parameters that can be used when loading a Magic engine.
- **MagicEnginePath** - The engine path name. The Magic engine loaded at runtime is specified from the path defined in this property. If this property is not set, Magic will load the Magic engine from the file path specified in the registry of the Magic installation.
- **EngineLoadTimeout** - The timeout value after which the MagicEngineLoad method will fail if the application cannot load and connect to a local Magic engine.

The remote engine properties are:

- `ApplicationName` - This property specifies the name of the application to be opened in runtime and which public programs will be executed.
- `MessagingServer` - Magic broker host name and port number.
- `BrokerTimeout` - The timeout value for the Magic broker.
- `RequesterTimeout` - The timeout value for the Magic requester.
- `Priority` - The priority number of the requests that are generated for the method calls.
- `UserName` - The user name for the Magic Remote engine.
- `Password` - The password for the Magic Remote engine.
- `AltMessagingServer` - An alternate Magic broker.
- `RetryMainTime` - The timeout value for the COM object to retry connecting to the main broker.

Object Settings

The Object Settings dialog provides additional settings for the COM object:

- General Settings
- Help Settings
- Class ID
- Information

General Settings

The General settings are:

- Version - You can enter a version number for the COM object. The default value is 1.0.
- Program ID - Displays the COM program identifier by which the object is registered. The Program ID is read-only and is automatically generated by the builder according to the object name.
- Type Library Name - Displays the type library name by which the object is registered. The Type Library Name is read-only and is automatically generated by the builder according to the object name and version.

Help Settings

Methods that were set with a help key will look for the help key in this help file.

- Library Help File - Defines the help file for the library.
- Library Help Key - Defines the help key in the help file that provides a description of the library.
- Library Help String - Defines the help string that provides primary information about the library.
- Object Help - Defines the help file for the Com object.
- Object Help Key - Defines the help key for the Com object.

Class ID

If you wish to generate an object using a preset collection of IDs, you may click Set CLSID to open the Class ID modification dialog, as shown in Figure 15-9. It is best to let the builder set the CLS ID collection automatically and not to set it manually.

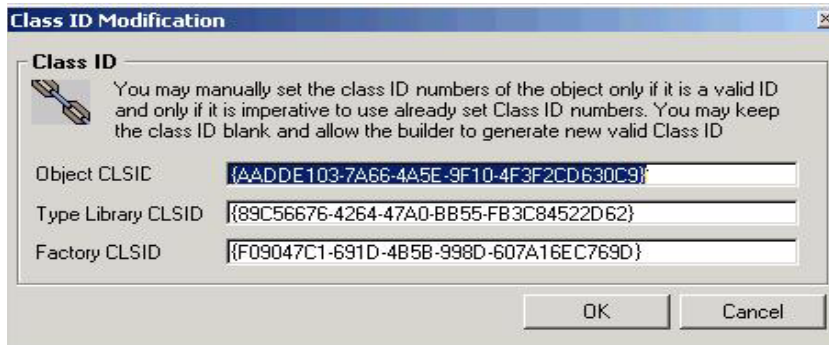


Figure 15-9 Class ID Modification

Either all class identification fields are blank or they all must have a value.

Information

You can enter additional reference information about the COM object. This information is not part of the generated object but is kept solely in the builder for future reference.

Generating a COM Object

After specifying your COM object settings, methods, and properties, click Generate Object from the COM Builder menu or press **CTRL+I**. The Generate COM File dialog appears with the file name, as shown in Figure 15-10.



Figure 15-10 Generating a COM file

If the directory for the COM object does not exist, the COM Interface Builder creates it.

Note: The COM Interface Builder does not support BLOB, OLE, Active-X, or Vector data attributes.

Registering the Object

After the object is created, you should register the COM object on the machine intended for its use.

Local COM Object Runtime Behavior

The Magic engine is not automatically loaded when the local Magic OCX object is loaded. The user must instruct the COM object to load and connect to a Magic engine by using the MagicEngineLoad method described below.

MyObject.MagicEngineLoad Method

When an object is instructed to load an engine, it executes the Mgrntw.exe located in a path defined by the MagicEnginePath property.

If the property is not set, the object will try to load the Magic runtime engine from the installation directory that is set in the registry by the Magic product installation.

When the engine is loaded, the COM object connects to it and the Magic engine is available to handle the method calls.

Activating Methods for Public Programs

When the connection to a Magic COM engine is established, the container can activate the methods of the COM object that runs the corresponding programs.

For example: `MyObject.AddCustomer('John','Smith')`

If the corresponding program is found, it will be executed. When the program has been completed, the method returns the program return value.

The execution of the public programs is synchronic, which means that the call method will be completed only after the called public program completed its execution.

Properties

The properties for a Remote COM object are:

- **ApplicationName** - This property keeps the name of the application by which all methods should be handled.

Each public program that corresponds to a Method call is expected to be found in the application defined by this property. At any stage this property can be set or retrieved. Any modification of the application name affects the next method call.

- **WindowHandle** - This property keeps the window handle value of the loaded Magic engine. The window handle can be retrieved by this property and cannot be set manually.

For example: `MyObject.WindowHandle` can return a numeric value.

When no Magic engine is loaded for the object or when the loaded engine is in background mode, the property returns zero.

- **CommandLineParams** - A series of added environment parameters to be used when loading a Magic engine. Any modification to this property can take effect only when the property is set prior to a `MagicEngineLoad` method call.

For example, `MyObject.CommandLineParams="/ApplicationStartup=B"` loads the engine in background mode.

- **MagicEnginePath** - This property sets the path of the Magic runtime engine. Any modification to this property can take effect only when the property is set prior to a `MagicEngineLoad` method call.

If no engine path is set, the Magic runtime engine is loaded from the installation directory of the Magic product as defined in the registry.

- **EngineLoadTimeout** - The timeout after which the `MagicEngineLoad` method will fail if it could not load and connect to a Magic runtime engine.

The Timeout value is defined in seconds. A zero value means zero seconds, which results in an immediate failure of the MagicEngineLoad method.

Remote COM Object Runtime Behavior

A COM object set for a Remote Magic engine enables you to activate public programs on a Magic enterprise server in the form of remote requests. Every method call is executed as a Magic remote request. The COM object in this case serves as a requester and can be set with various properties that affect its activation.

To enable a COM object for a remote engine to execute properly, the correct configuration of a Magic enterprise server should be constructed and accessible to the COM object.

For more information about setting the Magic server and distributed application architecture, refer to Chapter 19, Distributed Application Architecture.

Messaging layer

The Remote COM object, serving as a requester, requires the Messaging layer to be available to the application that hosts the remote COM object.

You should make the messaging layer dynamic library file, Mgrqgnrc94.dll, available for the application that hosts the remote COM object by either placing the dynamic library file in the working directory of the host application, or set the PATH environment setting of the computer to include the directory in which the dynamic library file resides.

Activating Methods of Public Programs

For every method call to a Magic public program, the COM object issues a synchronous request to a Magic enterprise server by using a Magic broker.

The following COM object settings, which are required for any remote requester, must be correctly set:

- The messaging server, which is the Magic broker, should be active and available. The `MessagingServer` property of the COM object should be set with the Magic broker location.
- A Magic engine should be connected to the Magic broker.
- The application specified in the `ApplicationName` property of the COM object must be defined in the application list of the Magic enterprise server.

Properties

The properties for a local COM object are described below. Any modification of these properties will affect the next method call.

- `ApplicationName` - This property keeps the name of the application by which all methods should be handled.

Each public program that corresponds to a Method call is expected to be found in the application defined by this property. At any stage this property can be set or retrieved.
- `MessagingServer` - The address of the Magic broker that receives the requests. The address is specified in a host and port number syntax, such as `mymachine/3300`.
- `BrokerTimeout` - The maximum time in seconds that the COM object waits for an available engine to be returned by the Magic broker. If the Magic broker cannot provide an available engine within the specified time, the method call fails.
- `RequesterTimeout` - The maximum time in seconds that the COM object waits for the called public program to be completed. If the value is 0, the COM object waits indefinitely.
- `Priority` - The priority of the requests generated by the method call. The priority is determined by a number from 0 to 9, where 9 has the highest priority.
- `UserName` - The user name used by the Magic enterprise server for the execution of the call method.

- Password - The password used by the Magic enterprise server for the execution of the call method.
- AltMessagingServer - You can define an address of an alternate Magic broker. The COM object will refer to this address when the connection to the main messaging server fails.

The following properties can be set only prior to the first method call. After the first method is called, the values of the properties below cannot be changed.

- RetryMainTime - When the COM object connects to the alternate Magic broker, as specified in the AltMessagingServer property, the Magic engine will try to reconnect to the main Magic broker in the time interval specified. The time interval is the number of minutes the COM object waits before trying to reconnect to the main broker, as set in the MessagingServer property. The default value is 5. The minimum value is 1.

COM Object Errors and Troubleshooting

Error values are returned in the form of a result handle, a 32-bit number, also known as HRESULT.

Local Magic Engine

The list below describes the HRESULTS that may be returned by the COM object of a local Magic engine.

HRESULT	Description	Troubleshooting
0x803307D2	Magic engine not loaded	<p>This error is usually encountered when a method is called and no Magic engine is available.</p> <p>You should load the Magic engine using the MagicEngineLoad method prior to all other method calls.</p>

HRESULT	Description	Troubleshooting
0x803307D3	Failed to load Magic engine	<p>The MagicEngineLoad property failed to load the Magic runtime engine. This error can be encountered when:</p> <ul style="list-style-type: none"> • The Magic runtime engine could not be found in the path defined by the MagicEnginePath property. • The MagicEnginePath property is blank, and the Magic runtime engine could not be found in the path defined by the registry key of the product that specifies the installation directory. • The Magic engine cannot be loaded due to erroneous environment settings, such as using a wrong license. Make sure that the Magic runtime engine is properly installed and that it can be loaded properly with the environment settings defined for it.

HRESULT	Description	Troubleshooting
0x803307D4	Failed to load Magic engine - Bad registry path	<p>The MagicEngineLoad method failed to load the Magic runtime engine through the install directory specified in the registry because the expected registry key could not be found. This error is encountered when:</p> <ul style="list-style-type: none"> • The Magic product of the expected version is not installed. • The registry key was manually modified. • The Magic product was not properly installed resulting in improper registry key settings. <p>You should make sure that the Magic product is properly installed or set the MagicEnginePath property to the Magic runtime engine directory.</p>
0x803307D5	Failed to unload Magic engine	<p>The MagicEngineUnload method failed to unload the Magic engine. This error is encountered when:</p> <ul style="list-style-type: none"> • The Magic engine was terminated prematurely. • The Magic engine is busy due to activation external to the COM object.
0x803307D6	Failed to load Magic engine - Timeout occurred while trying to synchronize with engine	<p>The timeout defined by the EngineLoadTimeout property passed before a Magic engine was loaded. You should make sure that the EngineLoadTimeout property is properly set to provide sufficient time for the Magic engine to load.</p>

HRESULT	Description	Troubleshooting
0x803307D7	Failed to load Magic engine - No HTTP connection with engine	<p>The MagicEngineLoad method fails to establish an HTTP connection with the Magic engine. This error is encountered when:</p> <ul style="list-style-type: none"> • No TCP/IP layer is installed. • The Magic engine was set to serve remote requests. <p>You should make sure that the TCP/IP layer is installed and that the Magic engine is not set to be activated as a request server.</p>
0x803307D1	General error	An unmapped error.

Remote Magic Engine

The COM object of the remote Magic engine executes the methods as remote requests.

The errors that may be encountered are request-related errors. The request error number is delivered as part of the HRESULT, which is a DWORD hexadecimal number. The low word of the HRESULT represents the positive value of the actual error number.

For example, the request error of `Messaging server not found` has an error code of -102, the low word of the HRESULT should be 0066, representing the decimal value of 102. For this error, the complete HRESULT is 0x80330066.

Java Integration lets Magic interface with a Java class or Enterprise JavaBeans (EJB) by using pseudo-references that represent instances of Java classes or EJB files. For Java classes, you can access static methods and variables without the pseudo-reference. Instances, static methods and variables are invoked by Java and EJB Functions. The Java Component Generator is a wizard that generates a component containing programs that call Java class or EJB functions.

In this chapter:

• Java Terminology
• Java and EJB Functions
• Code Pages
• Type Signatures
• Runtime Engine Behavior
• Java Component Generator

Java Terminology

The terms below describe the various Java class elements.

Java Class	A Java class is a template definition of the methods and variables in a particular kind of object. An object can be a specific instance of a class and can contain real values instead of variables.
Class Instance	A specific implementation of the Java class.
Static Class Method	A class method that can be called without instantiating the class.
Static Class Variable	A class variable that can be accessed without instantiating the class.
Constructor Method	A function that is automatically initiated when a new class object is instantiated.
Pseudo-Reference	A Magic BLOB variable that represents a class instance. Any method that creates a new class instance returns a pseudo-reference to the newly created object. At a later time, programs that want to operate with the object must use the identifier.
Java Native Interface	Lets non-Java programs interact with Java programs.
Java Virtual Machine	Interprets the bytecode into code that runs on actual computer hardware.

Java Name Directory Interface	Enables Java platform-based applications to access multiple naming and directory services.
J2EE Enterprise Server	A Java platform enterprise server that creates standardized, reusable modular components.
Enterprise JavaBeans	An architecture for defining Java program components that run on a client/server network.
Java Development Kit	A program development environment for writing Java applets and applications.

Java and EJB Functions

eDeveloper can create an instance of a Java class that lets you activate a method, or query and update variables using a pseudo-reference. The pseudo-reference is a BLOB variable that represents the Java class instance. When a method is invoked, the BLOB variable returns values to the eDeveloper program as function return codes.

Class static methods and variables can also be accessed without having an instance of the Java class by using the class name and the method or variable name.

The eDeveloper functions below are used to simulate the calling methods in Java. Each context of the engine internally maps between the pseudo-reference and the actual reference.

For example, an eDeveloper program can call the JCreate function to create a pseudo-reference as Virtual Variable A. The actual reference to the instance is then internally associated with Virtual Variable A. Any eDeveloper program in the context is able to access the methods and variables of the object by using JCall(A,...).

The Java functions below can be called by an eDeveloper program to:

- Create a new instance of a Java class
- Call a method directly from the Java class or from an instance
- Report and return exceptions
- Query and update variable values

The Magic Java functions are:

JCreate	Obtains a new instance of a Java class.
JCall	Lets a Magic program call the method of an instance of a Java class.
CodePage	Sets a code page that can be used when converting Java characters and strings to a Magic Alpha type and converting from a Magic Alpha type to Java characters.
JCallStatic	Lets a Magic program call a method from a Java class.
JException	Returns a pseudo-reference to the last exception of the current context.
JExceptionOccurred	Returns a True value when the last J* or EJB* function throws an exception.
JExceptionText	Returns the description of the last exception and an optional backtrace of the stack. This function refers to the last exception thrown during the last j* or ejb* function.
JExplore	Describes a Java class.
JGet	Queries a value from an instance variable.
JGetStatic	Queries a value from a class variable.
JInstanceOf	Simulates the Java's instanceof operator.

JSet	Updates an instance variable.
JSetStatic	Updates a class variable.

The EJB functions below enable Magic to explore and invoke Enterprise JavaBeans.

EJBCreate	Creates an instance of the EJB and returns a pseudo-reference from which any of the other related Java functions can be activated.
EJBExplore	Provides a description of the EJB instance.

For more information about Java and EJB functions, see Chapter 8, Expression Rules.

Code Pages

The Code page identifiers below can be used with the CodePage function to specify a set of language characters, used when converting Java strings to the Magic Alpha data type and back.

Identifier	Language
037	EBCDIC
437	MS-DOS United States
500	EBCDIC 500V1
708	Arabic (ASMO 708)
709	Arabic (ASMO 449+, BCON V4)
710	Arabic (Transparent Arabic)
720	Arabic (Transparent ASMO)
737	Greek (formerly 437G)
775	Baltic
850	MS-DOS Multilingual (Latin I)

Identifier	Language
852	MS-DOS Slavic (Latin II)
855	IBM Cyrillic (primarily Russian)
857	IBM Turkish
860	MS-DOS Portuguese
861	MS-DOS Icelandic
862	Hebrew
863	MS-DOS Canadian-French
864	Arabic
865	MS-DOS Nordic
866	MS-DOS Russian
869	IBM Modern Greek
874	Thai
875	EBCDIC
932	Japanese
936	Chinese (PRC, Singapore)
949	Korean
950	Chinese (Taiwan; Hong Kong SAR, PRC)
1026	EBCDIC
1200	Unicode (BMP of ISO 10646)
1250	Windows 3.1 Eastern European
1251	Windows 3.1 Cyrillic
1252	Windows 3.1 US (ANSI)
1253	Windows 3.1 Greek
1254	Windows 3.1 Turkish
1255	Hebrew
1256	Arabic

Identifier	Language
1257	Baltic
1361	Korean (Johab)
10000	Macintosh Roman
10001	Macintosh Japanese
10006	Macintosh Greek I
10007	Macintosh Cyrillic
10029	Macintosh Latin 2
10079	Macintosh Icelandic
10081	Macintosh Turkish

Type Signatures

The Java Native Interface uses the Java Virtual Machine's definition of type signatures, as shown in the table below.

Type Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L	fully qualified class
[type	type[]
(arg-types) ret-type	method type

For example, the Java method, **long f (int n, String s, int[] arr)** has the type signature, **(Ljava/lang/String;[I)J**

where:

- **int n** - I
- **String s** - Ljava/lang/String;
- **int[] arr** - [I
- **long f** - J

The command line entry, `javap -s <class name>`, analyzes a class and lists the signatures of its methods and variables.

Runtime Engine Behavior

This section describes the runtime behavior of programs that call Java functions. For the Java functions to work at runtime, the Java Development Kit or the Java Runtime Environment must be installed, and the called Java classes must be in the environment class path.

Life Cycle

All pseudo-references obtained by an engine from a Java or EJB function are released to the Java Virtual Machine when the context is terminated.

Multi-Threading

Each engine context has its own pseudo-references. Pseudo-references of one engine context has no influence over pseudo-references of another engine context, even if these pseudo-references represent the same Java class. Access to static class methods or variables need to be synchronized on the Java side by using the `synchronized` keyword in the method.

Browser-Based Programs

Magic can create a pseudo-reference from the JCreate function at any time during the engine context's life cycle and use the pseudo-reference during subsequent events of the engine context.

Conversion Tables

The table below describes the automatic conversions by the signature of outgoing values for JCreate, JCall, JSet, JCallStatic, and JSetStatic functions and the return values for JCall, JGet, JCallStatic, and JGetStatic functions.

Magic Types	Java Type	
	Primitive Type	Object Type
Alpha, Memo, BLOB	char, byte, char[], byte[]	String, StringBuffer, Character, Byte, Byte [], Character []
Logical	boolean	Boolean
Numeric, Date, Time	byte, short, int, long, float, double	Byte, Short, Integer, Long, Float, Double

The table below displays the default mapping for the return value types.

Java Types	Java Type	Magic Types
Primitive Type	Object Type	
char, char[]	String, StringBuffer, Character	Alpha (up to 32K) BLOB (above 32K)
byte[]		BLOB

Java Types	Java Type	Magic Types
Primitive Type	Object Type	
boolean	Boolean	Logical
byte, short, int, long, float, double	Byte, Short, Integer, Long, Float, Double	Numeric

Returning Pseudo-Reference Values

You can disable the automatic conversion described in the previous section by using: **signature*p**

For an automatic conversion:

JCall(A, 'increment2', '()Ljava/lang/Long;') -The function returns the numeric value of the Long object

For disabling an automatic conversion:

JCall(A, 'increment2', '()Ljava/lang/Long;*p') -The function returns a pseudo-reference (BLOB variable) of the Long object.

Errors and Exception Handling

Magic Java functions can fail to load a class, locate a requested constructor by a specified type signature, or catch an exception.

External Errors for a Method

When an exception is thrown by a method of the Java class, JCreate, JCall, and JCallStatic functions return a Null value.

You need to call either the JException or the JExceptionText function to receive a pseudo-reference to the exception or the exception text description.

Magic can call the JException or the JExceptionText functions repeatedly until another function is called.

The JExceptionOccurred function provides an easy way of knowing that an exception occurred.

JExceptionOccurred

Internal errors are returned to you as Null or False.

The JExceptionText returns one of the following messages:

- JVM could not be loaded
- Class not found
- Method not found
- Failure to convert parameters to JNI
- Failure to convert return value from JNI

The JException function creates a reference to a Java exception object that has occurred.

Garbage Collection Mechanism

When closing a task, after applying modifications to updated parameters or return values, Magic automatically releases pseudo-reference values that are not assigned to a variable in the task tree containing the task. This action releases the unnecessary resources.

Environment

Magic can load and activate Java classes without setting the operating system's environment variables. All required environment variables for the Java settings are entered in the [MAGIC_JAVA] section of the Magic.ini file. The Magic-Java section can contain the following settings:

JAVA_HOME - This setting overrides the JAVA_HOME set in the operating system.

CLASSPATH - This setting is used for locating additional classes and is added at the beginning of the operating system's class path string. Semicolons are used as a delimiter to specify several directories. For example, if the operating system has C:\; C:\temp as class paths and D:\ is specified in the CLASSPATH setting under the MAGIC_JAVA section in the Magic.ini file, in runtime the operating system's class path will appear as D:\;C:\; C:\temp.

JVM_ARGS - Additional parameters that can be sent to the Java Virtual Machine, such as JVM_ARGS=Djms.properties=F:\j2ee\config\jms_client.properties

Java Component Generator

The Java Component Generator is a wizard that lets Magic automatically access Java classes or EJB files by creating a component that eliminates the need for you to specify JNI signatures.

The Java Component Generator lets you specify the Java component by defining the:

- Java type
- Java class
- Java interface objects

The Java Class or Enterprise JavaBeans Type

Click Java from the Components menu to start the Java Component Generator. The Java Class and EJB screen opens, as shown in Figure 16-1.

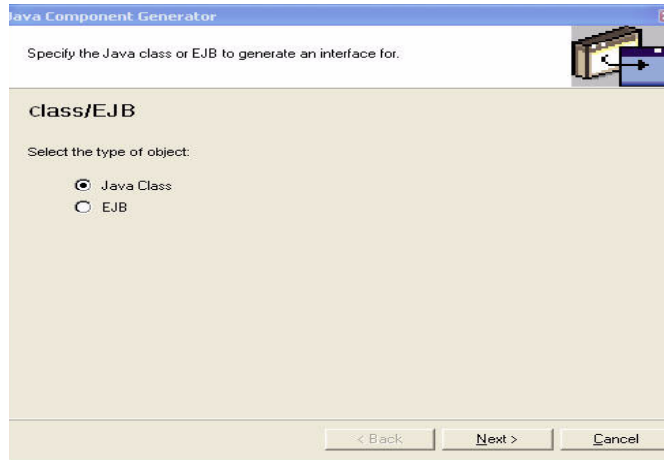


Figure 16-1 Select a Java Class or EJB Type

From the Java Class and EJB screen, you can determine the type of Java component, Java Class or EJB, that will be generated,

If you select the EJB, the following fields are enabled:

- EJB Name - Specify the EJB name as it is displayed in the Java Name Directory Interface (JNDI).
- JNDI Property String - Specify the JNDI property string when needed, such as `java.naming.factory.initial=com.sun.jndi.cosnaming.CNctxFactory,java.` The comma is used as a separator.

When the EJB type is selected, the EJBExplore function is automatically called to retrieve the EJB definition.

The Java Object Browser

You can select a Java class from the tree or specify the Java class name in the Class Name field, as shown in Figure 16-2.

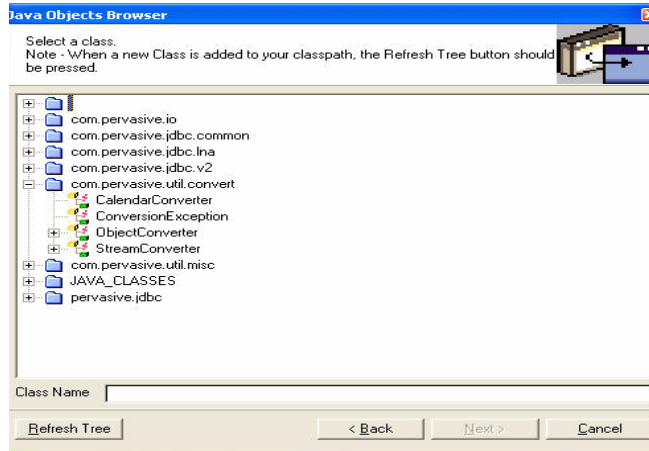


Figure 16-2 Select a Java Class Object

After the Next button is pressed, the JExplore function is automatically called by the wizard. If the function fails, Magic displays the following message:

Failed to find the Java class. Please check that the class is available.

Java Class Structure

You can select the objects to create a component interface, as shown in Figure 16-3.

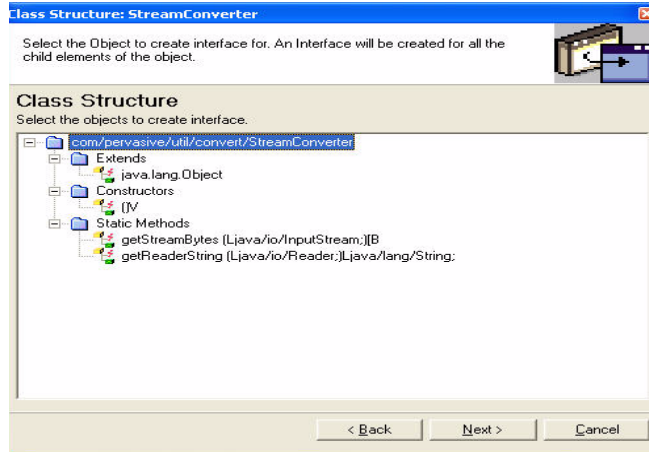


Figure 16-3 Select a Java Class Object

Under each Java class, the following interface objects are displayed:

- Constructors
- Member variables
- Methods
- Static member variables
- Static methods

Only a Java class with content is displayed in the Class Structure screen. For example, when there are no static members, the static member object is not displayed in the tree. Methods are presented with their parameters.

The Java Component Generator creates programs to call the object from the node that you are positioned on and to all of the node's child elements.

The Generated Java Component

When the Java component has been generated, the Java Component Generator displays the Java Component Completion screen, as shown in Figure 16-4.

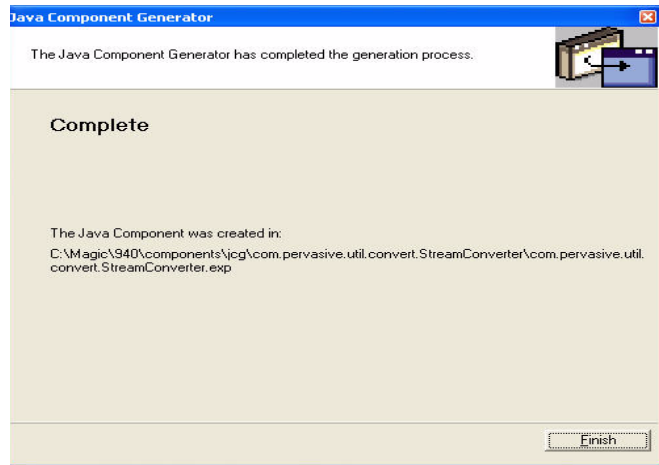


Figure 16-4 The Java Component Completion Screen

The created programs are described in the table below.

ID	Object Type	eDeveloper Object
1	Java Class or EJB if the constructor was selected	The following programs are created: A program with JCreate or EJBCreate functions that return the pseudo-reference BLOB variable. Each constructor has its own program.
2	Method	A program with the JCall function that defines and receives all required parameters and returns the function value.

ID	Object Type	eDeveloper Object
3	Member Variables	<p>For each member, the following programs are created:</p> <p>A program with the JSet function that sets the variable. The program returns True if the function succeeds or False if the function fails.</p> <p>A program with the JGet function that returns the queried value.</p>
4	Static Member Variables	<p>For each static member, the following programs are created:</p> <p>A program with the JSetStatic function.</p> <p>A program with the JGetStatic function.</p>
5	Static Methods	<p>A program with the JCallStatic function that defines and receives all of the required parameters and returns the value of the function.</p>

Created Files

The Java Component Generator crates the files listed below in a specified directory:

- Magic Flat File (MFF) - An MFF application ready to be deployed.
- Magic Component Interface (MCI) - An MCI file that exposes all of the required objects in the generated component.
- Magic Export File (EXP) - An export file of the generated component.

XML Component Generator

17

An XML schema describes a particular dataview hierarchy that can be used to transfer data between applications. The XML Component Generator (XCG) simplifies the integration of an application with an XML schema by generating a component that lets you access an XML document's data from within an application. The XCG uses an XML Schema Definition (XSD) file to create the component.

In this chapter:

• XCG Wizard
• XCG Programs
• Generating the Component
• Namespace Support

XCG Wizard

You can access the XCG wizard from the **Components** menu. Click **Components** and then click **XML** to open the wizard. The XCG wizard can also be opened from the **New Application** dialog.

This wizard lets you:

- Add or delete a new component according to a schema definition
- View and modify existing schema compound details
- Generate XML components as export and MCI files

When the Wizard is loaded, the **XCG Wizard Welcome** screen appears, as shown below in Figure 17-1.

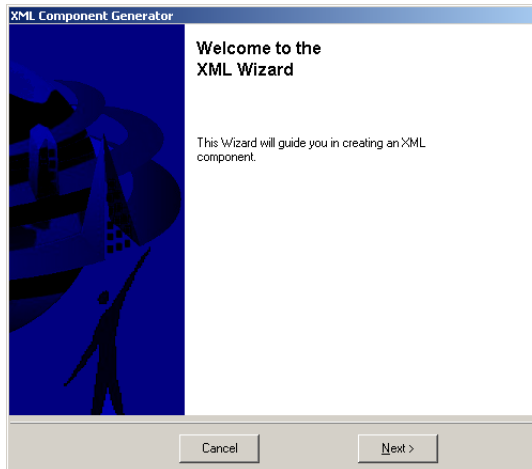


Figure 17-1 XML Wizard Welcome Screen

XCG Main Options

The XCG Main Options screen lets you create, modify, or delete an XML component in the application, as shown below in Figure 17-2.

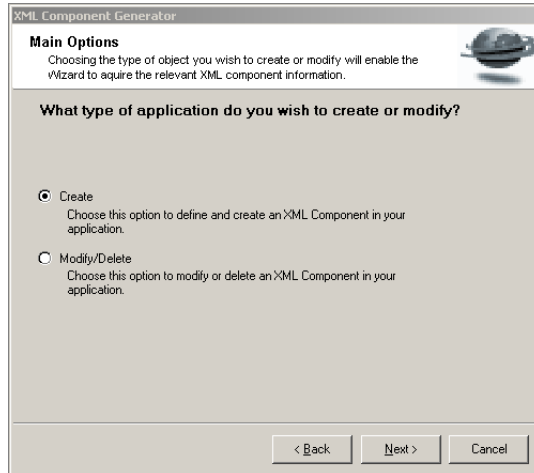


Figure 17-2 Main Options

The options available are:

- Create - You can create a new component for the XML schema.
- Modify - You can modify an existing component definition.
- Delete - You can delete an existing component definition.

Modifying a Component

You can modify or delete a component definition. Changing the definition does not affect the actual component.

The XML Schema screen displays a list of all XML applications that are created using the Wizard, as shown below in Figure 17-3.

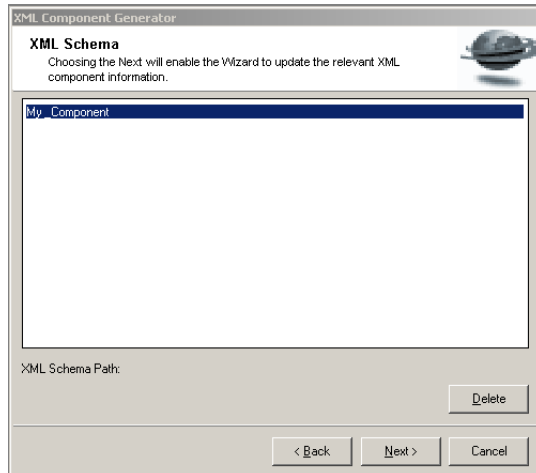


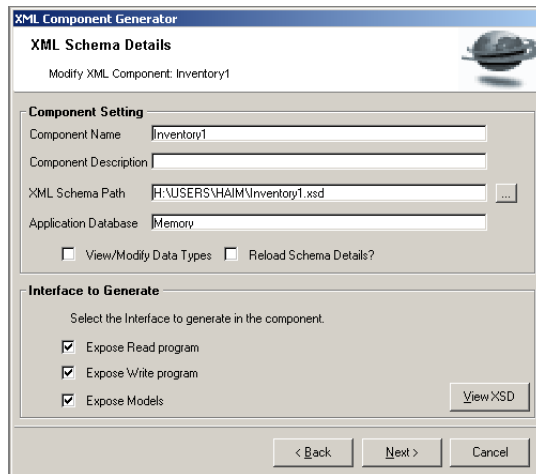
Figure 17-3 Modify or Delete a Component

When you select a row, you can see the location of the XML schema. If no component is specified, the component items are deleted. The XCG Wizard automatically moves to the next screen.

You can click **Next** to make changes to the selected definition or **Delete** to remove the XML schema definition.

XML Schema Details

The XML Schema Details screen is activated when you create, modify, or delete a component option, as shown below in Figure 17-4. Use the screen to define various details about the component.



The screenshot shows a dialog box titled "XML Component Generator" with a sub-tab "XML Schema Details". Below the title bar, it says "Modify XML Component: Inventory1". The dialog is divided into two main sections: "Component Setting" and "Interface to Generate".

Component Setting

- Component Name:** A text field containing "Inventory1".
- Component Description:** An empty text field.
- XML Schema Path:** A text field containing "H:\USERS\HAJM\Inventory1.xsd" with a browse button (three dots) to its right.
- Application Database:** A text field containing "Memory".
- Two checkboxes at the bottom: ☐ View/Modify Data Types and ☐ Reload Schema Details?

Interface to Generate

Select the Interface to generate in the component.

- ☒ Expose Read program
- ☒ Expose Write program
- ☒ Expose Models

A "View XSD" button is located to the right of the checkboxes.

At the bottom of the dialog are three buttons: "< Back", "Next >", and "Cancel".

Figure 17-4 XML Schema Details

The options are:

Component Name - You can enter a name for the component. The name cannot have embedded spaces.

Description - You can enter a description of the component, up to 256 characters long.

Path - The schema file location. You can zoom from the field to browse for the **xsd** file.

View/Modify Data Types - Select the check box to make changes to the global XCG data types. For more information, see the Data Type Management section in this chapter.

Reload Schema Details - Select the check box to reload the XML schema, which is necessary if you decide to make changes to the global XCG data types or for changes made to the eDeveloper definitions. For more information, see the XML Schema Interface Details section in this chapter.

You can access (expose) all Read, Write, and Model XML component interface types. The XCG creates these programs but will expose the interface depending on how the Read Program, Write Program, or Model fields are set, as described below.

- **Expose Read Program** - This program reads the XML file and enters the data into the relevant tables in the component.
- **Expose Write Program** - This program writes an XML file that is based on the XSD structure and the data from the component tables.

The program public names are **Read_schema_name** and **Write_schema_name**, where the schema_name is the actual name of the XML schema.

- **Expose Models** - This option defines whether the component models will be exposed to other applications.

View XSD

You can view the XML hierarchy in the **Schema Interface Details** window by clicking the **View XSD** button, as shown in Figure 17-5.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- edited with XML Spy v4.1 U (http://www.xmlspy.com) by Moshe Levy (MIS) -->
<!-- W3C Schema generated by XML Spy v4.1 U (http://www.xmlspy.com) -->
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
- <xs:element name="INVENTORY">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="BOOK" maxOccurs="unbounded">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="TITLE" type="xs:string" />
- <xs:element name="AUTHOR">
- <xs:complexType>
- <xs:simpleContent>
- <xs:extension base="xs:string">
  <xs:attribute name="Born" type="xs:short" />
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="PAGES" type="xs:short" />
<xs:element name="PRICE" type="xs:string" />
</xs:sequence>
<xs:attribute name="Binding" type="xs:string" use="required" />
<xs:attribute name="InStock" type="xs:string" use="required" />
<xs:attribute name="Review" type="xs:string" />
</xs:complexType>
```

Figure 17-5 Viewing an XSD File

You cannot modify the XSD file by clicking the **View XSD** button. This screen appears for viewing purposes only.

XML Schema Interface Details

The **Schema Interface Details** window displays the Component Compound tree in the left pane. You can park on an entry to view details in the **Simple Element Details** table and component compound options, as shown in Figure 17-6.

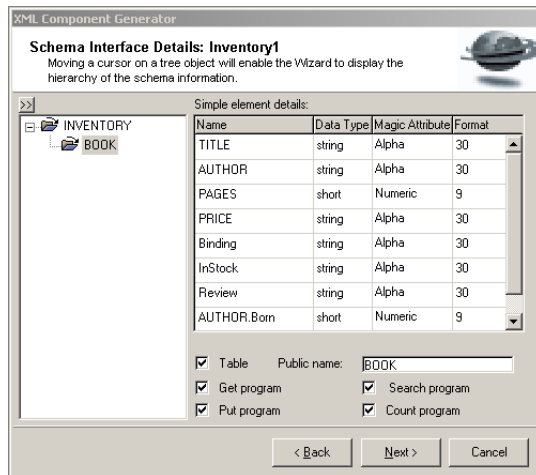


Figure 17-6 XML Schema Interface Details

The **Simple Element Details** table lists the data types with their corresponding Magic attribute and attribute format. The **Simple Element Details** table represents an eDeveloper table. The table initially displays the XSD default values, but you can make changes to these definitions. For example, the XSD default value for a string is Alpha 30. However, you can decide that the string should be Alpha 100.

Important: If the XML Component Generator cannot fetch the element data type, it returns the default mapping of the string element data type.

The settings for the Simple Element Details table are:

Table - When you select the check box, the table is exposed and added to the MCI file, becoming available to other applications.

Public Name - You can change the table public name.

Get program - This program retrieves data from the compound table according to the parent and table identifier. The compound element is named **Get compound_name**, where **compound_name** represents the actual name of the compound.

Put program - This program manipulates data in the compound table according to the parent and table identifier. The compound element is named **Put compound_name**, where **compound_name** represents the actual name of the compound.

Search program - This program retrieves the parent and table identifier according to the available data in the table. The compound element is named **Search compound_name**, where **compound_name** represents the actual name of the compound.

Count program - This program counts the number of occurrences of a selected compound element. The compound element is named **Count compound_name**, where **compound_name** represents the actual name of the compound.

Note: When the Expose, Get, Put, Search, and Count Program check boxes are not selected, the program is generated but without the public name. The program will not be available for another application.

Data Types

You can specify different data types, complying with XSD standards, to be the default data types of the generated component. Select the **View/Modify Data Types** check box from the **XML Schema Details** screen to display the **Data Type Translations Default**, as shown in Figure 17-7.

XML Component Generator

Data Type Translation Defaults

Note - Changing the data types will affect all the created components.

Define the default Datatype translation from XSD types to eDeveloper types.

Data Type	Magic Attribute	Format
anyURI	Alpha	10
boolean	Alpha	1
byte	Numeric	4
date	Alpha	10
decimal	Numeric	10.2
float	Numeric	10.2
int	Numeric	7
integer	Numeric	9
positiveInteger	Numeric	9
short	Numeric	9
string	Alpha	30

< Back Next > Cancel

Figure 17-7 Data Type Translation Defaults

You can add new entries by pressing **F4**. You can also change the Magic attributes and modify the attribute formats. The default data types are read from the Default.txt file that is created in the XCG directory when the XCG repository is invoked.

Component Details

You can define where the component export file is placed by using the **Component Path** screen, as shown in Figure 17-8.

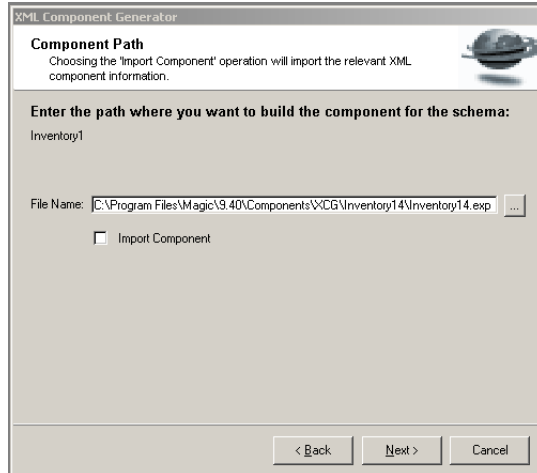


Figure 17-8 Component Path

The options for the component path are:

- File Name - Specifies where the component export file is saved. If the component export file already exists, the Wizard prompts you with the following message: **The component export file already exists. Do you want to overwrite?** Click **Yes** to overwrite.
- Import Component - Determines whether the component will be imported into the current application. If the XCG is executed from the **New Application** dialog, this option does not appear because the component is automatically imported.

If the directory specified in the File Name field does not exist, the following message is displayed:

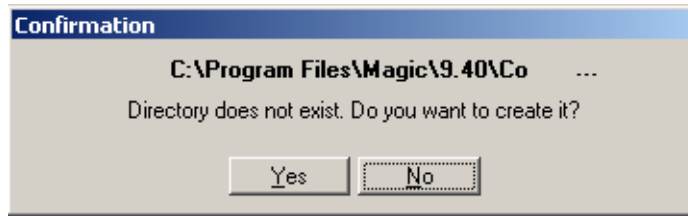


Figure 17-9 Directory Does Not Exist

XCG Programs

The XCG creates an eDeveloper component which is comprised of memory tables that represent the XML data structure hierarchy. The developer can select a different database for the data.

When you generate your new component, the XCG creates several types of programs, as described below. Each program is created in an eDeveloper folder for that program type. The different programs receive arguments and return the appropriate values. The programs are defined in the created MCI file.

Count Program

The Count program receives the parent ID and returns the number of occurrences of the compound element.

DbDel Program

The DbDel program erases all the data in a specific table before executing the Read program to read a given XML file. This program can be used to clear the table before creating data for a new XML output file.

Get Program

The Get program receives the parent and table IDs and retrieves the record data.

Put Program

The Put program receives the parent ID, the table ID, and all the simple elements. The program updates an existing record or creates a new record if no record is found.

Read Program

The Read program receives a file name or a BLOB argument as an input. The program first clears the internal tables of existing data to ensure the consistency of the read. The program then reads the XML data into the tables. When reading the compound elements, the program assigns a parent ID to each element in the list, and then assigns identifiers to the compound elements.

Read programs are nested. The main Read program imports the root data and then calls the root's direct compound program. Each compound program calls its child with the parent ID.

Search Program

The Search program lets you locate specific data in the corresponding table. Each table represents a compound object. The Search program can retrieve the unique identifier of a specific record according to a given value.

The Search program receives all the simple elements of the table as an argument, none of which are mandatory, and returns the parent and table IDs of the located record. If the parent ID is known, the ID can also be sent as an argument, but this is not mandatory. If the search is called without a parent ID, the search returns the index of the first record matching the search criteria. The Search program returns the parent and record IDs.

Write Program

The Write program executes a task tree program that uses eDeveloper HTML Merge forms to create the XML data file. The program returns the XML data as a BLOB.

Write programs are nested. The first Write program runs initially on the root table and then calls all the sub-programs.

Generating the Component

From the **Completing the XCG Wizard** screen, shown in Figure 17-10, you can click **Finish** to generate the component as an export file representing your component interface selection.

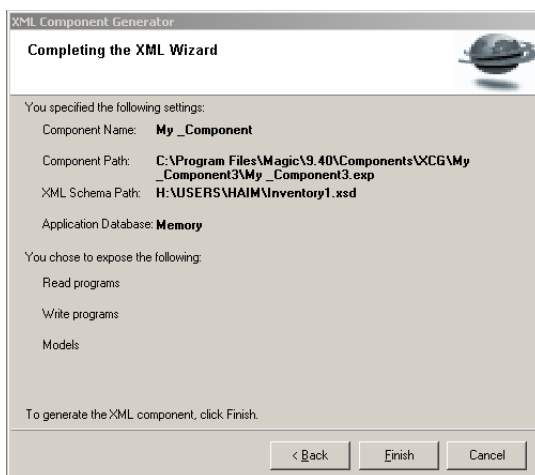


Figure 17-10 Completing the XML Wizard

The file is saved in the path defined in the **Component Path** field.

Text in red indicates an error in one or more of your settings. An example of an error could be two components with the same name or a file-name path that includes a non-existent directory. The component cannot be generated when there are errors.

After you generate the component, the following message appears.

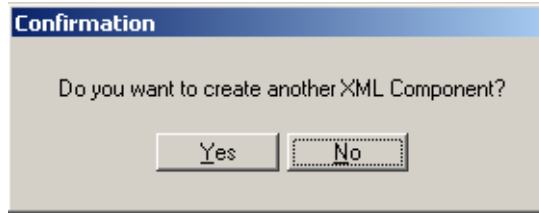


Figure 17-11 Create Another XML Component

When you click **Yes** to create another XML component, you are returned to the **Main Options** screen. If you are importing a component, as specified in the **Component Path** screen, the Wizard returns you to the Toolkit mode without the option of creating another XML component.

Notes:

The MgSchemaParser.dll file is used to parse the XML schema and insert data into the XML Component Generator tables.

The XML Component Generator is not available for an application set for team development.

Output Files

When you generate your component, eDeveloper creates a new folder in the XCG folder. The folder name is made up of the component name and ID. For example:

address1

where address is the component name, and 1 is the component ID.

The files below are created in the directory:

- **MCI File:** This file exposes all the objects that you specified to be exposed when creating the component in the XCG Wizard.
- **TPL File:** A template file to be used by the component when writing the XML file. You must copy the TPL file to your deployment environment, into the same folder where the component will be saved.

Changing the XCG Directory

By default, the XCG folder is under *Magic\Components\XCG*. You can change the location of the XCG directory by entering an entry to the Magic.ini file. Add the **[Interface_Builders]** section to the Magic.ini file. If the folder does not exist, enter the following environment setting:

XCG_Data=[file path name]

This environment setting lets you determine where you want the XCG data to be saved.

Namespace Support

eDeveloper supports schemas that incorporate namespaces. For more information, see Namespaces on page 519 in Chapter 8, Expression Rules.

The XML Component Generator also supports an XSD containing namespaces, as listed below:

- The Main Program will have an alpha variable for each namespace defined in the XSD file. These alpha variables contain the URI of the namespace.
- The Main Program will have a logical variable called **UseNamespaces** that determines whether the XML file, which is read or written, will use namespaces.
- The main write and read programs will have a logical parameter called **UseNamespaces** that determines whether the write or read XML file will use namespaces. The default value is True.

Important: If an XSD file contains namespaces and the export file is imported into an existing application, the host application's Main Program is overwritten.

Connecting Magic to External Applications

18

Magic can receive data from other applications through the following processes: OLE Automation, and Call Operations for a DLL or a 3rd Generation Language.

In this chapter:

- | |
|-------------------------------------|
| • Dynamic Data Exchange |
| • Magic and OLE Automation |
| • Call to a DLL |
| • Call to a 3rd Generation Language |

Dynamic Data Exchange

DDE implementation in Magic uses the DDEML.DLL that is supplied with Windows 95, Windows 98, Windows 2000, and Windows NT. It will not work with earlier versions of Windows. The DDEML.DLL must be stored in the Windows System directory.

Magic's DDE implementation is done through Magic functions. Every function initiates a complete DDE conversation, and terminates the conversation before it returns. This means that the DDE exchange is a "cold" exchange that is always initiated by Magic. Magic does not monitor the changes made by other external applications of the data that Magic receives from the DDE exchange.

The DDE server application must be online for Magic to communicate with it. Accessing a DDE server from Magic without the server online prevents the server from loading.

The DDE server application appears in the background when accessed by a client application. If the Magic DDE functions cause an error in the server application, the server application will respond by issuing an error message, this message is also not in focus and may not be noticed. Often the Magic DDE operation fails simply because the time-out limit has been exceeded.

The Magic DDE functions are synchronous. This means that Magic will wait to receive the result of the DDE operation before it continues processing.

The DDE functions are not portable. They can be used only under Windows. If you attempt to execute DDE functions under a different operating system, the following occurs:

- DDEGet returns an empty strings
- DDEPoke and DDEExec return "False"
- DDERR displays return code '15'. This return code is displayed only when a DDE function was executed on an operating system other than Windows.

Functions

DDEBegin	<p>DDE Begin Creates a session.</p> <p>Syntax: DDEBegin (<i>service, topic</i>)</p> <p>Parameters: <i>service</i>: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord, for MS Word for Windows, or Excel, for MS Excel.</p> <p> <i>topic</i>: String, depending on the service.</p> <p>Return Value: TRUE if the DDE server is already connected or a new connection has been established, or FALSE for failure initializing the DDE or connecting to the DDE.</p> <p>Example: DDEBegin('Excel','c:\docs\budget.xls') Initiates a DDE session control by the User, independent of the DDE process. The DDE session remains open until the user selects the DDEEnd function.</p> <p>See also: DDEEnd, DDEGet, DDEPoke, DDERR, DDEExec</p>
DDEEnd	<p>DDE End Terminates a session.</p> <p>Syntax: DDEND (<i>service, topic</i>)</p> <p>Parameters: <i>service</i>: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord, for MS Word for Windows, or Excel, for MS Excel.</p> <p> <i>topic</i>: String, depending on the service.</p> <p>Return Value: TRUE for successful completion, or FALSE for failure if the service and topic were not started by the DDEBegin function.</p> <p>Example: DDEEnd('Excel','c:\docs\budget.xls')</p> <p>See also: DDEBegin, DDEGet, DDEPoke, DDERR, DDEExec</p>

DDEGet

DDE Get

Get a string of characters from a DDE server.

DDEGet returns a string value of size *length* from the specified DDE server. Each DDE server application provides access to its services via a combination of the three identifiers: *service*, *topic*, and *item*. Taken together, the three identifiers provide a unique identification of the service.

Syntax: DDEGet (*service*,*topic*,*item*,*len*)

Parameters: *service*. The main identifier of the DDE service. Usually this is an application name such as WinWord for MS Word for Windows, or Excel for MS Excel.

topic. The area within the server application with which you want to exchange information. A topic may be a document name in MS Word for Windows or a spreadsheet in MS Excel. Server applications usually provide a system topic as standard practice. The system topic provides information about the application and topics that may be accessed by DDE. The system topic services can be requested through the item parameter. For example, the MS Excel system topic has a system item that returns the items that are available in the system topic. The format of the information returned by the system topic depends on the server application.

item. Further defines the exact data item for the exchange. Together with service and topic it points to a unique item. For example, a paragraph in a Word for Windows document may be read by DDE if the topic specifies the document name, and the item points to a bookmark in that document that marks the required paragraph.

len. The maximum length of the information that will be returned by the function. If the information returned is less than the maximum length, its trailer will be padded with blanks.

Returns: A character string of size length. If the DDEGet failed, the string will be empty. The string returned is provided by the DDE server application according to the *service*, *topic*, and *item* parameters of the function.

Examples: DDEGet('WinWord', 'c:\docs\ddetest.doc', 'toMagic',2000)
reads a paragraph from an MS Word for Windows document that is marked with a bookmark in Word where 'c:\docs\ddetest.doc' is the document name; 'toMagic' is the bookmark name, and 2000 is the maximum size in bytes of the paragraph that will be returned by the function.
Note that the single quotes are required on the first three columns, all string columns.

DDEGet('Excel','c:\docs\budget.xls','R19C1:R22C7', 2000)
reads a range of cells from an MS Excel spreadsheet where: 'c:\docs\budget.doc' is the spreadsheet name, 'R19C1:R22C7' is the cell range, and 2000 is the maximum size in bytes that will be returned by the function.

Magic requires the single quotes shown in these examples to identify columns as strings.

See also: DDEBegin, DDEEnd, DDEPoke, DDERR, DDExec

DDEPoke

DDE Poke
DDEPoke transfers a string from Magic to the DDE server specified by the function's parameters. Each DDE server application provides access to its services via a combination of three identifiers: *service*, *topic*, and *item*. Taken together, the three identifiers provide a unique identification of the service. The string transferred by Magic will be inserted to the server application at the location identified by *service*, *topic*, and *item*.

Syntax: DDEPoke (*service*,*topic*,*item*,*data*)

Parameters: *service*: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord for MS Word for Windows, or Excel for MS Excel.

topic: Provides a definition of the area within the server application with which you wish to exchange information. A topic may be a document name in MS Word for Windows or a spreadsheet in MS Excel.

item: Further defines the exact data item for the exchange. Together with service and topic it points to a unique item.

For example, a paragraph in a Word for Windows document may be inserted by DDE if the topic specifies the document name, and the item points to an empty bookmark in that document that marks the required paragraph.

data: The data string that will be passed by the function to the server application.

Return Value: TRUE for successful completion, or
FALSE for failure to poke the data to the server application.

NOTE: The information transferred to the server application may have to follow some formatting rules dictated by the server application. Refer to the server application documentation for details.

Examples: DDEPoke ('WinWord', 'c:\docs\ddetest.doc', 'fromMagic', 'This paragraph was planted by Magic using DDE')
Will write a paragraph to an MS Word for Windows document, where the paragraph has been marked with a bookmark in Word, and where 'c:\docs\ddetest.doc' is the document name, 'fromMagic' is the bookmark name, and 'This paragraph was planted by Magic using DDE' is the data that will be transferred by the function to the server application.

DDEPoke ('Excel', 'c:\docs\budget.xls', 'R1C1', '100')
Will write the number 100 into an MS Excel spreadsheet cell, where: 'c:\docs\budget.doc' is the spreadsheet name, and 'R1C1' is the first cell of the spreadsheet that will receive the information passed by the function.

Magic requires the single quotes shown in this example, to identify columns as strings.

See also: DDEBegin, DDEEnd, DDEGet, DDERR, DDEExec

DDERR

DDE Error

DDERR retrieves the last error that occurred during a Magic DDE conversation. A subsequent call to DDERR clears the previously reported error code and resets the return value. This function retrieves useful information for debugging purposes.

Syntax: . DDERR ()

Parameters: None

Return Value: A numeric value ranging between 0 and 15, with the following meanings:

- 0 No error in the last DDE operation, or a reset return value if this is the second consecutive call to the function
- 1 Failure to initialize the DDEML system
- 2 Failure to connect to the server (could be a wrong or missing *service* parameter in the preceding DDE call)
- 3 Server was busy during the DDE call and could not service the call
- 4 Server did not process the DDE service required (could be a wrong parameter, an invalid combination of parameters, or an invalid required service combination)
- 5 The last DDEGet failed (server could not provide the required *item*)
- 6 No *item* was specified on a GET or POKE request
- 7 No *command* was specified on an EXEC request
- 14 Unknown type of error
- 15 Attempt to execute a DDE function on an operating system other than Windows

See also: DDEBegin, DDEEnd, DDEGet, DDEExec, DDEPoke

DDEExec

DDE Execute

DDEExec transfers a command string from Magic to the DDE server specified by the function's parameters. Each DDE server application provides access to its services via a combination of three identifiers: *service*, *topic*, and *item*. Taken together, the three identifiers provide a unique identification of the service. The command string transferred by Magic will be transferred to the server application, which in turn will try to

execute it. The command string must follow strict DDE format rules. The command contents must be a valid server application's command. If the command contents are not a valid server application command, the server application will fail.

Syntax: DDEExec (service,topic,item,command)

Parameters: *service*: Provides the main identifier of the DDE service. Usually this is an application name such as WinWord, for MS Word for Windows, or Excel, for MS Excel.

topic: when used with DDEExec, the topic will usually be 'System', as the server application, represented by the System topic, rather than a data object within the server application, is responsible for the exchange.

item: Further defines the exact data item for the exchange. When used with DDEExec, it will usually remain empty.

command: The command string that will be passed by the function to the server application that will then execute it. DDE commands must be contained within square brackets [].

Several commands may be included within one command string, each in its own brackets, separated by blanks.

For example:

[command1]

[command2(parameter1)]

[command3(parameter1, parameter2, parameter3)]

[command1] [command3(parameter1, parameter2, parameter3)]

The command string must contain a valid server application command. Refer to the server application documentation for details about command syntax.

Return Value: TRUE for successful completion, or FALSE for failure to execute the command at the server application.

Examples: DDEExec ('WinWord', 'System', '', '[FileOpen "c:\docs\ddetest.doc"]')
Opens an MS Word for Windows document where:
'[FileOpen "c:\docs\ddetest.doc"]'

is the command transferred by the function to the server application for execution.

```
DDExec ('Excel', 'System', "",  
[run("MACROS.XLM!FormatCells")])  
Executes the MS Excel macro (FormatCells).
```

```
DDExec ('Excel','System','',  
[Open("c:\docs\test.xls")])  
Executes the MS Excel system.
```

Magic requires the single quotes shown in these examples, to identify columns as strings.
MS Excel requires the double quotes shown in these examples.

See also: DDEBegin, DDEEnd, DDEGet, DDEPoke, DDERR

Magic & OLE Automation

OLE Automation is the process by which Magic controls documents or other objects supported by a target application. OLE Automation in Magic is implemented as a set of functions. Automation calls can be made to new objects and objects contained in an OLE control. The OLE Automation mechanism is implemented by mg_ocx.dll.

Implementing OLE Automation

Follow these steps to implement OLE automation:

1. Define an OLE object in your Magic application by placing an OLE control on a GUI form. (Optional)
2. Establish a reference to the object by defining a Call UDP operation that invokes the ObjectLoad function. Zoom on the Prm setting to define the parameters of the function. If you have placed an OLE control, refer to the control name. Otherwise, establish the reference by specifying the object's class name.
3. Specify the properties and methods of the object by defining Call UDP operations to the PropSet, PropGet, and MethodCall functions. Zoom on the Prm setting to define the parameters of the functions.
4. Release the object and any child objects it may have by calling the ObjectRelease function.

Parameter Type String

When you call an OLE Automation function that sends or receives parameters, you define a Parameter Type string that describes the parameters that will be passed to the automation object. This string precedes the required parameters, specifying their type and whether or not they exist. The string is made up of a character for each parameter in the relevant method or property.

The characters are:

- 1 - Char
- 2 - Short
- 4 - Long
- F - Float
- 8 - Double
- D - Pointer to Double

E - Pointer to Float

L - Pointer to Long

A - Pointer to a Null terminated string

O - (zero) Void (return value only)

I - Image passed as HBITMAP

M - Missing optional parameter

O - Object reference. A numeric Long containing the object reference returned by the ObjectLoad function or returned as a property or parameter to a called method.

o - Object reference passed by reference. A pointer to Long.

Do not confuse the Parameter Type string with the Argument Type string passed as the first parameter in the Call UDP operation using a common DLL.

An Argument Type string has the following syntax:

[parametertype] [parametertype] [parametertype]

where you enter the number of arguments relevant for the specific Call UDP operation.

OLE Automation Functions

ObjectLoad The ObjectLoad function establishes a reference to an OLE object.

Syntax: A Call UDP operation with the following expression:
 `@mg_ocx.objectLoad'

Parameters: *Argument Type:* `AAL4' (required for calling a common DLL)

Control Name: The name of the OLE control defined in your Magic application. (If no OLE BLOB field and control are used, a blank string should be specified.)

Class Name: If you have not defined an OLE control, load a new OLE object by specifying the class name of the new OLE object. For example, to load a new Excel object, define the

Class Name to be `Excel.Sheet.8`

Object Instance: A 10 digit numeric variable that will receive the object instance number. *Return Code*: A numeric variable that will receive the function's return code. For more details, refer to the section on Return Codes.

Returns. A reference to the OLE object as the *Object Instance* variable and the *Return Code*.

ObjectRelease The ObjectRelease function releases a loaded OLE object.

Syntax: A Call UDP operation with the following expression:
`@mg_ocx.objectRelease`

Parameters: *Argument Type*: `L4` (required for the calling a common DLL)

Object Instance: A numeric variable or expression containing the object instance number.

Return Code: A numeric variable that will receive the function's return code. For more details, refer to the section on Return Codes.

Returns: The function's *Return Code*.

Note: After the OLE object is released, the *Object Instance* variable is set to zero.

PropGet The PropGet function gets a value from a property in an OLE object, which can be placed in a specified Magic variable.

Syntax: A Call UDP operation with the expression
`@mg_ocx.PropGet`

Parameters: *Argument Type*: `4AA[parmtype][parmtype][parmtype]4` (required for calling a common DLL).

Object Instance: A numeric variable or expression containing the object instance number.

Property Name: A variable or expression containing the property name.

Parameter Types: A variable or an expression containing a

string describing the property's parameters. Optional parameters, which are not passed, are represented by the `M' character.

Parameters: The parameters that are sent to and received from the property.

Return Code: A numeric variable that will receive the function's return code. For more details, refer to the section on Return Codes.

Returns: The function's *Return Code*.

Example 1: To receive the instance of a worksheet from an Excel OLE instance,

1. Load the OLE object and receive its instance as described in the 'objectLoad' section.
2. Define a Call UDP operation invoking PropGet.
3. Zoom to the Parameter list and define the following parameters:

Argument Type string – `4AAL4'

Instance of the OLE - as previously obtained. (4)

Property Name – `ActiveSheet' (A)

Parameter Type string – `o' where `o' is the returned object (A)

Parameter: the variable to receive the cell range instance. (L)

Return Code: the return code variable. (4)

The result of this operation will be the instance of the active sheet of the Excel application that can be used to refer to its data.

Example 2: To receive the instance of a cell or a collection of cells in an Excel Worksheet,

1. Load the OLE object and receive its instance as described in the 'objectLoad' section.
2. Get the instance of the active sheet in the OLE Instance as described in Example 1.
3. Define a Call UDP operation invoking PropGet.
4. Zoom to the Parameter list and define the following parameters.:

Argument Type string – `4AALA4`

Instance of the worksheet - as previously obtained. (4)

Property Name – `Range` (A)

Parameter Type string – `oA` where `o` is the returned object and A is the string describing the range of the cells. (A)

Parameter1: the variable to receive the cells range instance. (L)

Parameter2: the variable or the expression describing the range of cells (i.e. `A1:A5`) (A)

Return Code: return code variable. (4)

5. The result of this operation will be the instance of the range of cells that can be used to refer to its properties and methods.

Note: After obtaining an instance of a child object, such as a worksheet or cell, release it using the OBJECTRELEASE function.

PropSet

The PropSet function sets the value of a known property of a selected OLE object.

Syntax: A Call UDP operation with the following expression:
`@mg_ocx.PropSet`

Parameters: *ArgumentType*: `4AA[parmtype][parmtype][parmtype]4`
(required for calling a common DLL)

Object Instance: A numeric variable or expression containing the object instance number.

Property Name: A variable or expression containing the property name.

Parameter Types: A variable or an expression containing a string describing the property's parameters. Optional parameters, which are not passed, are represented by the `M` character.

Parameters: The parameters that are to be set.

Return Code: A numeric variable that will receive the function's return code. For more details, refer to the section on Return Codes.

Returns: The function's Return Code.

Example: To set the value of a cell or a collection of cells in an Excel

Worksheet,

1. Load the OLE object and receive its instance as described in the 'objectLoad' section.

2. Get the instance of the active sheet in the OLE Instance, as described in Example 1 of the PropGet function.

3. Get the instance of the range of cells in that worksheet, as described in Example 2 of the PropGet function.

4. Define a Call UDP operation invoking PropSet.

5. Zoom to the Parameter list and define the following parameters:

Argument Type string – `4AA44'

Instance of the range - as previously obtained.

Property Name – `Value'

Parameter Type string – the single character `4' for the given value.

Parameter - The variable or expression containing the value to be set.

Return Code - return code variable.

MethodCall

The MethodCall function invokes a method in an OLE Object.

The first parameter described in the Parameter Type string is a variable that receives the return value of the method as either void or a pointer. If the return value is void, no variable should be specified in the Parameter Type string. The remainder of the parameters of the method follow as described in the Parameter Type string. Optional parameters can be omitted by indicating the letter 'M' in the relevant position of the string.

Syntax: A Call UDP operation with the following expression:
`@mg_ocx.PropGet'

Parameters: *Argument Type:* '4A[parmttype][parmttype][parmttype]4' (required for calling a common DLL) *Object Instance:* A numeric variable or expression containing the object instance number.

Property Name: A variable or expression containing the property name.

Parameter Types: A variable or an expression containing a

string describing the property's parameters. Optional parameters, which are not passed, are represented by the `M' character.

Parameters: The required parameters to be received and sent.

Return Code: A numeric variable that will receive the function's return code. For more details, refer to the section on Return Codes.

Returns: The function's Return Code.

Example: To print out the second page of the worksheet,

1. Load the OLE object and receive its instance as described in the 'objectLoad' section.
2. Get the instance of the active sheet in the OLE Instance as described in Example 1 of the PropGet function.
3. Define a Call UDP operation invoking the MethodCall function.
4. Zoom to the Parameter list and define the following parameters:

Argument Type string – `4AA2224'

Instance of the worksheet - (previously obtained).

Method name – `PrintOut'

Parameter Type string – `0222MMMM' made up of 8 characters where:

`0' – void for no returned value from the function,

`222' for three numeric values to pass (from page, to page, and number of copies)

`MMMM' for the following four missing optional parameters.

Parameter1 - A variable or expression containing the number of the first page to be printed.

Parameter2 - A variable or expression containing the number of the last page to be printed.

Parameter3 - A variable or expression containing the number of the copies to be printed.

Return Code:

The Return Code variable.

Each procedure of the mg_ocx.dll may return different error codes depending on the success or failure of its execution.

The following table interprets the codes:

Code	Symbol	Description
0	Ok	Success
1	E_BADPARAMCOUNT	Wrong number of parameters
3	E_INTERNAL_ERROR	
4	E_MEMBERNOTFOUND	Either the property or the method does not exist for the given object.
5	E_OVERFLOW	
6	E_TYPEMISMATCH	The wrong type of data was sent.
8	E_PARAMNOTOPTIONAL	A required parameter was missing
9	E_INTERNAL_ERROR	
10	E_NOOBJECT	No instance of an acquired object was given.
11	E_CTRLNOTFOUND	The control name given in the objectLoad function was not found.
12	E_BADPARAM	Neither a control name nor a class name was given in the objectLoad function.

Call to a DLL

The CalIDLL, CalIDLLF, and CalIDLLS functions let you call a DLL from an external application. You can also call a program or function from an DLL of an external application. The CalIDLL function enables a dynamic and direct call to a DLL from an external application within in eDeveloper. The CalIDLLF function calls to an external Fastcall function. The CalIDLLS function calls to an external Stdcall function. The syntax parameters of these functions are described below.

- modulename.functionname - the module and function names from the DLL.
- argument type string - a string in which each character represents the type of argument. The last character represents the type of the return value of the function. The Argument types are:

- 1 - Char
- 2 - Short
- 4 - Long
- F - Float
- 8 - Double
- D - Double pointer
- E - Float pointer
- L - Long pointer
- A - Null terminated string pointer
- V - Void pointer
- 0 - Void

- Arg1,Arg2,... - The function arguments from the DLL.

Call to a 3rd Generation Language

You can call a UDP to invoke the execution of a 3rd generation language program and to also pass parameters by using the UDF, UDFF, and UDFS functions. Magic uses the C language calling convention when calling this program. The call is done in memory as if the user procedure is an internal subroutine of Magic, using the simplest call or jump instruction of the machine.

You can call a UDP for the following uses:

- Implementation of Magic extensions: user-supplied routines that perform specific tasks or functions not supported internally in Magic.
- Utilization within Magic of existing code written outside of Magic.
- Implementation of specialized calculations that have been optimized for speed in the external procedure.

Call UDP Operation Parameters

Only the parameters which are specific to the Call UDP operation appear below. All the other parameters that are common to all the Call types are explained in Chapter 7, Operations.

Call Category

The words UDP must be specified in place of the word Task. Click on the combo box to see the five Call categories and select UDP.

Identification Number of the UDP Expression

The Identification Number of the UDP Expression is required. It is the number of the expression in the Expression Rules repository that at Runtime returns a string with the name of the called program.

Name

After you have selected the expression, the first part of its text appears in the Name parameter for display only.

UDP Functions

The UDP functions are described below.

- UDF - User functions written in the third generation programming languages, like C or Pascal, can be called as functions within Magic expressions (cdecel).
- UDFF - User functions written in third generation programming languages, like C or Pascal, can be called as functions by the fastcall convention.
- UDFS - User functions written in third generation programming languages (like C or Pascal) can be called as functions by the stdcall convention.

Distributed Application Architecture

19

e Developer achieves a high level of interoperability among different computing environments. Interoperability means the ability of eDeveloper applications to operate in multi-database, multi-platform, and multi-network data processing contexts. Using the simple interface common to all eDeveloper applications, every user, from any workstation, can access any type of local or remote database, execute queries, and update the data.

In an environment where multiple computers are connected by Local Area Network (LAN) or Wide Area Network (WAN), eDeveloper's distributed application architectures designate a specific form of distributed processing.

In this chapter:

- | |
|------------------------------------|
| • Enterprise Server General Scheme |
| • Enterprise Server Setup |
| • Supported Middleware |
| • Internet Requesters |
| • Browser Client Applications |
| • Application Partitioning |

The Enterprise Server General Scheme

The Enterprise Server General Scheme, as described in Figure 19-1, allows any supported client to communicate with the eDeveloper enterprise server.

The following figure shows the basic elements required for the communication between the client and the enterprise server.

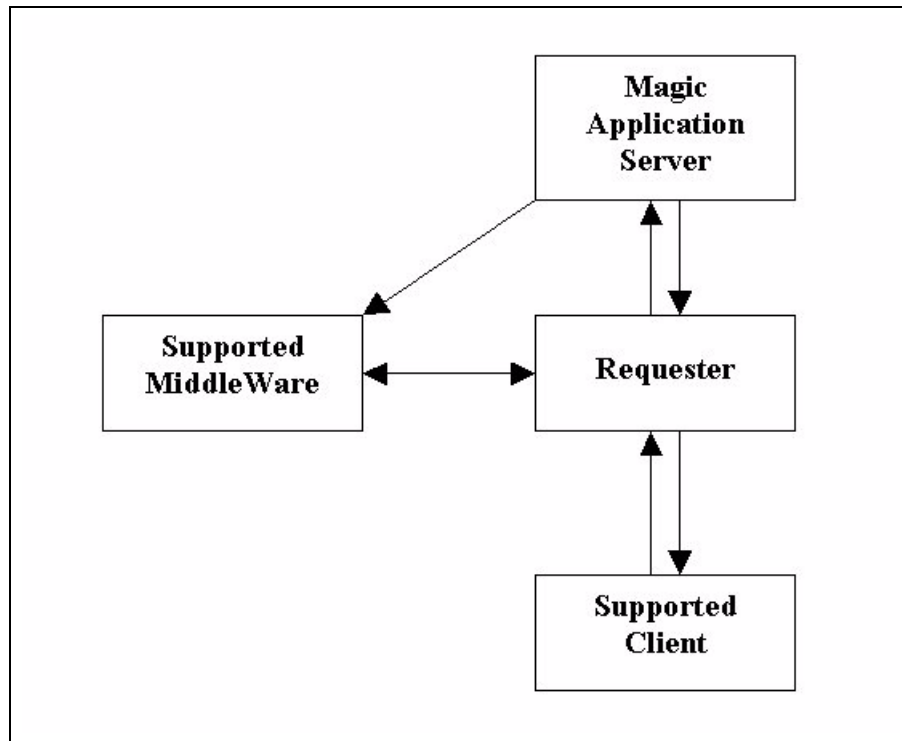


Figure 19-1 Communication Between the Client and the Enterprise Server

The following are the components of the enterprise server configuration:

- eDeveloper enterprise server – An eDeveloper multi-threaded runtime engine connected to a supported middleware module may act as an enterprise server for any type of supported requester.

- Requester – A client may issue a request to an enterprise server only through a supported requester. Each client has its own requester. There are three major requester types:
 - eDeveloper requester – The eDeveloper engine has a requester module as an integral part of the engine.
 - Internet requester – eDeveloper provides three types of Internet requesters – CGI, ISAPI and NSAPI.
 - Command line requester – eDeveloper provides a command line requester that can be executed from an OS Shell.
- Middleware – Software modules can be used to direct a request from a requester to an eDeveloper server and back. eDeveloper supports the following middleware:
 - Magic Request Broker – eDeveloper provides a proprietary middleware module.

Uses of Distributed Application Architecture

Application Partitioning

The term *application partitioning* is used to describe the process of developing applications that distribute the application logic among two or more computers in a network. In the simplest case, the application can run on a single PC, as a client, and send task requests for execution to a server. In more advanced cases, the application logic can be distributed among several servers. eDeveloper also allows the following network configurations:

- eDeveloper to eDeveloper – the eDeveloper client can pass requests to an eDeveloper enterprise server using any of the middleware types listed above.
- eDeveloper to third party applications – the eDeveloper client can pass requests to a third party application.

Internet/Intranet Applications

eDeveloper provides various ways to create a browser-based application. Using the Internet requester that resides on a Web server, eDeveloper can accept a request from a remote browser, process this request, and return an HTML-based result. eDeveloper can produce needed HTML based content in the following ways:

- A browser task that creates a full browser-based task where the HTML content is enhanced to provide the runtime logic needed to maintain HTML controls with their associated data, such as event trapping and handling.
- A batch task that outputs an HTML form or a frame set interface type.
- A batch task that outputs data merged into an HTML template.
- A batch task that outputs data merged into an HTML template enhanced with Web Online features.

Enterprise Server Setup

Runtime Engine Behavior

The eDeveloper Runtime Engine executes requests for remote services. The runtime engine is an instance of the same eDeveloper Runtime executable that is used for running an application locally. The eDeveloper Request Gateway is a software layer that enables the runtime engine to communicate with the eDeveloper Request Broker and the Requester Client.

The eDeveloper Runtime Engine can handle a request from only one middleware agent. The requester client, however, can communicate with many middleware agents, and the request server can communicate with many runtime engines.

The runtime engine's broker address is stored in the Magic. ini file. When the runtime engine starts up, it notifies its request broker that it is available. The message also includes the list of applications that the runtime engine

supports. The runtime engine then proceeds to listen on a known port, which is also specified in its Magic.ini file. When a request is received and the runtime engine is idle, the runtime engine will open the requested application.

The runtime engine enters the user name and password, specified in the request, into the request log. It will then run the requested program, and send the execution results to the caller.

During the execution of the request, the runtime engine sends "I'm Alive" messages to either the request broker or the requester client, at intervals of ServerTimeout seconds. A ServerTimeout second is a value that the Request Client sends along with the request.

If the request is not valid, or if the runtime engine is busy, an appropriate message is sent to the request source.

The eDeveloper enterprise server is constructed in the same way regardless of the type of client. The only difference in the enterprise server for each constellation is the middleware module.

Loading a Middleware Gateway

The following settings are needed to connect the eDeveloper engine to the middleware gateway as the middleware gateway is loaded to an enterprise server.

1. Load the required middleware gateway.

You should unmark the required gateway in the [MAGIC_MESSAGING_GATEWAYS] section in the MGREQ.INI file located in the working directory of your eDeveloper engine.

Magic Request Broker – The eDeveloper engine can always be connected to the Magic Request Broker. No specific settings need to be made to the MGREQ.INI

Note that although several gateways can be loaded, a single engine can be connected as a server to only one middleware module.

2. Define the server settings in the server table.

Refer to the relevant middleware section for exact details.

3. Assign the server setting to be the messaging server for this engine (*Settings/Environment/Enterprise Server*).
4. Define the engine to load as an enterprise server by selecting Yes for the Environment setting: Activate as Enterprise Server.
5. Activate the middleware agent.
6. Activate the eDeveloper executable to run as a full functioning enterprise server.

For more information, refer to the relevant middleware and client sections in this chapter.

Supported Middleware

The following middleware gateways are supported by eDeveloper:

- Magic Request Broker (MRB)

Magic Request Broker - MRB

The Magic Request Broker component receives and processes requests from the network. Requests are calls for remote program execution that include

parameters and options. A request is eventually sent to an eDeveloper engine for execution. A diagram of the MRB process is shown in Figure 19-2 .

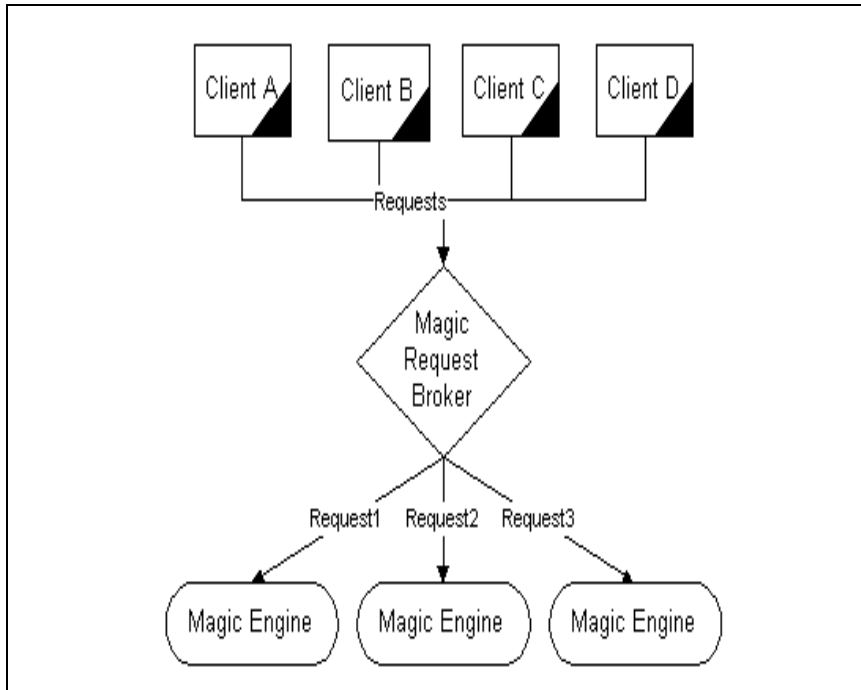


Figure 19-2 The Magic Request Broker Process

The main functions of the Magic Request Broker are:

- Queuing client requests
- Allocating available runtime engines for requests
- Logging all operations
- Maintaining and distributing status of all submitted requests
- Activating programs in NOWAIT mode

Prerequisites

The MRB requires a WINSOCK compatible TCP/IP stack installed under the Windows operating system.

Installation and Configuration

All the Magic Request Broker files should reside in the same directory:

- MGRQMRB.EXE is the broker executable. The MRB can run under Windows 95/98 or Windows NT/2000. On Windows NT/2000, the broker can be run as service.

Run MGRQMRB.EXE from the command line or a shortcut to start the MRB operation. An icon on the Windows task bar indicates that the MRB is running. If the MRB is started as a service, the task bar icon does not appear.

- MGRQGNRC.DLL is the generic messaging layer that contains the support libraries.
- MGRB.INI is the initialization and settings file. The MGRB.INI file is read during the MRB initialization. Note that the MGRB.INI is not necessary for the successful installation and configuration of the Magic Request Broker. If the MGRB.INI is not found, the MRB will use the defaults for each initialization setting.

The following parameters should be defined in the initialization file:

Keyword	Explanation
BrokerPort	The ports that the MRB listens to and waits for requests from. Syntax: BrokerPort = port number: (default number =3001) Note that the MRB will hold a total of 5 consecutive ports starting with the given port.
EnginesPriority	A list of up to 9 computer names that are used to provide server engines. This list controls the order in which the MRB chooses engines to serve client requests. The Engines Priority list operates sequentially so that the engine for the first computer executes the client command, if available, before the engine of the second computer, and so on.

Keyword	Explanation
Server Timeout	<p data-bbox="568 213 1296 309">The interval, in seconds, within which the broker instructs the engines to send an I-AM-ALIVE message during the execution of asynchronous calls.</p> <p data-bbox="568 352 1329 413">If the value is 0, then the engine will not send the I-AM-ALIVE message to the MRB.</p> <p data-bbox="568 456 1318 552">If the engine crashes or is terminated in an abnormal way during task execution, it is understood by the MRB as a no response from the engine.</p> <p data-bbox="568 595 939 621">Syntax: ServerTimeout = n</p>
PasswordSupervisor	<p data-bbox="568 642 1315 737">An optional value that restricts the user's access to the MRB. If specified, security checks will occur during the following broker operations:</p> <p data-bbox="568 781 1329 911">Clearing or prioritizing any command request in the queue (a user can modify a command request based on the user name and password used when the request was submitted).</p> <p data-bbox="568 954 856 980">Terminating the MRB.</p> <p data-bbox="568 1024 913 1050">Loading new executables.</p> <p data-bbox="568 1093 1282 1154">If PasswordSupervisor is not specified, security validations for user name and password are inactive.</p>

Keyword	Explanation
PasswordQuery	<p>An optional value that can be used to restrict access to the MRB. If specified, a password is checked before allowing a user to query a request of another user. Any user can access the Query Queue and Log to view their own requests, based on the user name and password entered or in effect when the request was submitted, without any security restrictions. The MRB accepts the user name and password of the service, the user name and logon password to the eDeveloper application, or the user name and password stored in the MGREQ.ini file (for cmdl/internet only).</p> <p>If PasswordQuery is not specified, security validations for user name and password are inactive.</p>
AutoLoad	<p>This setting is used to let the MRB load additional enterprise server engines when incoming requests cannot find an available enterprise server.</p> <p>If set, the MRB will load an additional enterprise server engine as specified to serve a new request that had no available enterprise server engines (either because all are busy or none are registered in the MRB).</p> <p>Syntax: AutoLoad=[Executable name], [Number]</p> <p>Example: AutoLoad = Background,2</p> <p>Executable name: A name of a define executable, in either the [MRB EXECUTABLES LIST] section or in the [MRB REMOTE EXECUTABLES LIST] section.</p> <p>Number: The total number of engines to be loaded for the specified executable name.</p>
Reload	<p>When this setting is set to Yes, the MRB reloads any instance of an eDeveloper enterprise server that was not terminated by the MRB.</p> <p>Example: Reload=Y</p>

Keyword	Explanation
Log	<p data-bbox="568 215 1329 413">A reference to a file that logs high level MRB activities such as initialization, receiving requests, locating enterprise servers, passing enterprise server information to the requester, and so on. This log can be used to check if a certain request was accepted by the MRB and how it was handled.</p> <p data-bbox="568 440 1258 501">Syntax: LOG= ([Log File Path and Name] [Sync] [Level])</p> <p data-bbox="568 529 1065 557">Example: LOG=C:/temp/mrb.log Y C</p> <p data-bbox="568 585 1308 645">Log File Path and Name: Any valid name that does not include spaces.</p> <p data-bbox="568 673 639 701">Sync:</p> <p data-bbox="568 729 1329 855">Y - The log file opens and closes for each line. This allows for the deletion of the file while the components are still loaded in memory. It also allows the log file to be shared among several modules.</p> <p data-bbox="568 883 1272 944">N - The log file opens and closes only during the component initialization and termination operations.</p> <p data-bbox="568 972 1322 1067">F - Each line designated to be printed is sent to the Log file. The Log file only opens and closes during the component initialization and termination operations.</p> <p data-bbox="568 1095 644 1123">Level:</p> <p data-bbox="568 1150 1133 1178">C - Customer level. The lowest log status.</p> <p data-bbox="568 1206 1179 1234">S - Support level. An intermediate log status.</p> <p data-bbox="568 1262 1072 1289">R - R&D level. The highest log status.</p> <p data-bbox="568 1317 1319 1404">To let the MRB log low level activities(such as connect, send, receive, and so on), you can define the log in the MGREQ.INI file located in the MRB directory.</p>

Keyword	Explanation
ActivityLog	<p>The ActivityLog setting lets you specify the Magic Request Broker log file name and path,</p> <p>Example of the ActivityLog setting is:</p> <ul style="list-style-type: none"> • ActivityLog=logs\broker.log <p>If the Activity log is not set it is created by default as mrb_event.log.</p>
[MRB EXECUTABLES LIST]	<p>Optional section containing a list of command line executables or shell commands that can be activated from a remote requester, or upon MRB initialization.</p> <p>Syntax:</p> <p>EXE_ENTRY_NAME=<command>[<work dir>],[<username>],[<password>],[<number of times to perform upon broker initialization>]</p>
[MRB REMOTE EXECUTABLES LIST]	<p>Optional section for setting up a multi-computer environment with engines running on computers independent of the MRB.</p> <p>An instance of the MRB, functioning as a loader, must be active for each computer with engines running on that broker. The local MRB automatically connects to the remote MRB, specified in the exe entry of this section, instructs it to load an executable from its list of executables. The PasswordAdmin must be used if the remote broker has its own password (if not specified, use the password for the local MRB).</p>

Keyword	Explanation
[MRB ENGINE CLIENTS MAP]	<p>The purpose of the MRB Engine Clients Map is to let you control the engine that receives the request. For example, when several developers are working on the same MCF file, a developer can check out an object and command the MRB to assign its own engine.</p> <p>Note: During the Remote Call execution (when the engine is mapped to this section), the Request Gateway utilizes the logon name and password that are sent with the request. If a user is not logged in to the engine, it is necessary to specify the user name and password from either the Mgreg.ini file for Internet and Command Line requesters, or in the Service repository for the eDeveloper 9 client to view the version of the MCF related to that logon.</p>
Filters	<p>You can use this field to specify how threads are allocated for requests of different filter types.</p> <p>The example below shows you how an enterprise server with a license of 20 thread can be designated to service different types of requests:</p> <ul style="list-style-type: none"> • HTTP End Users - 2 threads • SOAP_BT - 6 threads • COM Primary - 4 threads • Requests without filter - 8 threads <p>The filter string entered for the above example would be:</p> <pre>HTTP_endusers:10% SOAP_BT:30% COM_primary:20%</pre> <p>If the accumulated percentage is greater than 100%, the last filter entry that exceeds the accumulated percentage of 100% will be adjusted to be equal to 100% and all subsequent filters will be discarded.</p>

Keyword	Explanation
AllowReserve	<p>When the accumulated percentage of designated filters is less than 100%, you can specify Yes (default) to reserve the remaining percentage for other filters.</p> <p>When AllowReserve = No, only requests without a filter can use the remaining percentage.</p>

Magic Request Broker Behavior

The Magic Request Broker Queue

The Magic Request Broker (MRB) manages the request queue, which is based on priorities from 0 to 9, where 0 is the lowest priority. Each item in the queue contains the following information:

- Request ID
- Priority
- Execution status
- Application name
- Original request packet
- Error codes

The MRB manages a list of associated Runtime engines containing the following information for each Runtime engine:

- Address
- Status
- Current application
- Available applications

Automatic Reload

Following an abnormal termination, the Magic Request Broker automatically reloads through an administrative process that monitors the MRB.

Automatic Termination of the Enterprise Server

If the Magic Request Broker terminates and restarts, normally or abnormally, the enterprise server connected to the broker session will terminate after 60 seconds or less. This applies only to an enterprise server running in the background mode.

Load Balancing Enhancements

The Magic Request Broker load balancing mechanism ensures that requests are directed to the engines that have the highest priority, the best performance, and the least number of active contexts.

Broker Error Messages

This information below is used to understand the reasons behind broker error messages and suggestions on how the error can be resolved.

102 ERR_CNCT_REFUSED_MRB	<p>The connection to the broker was refused.</p> <p>Verify that the broker is listening to the port on the specified machine. Refer to its Mgrb.ini file.</p>
103 ERR_APP_NOT_FOUND	<p>The required application does not appear in the list of applications registered to the selected broker. Use the Monitor Application utility to view the list of applications.</p>

104 ERR_APP_IN_USE	<p>All enterprise servers supporting the application are busy. This error also can appear as a result of the broker timeout. Check the Timeout value in the Mgreg.ini file.</p> <p>Use the Monitor Application utility to view the status of the enterprise servers.</p>
105 ERR_MRB_NOT_RSPND	<p>The broker did not return an acknowledgement message or any response to the requester.</p> <p>For remote calls, increase the CommTimeout value in the Mgreg.ini file (the default is 10 seconds). For query and administrative requests, you can increase the Broker timeout value.</p>
106 ERR_RT_NOT_RSPND	<p>The requester was not able to send a request to an enterprise server. You can increase the CommTimeout value in the Mgreg.ini file.</p>
107 ERR_CNCT_RESET	<p>This message appears when:</p> <ul style="list-style-type: none"> • The enterprise server was aborted during the execution of a request. • The connection was reset due to network connection problems.
109 ERR_CNCT_REFUSED_RT	<p>This message appears when there are connection problems between the client requester and the enterprise server.</p> <p>You can view the monitor for the list of enterprise servers. Ping the host name and IP address of the assigned enterprise server.</p>

110 ERR_REQUEST_TIMEOUT	The execution of the task was not completed during the Request Timeout interval.
130 ERR_BAD_MCF	The application cannot be opened.
131 ERR_BAD_PRG	The enterprise server could not find the requested program. Check the program's public name.
132 ERR_BAD_ARGS	Invalid arguments were passed to the program, for example, an incoming SOAP envelope containing an invalid element. You can use a tracer to locate and fix the invalid elements.
133 ERR_ACCESS_DENIED	Access is denied to the application. This error can occur when: <ul style="list-style-type: none"> • An invalid user or password was passed to the enterprise server. • The user had no rights to execute the program.
137 ERR_REQ_REJECTED	The enterprise server cannot execute the request because of a timing problem. This error is usually related to switching between Runtime and Toolkit modes.
138 ERR_RT_ERROR_MSQ	During the execution of a program in an enterprise server, the program was not processed properly, for example, a verify error that aborted execution of the program or any other abort condition.
139 ERR_THREAD_ABORTED	During the execution of a program, the program terminated abnormally.

146 ERR_BIND_HOST_NOT_FOUND	<p>The Local Host entry in the Mgregl.ini file or /LocalHost in the TCP/IP parameters in the Magic.ini file (for example, TCP/IP = 2,30,1500-2000 /LocalHost=myserver) specifies an invalid host name.</p>
147 ERR_CNCT_HOST_NOT_FOUND	<p>Unknown host. This error is similar to ERR-BIND_HOST_NOT_FOUND.</p> <p>A requester cannot connect to an unknown broker or enterprise server. You should check the MessagingServer keyword in the Mgreg.ini file.</p> <p>The broker address must contain the Internet address, such as 88.0.184/2001.</p>
148 ERR_CNCT_CLOSED	<p>A connection was unexpectedly closed.</p>
150 ERR_REFUSED_MRB	<p>An application server could not connect to the broker.</p> <p>Check if the broker has started and that the host name of the broker, MessagingServer keyword in the Magic.ini file, belongs to the correct IP address (ping <mrbbhost>).</p> <p>You can reproduce the problem by setting the Log parameter to Enabled in the Mgreg.ini file (for example, log = reg.log Y R) in the enterprise server directory.</p>

151 ERR_CNCT_RESET_BY_REQ

During the execution of a request, the connection between the requester and the enterprise server was reset. As a result, no output was returned from the enterprise server to the requester.

Check the requester in the client machine and the enterprise server. If possible, reproduce the problem by setting the Log parameter to Enabled in the Mgreg.ini file (for example, log = req log Y R) in the requester and enterprise server directories.

eDeveloper Requesters

Various types of clients can use the services of the eDeveloper enterprise server. Each supported client needs to use the required requester module.

The requester module is specified for each type of client as described below.

Requester Settings

Supported clients that use one of the supplied eDeveloper requester interfaces (such as eDeveloper client, Command line, Browser) can be set in the MGREQ.INI file. This file must reside in the working directory of the requester module.

During the requester initialization, the MGREQ.INI file is read. The sub-directory with the MGREQ.INI file is located in the directory of the General Messaging module (mgrqgnrc94.dll).

The following information is defined in the initialization file:

Keyword	Explanation
Gateway	Describes the middleware that is used by a requester. Gateway = 1 means use The MRB. Syntax: Gateway = <i>number</i>
MessagingServer,	The address of the Middleware agent that receives the requests. Syntax: MessagingServer = <i>node name/port number</i>
AltMessagingServer	You may define an additional address of another middleware of the same type. The requester will refer to this address if the connection to the main Messaging server fails. Syntax: MessagingServer = <i>node name/port number</i>
BrokerTimeout	The maximum time, in seconds, that the request waits for an available engine to be returned by the middleware. If the middleware agent cannot provide an available engine within this time, the requester returns one of the following two error codes: APP-NOT-FOUND - the application is not listed in the MRB's internal database. APP-IN-USE - the application is listed but the supporting engine is busy. If the value is 0, the eDeveloper engine does not execute the request for the MRB. Syntax: BrokerTimeout = <i>n</i>

Keyword	Explanation
KeepAlive	<p>Uses the mgrqgnrc94.dll to set a requester option for each connection opened, and to use OS level settings to control the keep-alive intervals.</p> <p>KeepAlive is only enabled when OS settings are used. System managers must set the OS setting intervals.</p> <p>KeepAlive = Y or N [default=Y].</p>
RequesterTimeout	<p>The maximum time, in seconds, that the requester waits for the completion of a task.</p> <p>If the value is 0, the completed task execution time is unlimited.</p>
Priority	<p>Syntax: RequesterTimeout = n</p> <p>Optional. The default priority for requests that were not assigned a specific queue priority, 0 to 9 with 9 as the highest priority.</p> <p>The priority is used by the MRB only, to assign free engines to un-prioritized requests.</p> <p>Syntax: Priority = n</p>

Keyword	Explanation
CloseWaitTimeout	<p>You can use this keyword to define a mechanism for closing semi-closed connections, i.e. connections that were closed by a peer. You can use the keyword as follows:</p> <p>If you have previously-connected brokers or engines that were shut down and then restarted at the same address, though the module initiating the connection was not shut down, semi-closed connections will exist. You can use the CloseWaitTimeout keyword and define whether the connections are either automatically closed when the next request is received or after a certain defined period of time.</p> <p>Note: This keyword is recommended if you normally your broker and engines are shut down but the Web server is not shut down.</p> <p>You can use these values with the keyword:</p> <ul style="list-style-type: none"> • The duration, in minutes, between periodical closing of all semi-closed connections. For example, if you enter 5, the engine will close any semi-closed connections every five minutes. In addition, each semi-closed connection is closed when encountered by the requester. • -1: If you enter -1 for this keyword, there is no periodical closing and semi-closed connections are only closed as encountered. • You can enter 0 to determine that semi-closed connections are never closed, either periodically or as they are encountered. <p>Note: The default value for this keyword is -1.</p>

Keyword	Explanation
Username, Password	Optional. The default user name and password that is submitted with unidentified requests. Syntax: Username = <i>string</i> Password = <i>string</i>
DefPath, DefName	Optional. A directory and filename default for a requests specified without a filename. Syntax: DefPath = DefName =
DefHtml	For Internet requesters: A filename that can be queried by an Internet requester that can be display if the called program does not return an HTML result.
Http Vars	For Internet requesters: A list of variables that are sent from the web server to the activated task, using different requesters: NSAPI, ISAPI, or CGI. Syntax: httpVars=VarName1, VarName2,...
Appl	Allowed application names separated by a comma. When not specified all applications are allowed. Syntax: Appl=applic 1, test 2, abc,...

Keyword	Explanation
AutoLookBack	Y or N (default = Y) Informs the broker to use the local host IP address, 127.0.0.1, instead of resolving the host name, which is useful when the broker and the application server are on the same computer.

Keyword	Explanation
Log	<p>A reference to a file that logs low level requester activities</p> <p>Syntax: LOG = ([log file path and name] [Sync] [Level])</p> <p>Example: LOG = REQ.log Y S</p> <p><i>log file path and name:</i></p> <p>Any valid name that does not include spaces.</p> <p>Sync:</p> <p>Y - The log file opens and closes for each line. This allows for the deletion of the file while the components are still loaded in memory.</p> <p>And to share the log file with several modules.</p> <p>N - The log file opens and closes only during the component initialization and termination operations.</p> <p>F - Each line to be printed will be sent to the log file.</p> <p>The log file only opens and closes during the component initialization and termination operations.</p> <p>Level:</p> <p>C – Customer level – The lowest level of logging.</p> <p>S – Support level – An intermediate level of logging.</p> <p>R – R&D level – The highest level of logging.</p>

Keyword	Explanation
[MAGIC_MESSAGING_GATEWAYS]	<p>Specify the gateway of the required module to be loaded.</p> <p>Each requester can work with only a single middleware type. Unmark the middleware gateway entry to load the required middleware module.</p> <p>Syntax: MGSVR##=DLL,N,log file name, LogSyncFlush, middleware parameters</p> <p>Example: MGSVR06=MGRQSOAP.dll,N,mg.log,Y</p> <p>## - The number of middleware type:</p> <p><i>DLL</i> - The name of the required gateway dll file. 06 - SOAP Gateway - MGRQSOAP.dll</p> <p><i>log file name</i> - The name of the log file.</p> <p>LogSyncFlush Y - The log file opens and closes for each line. This allows for the deletion of the file while the components are still loaded in memory, and to share the log file with several modules. N - The log file opens and closes only during the component initialization and termination operations.</p> <p>F - Each line to be printed is sent to the log file. The log file only opens and closes during the component initialization and termination operations.</p> <p>Middleware parameters - Any required parameter to be passed to the middleware. If you are using the Magic Request Broker, there is no need to set anything in this section.</p>

Keyword	Explanation
UPLOAD LIMIT	<p>The MaxUploadKB setting determines the maximum size of a file upload: MaxUploadKB=nnnn. The size is defined by kilobytes.</p> <p>If a file larger than the limit is submitted, the Magic requester does not process the request, and returns the following error message to the client:</p> <p>File upload failed: file size exceeds maximum size limitation.</p> <p>The Error code is: -260</p> <p>If the MaxUploadKB parameter is not defined, set to zero, or to a negative value, eDeveloper will upload any file, regardless of size.</p> <p>Uploading large files can create a congestion on the enterprise server.</p>
Handles	<p>Specifies the maximum number of sockets that can be opened by the broker, enterprise server, or requester, depending on the location of the Mgreg.ini file</p> <p>Default = 500</p>
RetryMainTime	<p>Specifies the number of minutes a requester waits before retrying its primary broker. This setting applies only to persistent requesters, such as ISAPI and APACHE.</p> <p>Default = 5</p> <p>Minimum value = 1</p>

Keyword	Explanation
HttpSigVars	<p>Setting requests for execution within a defined context without any client identification means that any user can create a user-defined request with the same context identifier. To avoid this, the browser context requires some or all of the following client-side identification:</p> <ul style="list-style-type: none"> • Client-side IP, such as REMOTE_ADDR • Client-side host name, such as REMOTE_HOST • Client-side user name, such as REMOTE_USER • SSL related information, such as HTTPS, SSL_CLIENT_KEY_SIZE • User-defined cookie variables
Filter	<p>You can use this keyword for the requester to send a filter with each request by entering:</p> <p>[REQUESTER_ENV] Filter = <key name></p> <p>For requests that did not originate from the runtime engine, such as command line or ISAPI, the <key name> from the Mgreg.ini file is sent to the broker with each request.</p> <p>Refer to the MRB Filter on page 1167 for a complete explanation about filters.</p>

Internet Requester

An Internet requester lets the eDeveloper enterprise server respond to requests from a browser client through a web server. A requested eDeveloper program usually produces an HTML-based output that is returned to the browser.

A browser connects to an eDeveloper application by activating the Magic Internet Requester that resides on the web server.

Upon activating the Internet requester, the request is passed to an available runtime engine for processing. The result of the enterprise server task is then returned to the requester and is displayed on the browser client.

How You Can Use eDeveloper on the Internet

You can use eDeveloper to develop your browser-based applications without any prior HTML or Java programming knowledge. eDeveloper lets you design and develop an application for the Internet and Intranets by using five major approaches:

- HTML and Java-based interfaces
- Plain HTML
- Frame Set
- Merge
- XML

These approaches can be employed separately or in combination.

You can use eDeveloper as a tool for creating, editing, and testing your web applications. eDeveloper shows you how these elements will appear on your web browser, and lets you debug as you go along.

Application Development Concepts

Major Components

The major components of an eDeveloper Internet application are:

- Enterprise server
- Web server and related requester
- Web browser

- Middleware agent supported by eDeveloper

Software Requirements

- Supported Middleware

You need to load and run a middleware module that is supported by the eDeveloper enterprise server. For more information, see the Supported Middleware section.

- Web Servers

You must have at least one working web server installed before you can develop any eDeveloper applications for the Internet. An Internet Requester module must be installed on the web server. During a typical eDeveloper installation, the installation program automatically detects the installed web server, and defines and sets the required Internet Requester module.

Setting Up an Internet Requester

eDeveloper provides various Internet requesters for the various types of web servers.

- MGRQISPI.DLL – The Magic Internet Requester that can be used in Microsoft web servers.
- MGRQCGI.EXE – The Magic Internet Requester that can be used to work within any web server. Since this CGI-based requester is common to all web servers, it is slower in performance than the specific requesters mentioned above.
- The Magic Internet requester module requires additional files:
- MGRQGNRC.DLL – The Generic Messaging layer.
- MGRQHTTP.DLL – The HTTP services layer.
- MGREQ.INI – The initialization file.

Microsoft Web Server

If you are working with a Microsoft web server, do the following:

1. Place the MGRQISPI.DLL, MGROGNRC.DLL, MGRQHTTP.DLL, MGREQ.INI files in the web server *Scripts* directory.
2. Set the Scripts directory to be *Executable* and *Readable*.
3. Under the Enterprise Server tab in the Environment dialog set the Http Requester environment setting to the following format:

```
http://your server address/scripts/MGRQISPI.DLL
```

Substitute your actual server address after `//`.

4. Restart your web server.

Note: You can define your own directory with a different web alias, set the directory to *Executable* and *Readable*, and place your files there. The eDeveloper installation process simply creates an alias to the eDeveloper working directory where the required requester files reside.

Other Web Servers

If you are working with a web server other than the Microsoft web server, do the following.

Place MGRQCGI.EXE, MGROGNRC.DLL, MGRQHTTP.DLL, MGREQ.INI in the web server executables directory (e.g. *CGI-BIN*).

Set this directory to Executable and Readable.

Under the Enterprise Server tab in the Environment repository, enter the following format in the Http Requester setting:

```
http://your server address/CGI-BIN/MGRQCGI.EXE
```

Substitute your actual server address after `//`.

Restart your web server.

You can define your own directory with a different web alias, set it to Executable and Readable, and place your files in that directory.

Internet Application Paradigms

The eDeveloper Toolkit provides two major paradigms to construct an Internet application:

- Online oriented paradigm (or Browser Client Application)– the application is defined as a Browser task type, and provides an HTML-based output integrated with a proprietary .NET-based client-side runtime engine. eDeveloper also supports a Java-based, client-side, runtime engine.
- Batch oriented paradigm – the application is defined as a Batch task type. Each program name receives arguments from the browser (including the program to execute) and produces an HTML output with the required interface and data.

SOAP Server Requests

In the MGREQ.ini file, remove the semicolon from the MGSVR06 line to load the **mgrqsoap.dll** file. The mgrqsoap.dll file lets you send and receive Soap server messages from eDeveloper.

Browser Client Applications

eDeveloper Version 9 provides a comprehensive solution for a browser-based application by letting the browser connect to only one interaction at a time. Real life applications are based on numerous interactions. The client displays each interaction as a new page.

Constructing the online interface of an application for a browser demands specific requirements from the client and server. These requirements are explained below.

Creating a Browser Task

You can create a browser client task with full transparent runtime functionality by defining a task as a Browser task type in the Task property sheet.

Creating a Browser Client Program with the Automatic Program Generator (APG)

The Automatic Program Generator (APG) can generate a basic browser client program.

Executing the APG with the Browser Client option produces a fully online browser-based task.

Writing the Logic for the Browser Task

The logic required for optimizing a browser task is created by using the eDeveloper operations and functions.

The operations can be located in all handlers except for the Record Main.

The Record Main is used solely for creating the dataview (that is, Select and Link operations).

The following list of operations and functions shows those that are supported by the browser task.

Browser Operations

Operations supported by the browser task:

- Server operations:
 - Call
 - Input
 - Output
 - Browse

- Exist
- Client operations:
 - Verify
 - Raise Event
- Server and Client operations
 - Block
 - Evaluate
 - Update

Browser Functions

Server-side functions are

ANSI2OEM	ErrDbmsMessage	Line	PPD
Blb2File	ErrMagicName	LMChkIn	Pref
CalIDLL	ErrPosition	LMChkOut	Prog
CalIDLLF	ErrTableName	LMUVStr	ProgIdx
CalIDLLS	EuroCnv	Lock	RightAdd
CallProg	EuroDel	Logical	Rights
ClrCache	EuroGet	MailBoxSet	Rollback
CndRange	EuroSet	MailDisconnect	RqRtCtxs
Counter	EuroUpd	MailError	RqExe
CtxKill	EvalStr	MailFileSave	RqLoad
CtxLstUse	ExpCalc	MailLastRC	RqQueDel
CtxNum	File2Blb	MailMsgBCC	RqQueLst
CtxProg	File2OLE	MailMsgCC	RqQuePri
CtxSize		MailMsgDate	RqReqInf
CtxStat		MailMsgDel	RqReqLst
CurrPosition		MailMsgFile	RqRtApp

DbCache	FlwMtr	MailMsgFiles	RqRtApps
DbCopy	GetLang	MailMsgFrom	RqRtInf
DbDel	GetParam	MailMsgHeader	RqRts
DbDiscnt	GroupAdd	MailMsgIdMailMsgReplyTo	RqRtTrm
DbERR	HTTPGet	MailMsgSubj	RqStat
DbExist	HTTPPost	MailMsgText	RunMode
DbName	INIGet	MailMsgTo	SetLang
DbRecs	INIGetLn	MailSend	SetParam
DbReload	INIPut	MDate	Sys
DbSize	InTrans	MIsTrans	Term
DDEBegin	IOCopy	OEM2ANSI	Text
DDEEnd	IOCurr	RqCtxInf	TransMode
DDEGet	IODel	RqCtxTrm	UDF
DDEPoke	IOExist	RqRtCtxs	UnLock
DDERR	IORen	Owner	User
DDExec	IOSize	Page	UserAdd
Delay	KbGet		VarPic
DiscSrvr	KbPut		Visual
EOF	LIKE		XMLCnt
EOP			XMLExist
ErrDatabaseName			XMLFind
ErrDbmsCode			XMLGet

Client-side functions are:

CallJS	ClipAdd	ClickWY
CallOBJ	ClipRead	HandledCtrl
CHeight	ClipWrite	LastPark
CLeft	CtrlGoto	MAXMagic
ClickCX	CurRow	MINMagic
ClickCY	EditGet	SetCrsr
ClickWX	EditSet	WINBox

Both Client-side and Server-side functions are:

+	CHR	ISDefault	SoundX
-	CMonth	ISNULL	Stat
*	COS	Left	Str
/	CRC	Len	StrToken
MOD	CTop	Level	StrTokenCnt
^	CtrlName	LOG	StrTokenIdx
&	CWidth	Lower	TAN
=	Date	LTrim	TDepth
< >	Day	MAX	THIS
<	DbRound	MID	Time
< =	Del	MIN	Trim
>	DOW	MnuName	TStr
> =	DStr	Minute	TVal
OR	DVal	Month	Upprt
AND	EOM	MStr	VAL
NOT	EOY	MVal	VarAttr
ABS	EXP	NDOW	VarCurr
ACOS	Fill	NMonth	VarCurrN
AddDate	Fix	NULL	VarIndex
AddTime	Flip	RAND	VarInp
ASC	Flow	Range	VarMod
ASIN	Hour	Rep	VarName
ATAN	HStr	RepStr	VarPrev
BOM	HVal	Right	VarSet
BOY	Idle	Round	ViewMod
CASE	IF	RTrim	Year
CDOW	Ins	Second	
ChkDgt	INStr	SIN	

BLOB Support

The browser client supports BLOB fields as part of the browser task dataview. browser client behavior of BLOB fields is described below:

- BLOB fields cannot be displayed using an HTML element.

- BLOB fields with RTF content can be handled locally on the client, using the string manipulation functions.
- You can improve client browser performance by nullifying BLOB fields that are no longer required.
- BLOB variables are supported by the following string manipulation functions:

Del	OEM2ANSI
Flip	Rep
INS	RepStr
INStr	Right
LEFT	RTrip
LEN	StrToken
LOWER	TRIM
LTrim	UPPER
MID	

Explicit Handling for a Browser Task

You can define handlers as System or Internal event handlers.

System event handlers – You can define your own handlers for the various key combinations. However if the Browser has its own functionality for the key combination, this functionality cannot be overridden, even if your handler calls for no propagation.

Internal event handlers – Only the following Internal events are supported by the browser client:

Begin Field	End Table	Next Row
Begin Form	Help	Next Screen
Begin Line	Modify Records	Previous Field
Begin Page	Mouse Out	Previous Page
Begin Row	Mouse Over	Previous Row
Begin Screen	Next Field	Previous Screen
Begin Table	Next Page	Previous Tab
Click	Next Row	Query Records
DbClick	Next Field	View Refresh
Delete Line	Next Page	
End Row		

Help Action Support

The Help Action behavior is described below:

- The eDeveloper Help event can be raised implicitly upon activating the Help action from the browser by pressing **F1**, **CTRL+F1**, or by accessing Help from the Help Menu.
- When the Help event is raised, a new Help window that displays the URL of the defined Help screen opens. The browser's Help window is not displayed.
- When you request Help from an HTML control, eDeveloper displays the Help screen assigned to the HTML control. If no Help screen is assigned to the HTML control, eDeveloper displays the Help screen for the form. If no Help screen is assigned to the form, eDeveloper displays the browser's internal Help screen.

Creating the Browser Task Interface

The interface of a browser task is an HTML based interface. The interface of the browser task is based on an actual HTML file.

The actual editing of the interface is done through an independent HTML authoring tool of your choice.

You can edit this page outside of the eDeveloper Toolkit. Alternatively, you can edit the HTML page by zooming to the browser interface of your browser task. By zooming, eDeveloper opens the defined HTML file using your preferred HTML authoring tool. For more information, See the *Web Authoring Tool* environment setting description in Chapter 2, Settings.

Enhanced Preset Tool Images and Buttons

Preset tool images and buttons provide essential and frequently used actions that are embedded in a browser task.

The preset tool images available are:

- MG_VCR - Divides an image into six parts, where clicking on a part invokes the following eDeveloper internal event:
 - Begin Table
 - Previous Page
 - Previous Record
 - Next Record
 - Next Page
 - End Page
- MG_EDIT - Divides an image into three parts, where clicking on a part invokes the following eDeveloper internal:
 - New Line
 - Cancel
 - Delete

- MG_CANCEL - A single image that invokes the Cancel internal event when clicked.
- MG_EXIT - A single image that invokes the Exit internal event when clicked.

Managing the Interface

The interface consists of HTML elements, defined externally to the eDeveloper Toolkit. Nevertheless, eDeveloper allows you to handle these elements by using the HTML Control table.

The HTML Control table opens when you zoom into the interface entry of the browser task.

In this table you can see the list of participating HTML elements. For each element you may assign data variables or expressions, and define dynamic values for the various properties that are supported by the element.

Initial or fixed values of the common properties of the HTML elements need to be defined through the HTM authoring tool.

Optimized Server/Client Handling

When in the browser task, eDeveloper displays the handling mode of each handler, operation, and function. A character appears on the left side of the line number column representing the handling modes below:

- C - Client-side handling
- S - Server-side handling
- M - Mixed mode handling

If a character does not appear, the entry can be executed on either the client or server.

The handling mode can be disabled by unselecting the Show Handling Info option under the Options menu.

The Show Handling Info state is kept in the machine registry and has the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\MSE\Magic V9.3\9.30\Browser
Information\ShowHandlingInfo

When the Show Handling Info menu entry is disabled, ShowHandlingInfo is set to 0.

When the Show Handling Info menu entry is enabled, ShowHandlingInfo is set to 1.

if ShowHandlingInfo has another value assigned or no value, it is considered enabled.

Client Side

The output produced by eDeveloper and run on the client is enhanced by a .NET-based, client-side modules. These modules provide the following capabilities Java Applet and Java script modules. These Java modules provide the following capabilities:

- Trap events that occur on the page and handle them.
- Re-compute and refresh data and the appearance of the interface elements.
- Execute any eDeveloper function that can be run locally. For example, all number, string, date, and time manipulation functions.
- Update the dataview.
- Support any field level validation definition.
- Maintain a local cache of data greater than what the browser client can display. This cache lets you scroll through new data locally, which decreases the number of interactions necessary between the client and server.
- Client-side operations are:
 - Verify
 - Raise Event

Server Side

The sever side is the most significant component of the browser-based application. The server provides the following capabilities:

- Identify each remote client. The server keeps the context of the task for each client that opens a defined logical task.
- Establish the context, which is an internal description of the exact location of the client in the task and the manipulated data within the task's transaction.
- Translate the data manipulations delivered from the client and passes it to the required databases.
- Re-link a record on the client to retrieve the new linked record.
- Execute any other function that cannot be run locally on the client.

Client/Server Communication

On the initial load, a special message is sent to the server. The message provides the Context ID and the session counter. The session counter is set to 0.

On every consecutive request from the client the session counter is increased. When the server receives a session counter, with a value greater than 0, but lower than the value expected by the server, it returns the following message: `Context is already in use.` Client/Server behavior is as follows:

- When the page is unloaded but the task has not been closed properly (forward and backward, or closing the browser) the client sends to the server the changes in its dataview. The server issues a pending status. The client waits for an answer and then continues with the unload operations.
- The returned Context ID and Session counter must be the same as in the client message. A different value generates the following error:
`Invalid context ID or Session counter.`
- The context can stay in a pending status on the server for a timeout specified by the eDeveloper developer. For more information, see the Post Context Unload Timeout setting.

Browser/Client Events

Client Events

The Client module traps any browser event that may trigger an implicit or explicit eDeveloper handler. For each control in the HTML that is attached to a dataview field, the following browser events can be handled:

- Prefix
- Suffix
- Verification
- Click
- Dblclick
- Mouseover
- Mouseout

A method identifies the implicit or explicit eDeveloper event and executes a required handler. The implicit eDeveloper events are handled first and then the explicit ones.

Server Events

Each eDeveloper event handler is assigned to a list of commands. The client calls to the enterprise server for each of the following operations:

- An expression of the operation, which contains a function that the client cannot evaluate.
- The operation is not one of the following: Update, Call, Block, End Block, Evaluate, or Verify.
- The operation is Update and the target field is part of a link.

Implicit Events

Implicit events are those that are not defined in the Event Handlers table:

- Init task – assigning values to the form controls and to their properties.

- Pre-record Prefix – get the record, compute the init and properties expressions, and display the record.
- Previous/Next Record – if the record is in the client cache, retrieve and display it. Otherwise, retrieve a chunk from the server and display the record.
- Previous/Next Page – if the page is in the client cache, retrieve and display it. Otherwise, retrieve a chunk from the server and display the page.

Calling Programs and Tasks

A program or a task can be called from within an event handler. An event handler that contains a call is executed by the server and returns an HTML string that is displayed in a separate browser window.

A browser task cannot call an online task.

A browser task may call another browser task only by a direct call (that is, it cannot be done through an intermediate batch task). A browser task can call to a batch task that produces regular HTML output.

Closing an Overlaid Browser Task

When a browser task calls a child task (Task B) to replace a previously called child task (Task A), the following sequence of operations occur:

- Task B is opened. The Task Prefix and First Record Prefix are executed, and the result HTML page is returned to the browser client.
- Task A is closed completely. The Record Suffix and Task Suffix are executed. The Exit URL is ignored.
- Task B is displayed instead of Task A.
- If Task A fails to close properly (for example, Verify/Error in the Record Suffix of Task A), Task B will close completely. Any operation executed in Task B that is reflected on the client (for example, calling a browser task and a Verify operation) is ignored. In this case, you cannot prevent Task B from closing.

Top Window Overlay

You can replace the content of the topmost window (the window that displays the root task) without closing the context by:

1. Raising an asynchronous event (WAIT=NO), which is handled by the Main Program. The Main Program's handler contains a Call Program operation, whose frame name is *_top*.
In this case:
 - The new task's prefix is executed.
 - All the prior tasks are completely closed.
 - The new task appears on the screen.
2. If eDeveloper cannot close the prior tasks (a verify or data error with error Mode = Recover), the new task will be closed by executing its Task Suffix operations.
3. Raise an asynchronous event (WAIT=NO) in the Task Suffix of the root task, which is handled by the Main Program. The Main Program's handler contains a Call Program operation. It does not matter what the destination frame is, since when the handler is executed, all browser programs are closed. The called program becomes the new task.

The First Record Prefix

The first Record Prefix handler of the first record, which is executed at the beginning of the task, used to be executed on the server side just after the Task Prefix handler. Other Record Prefixes were executed on the client side. The first Record Prefix is now executed on the client side when the page is displayed and becomes active. In this way the execution for all Record Prefix handlers is the same.

Calling An External Event

The MGExternalEvent function provided by the Browser Client JS module enables eDeveloper to invoke the eDeveloper External Event from an external JS module that is executed from a Browser Client page. When the

MGExternalEvent function is executed, the External Event is added to the eDeveloper event queue and is recognized as a regular eDeveloper event.

MGExternalEvent function

- Syntax: MGExternalEvent(*parameter 1, parameter 2*)
- Parameters: *parameter*: A string value passed as a parameter. The passed values are received by the Handler's virtual variables according to their order. There is no limit to the number of parameters allowed.
- Returns: The function returns a True value if the Browser Client is loaded or returns a False value if the Browser Client cannot be loaded.

Calling a Batch Task

The browser task supports calling batch tasks that send output to the I/O requester, selected from the I/O repository's Media column. When you call and send a batch task that produces an HTML form or an HTML Merge form to an I/O requester, the batch output is displayed in a browser window as defined in the Call program, Call Task, or Call Exp operation destination property.

Any destination frame defined in subsequent Call operations to other batch tasks are disregarded. Only the destination frame defined in the Call operation to the browser task's batch task is used.

Batch Within the Current Context

The batch task that is invoked by a Call Program, Call Task, or Call Exp operation is run within the active context of the calling task. This means that all the context resources, such as variables, task tree, global parameters, memory tables, and transaction, are available to the called batch task. Any modification made to the context resources will be kept in the context when the batch task is completed.

Modeless Window

The window of the batch task's result output cannot be a modal window, which does not let the user select another window without first closing the selected

window. It is always opened as a modeless window, which lets the user select another window without having to close the selected window.

Result Window Out of Context

The result page that opens when the batch task ends is from the originating context.

Output from Nested Batch Tasks

The output to the requester can be produced by the immediate batch task that is called from the browser task, and also by a batch task that is a descendant of the first batch task called. The output is returned to the client only when the immediate batch task is completed.

No Output, No Window

If you run a batch program that does not produce output to the requester, a page does not open.

Recompute

Recompute is an implicit event that occurs with the change of a field value. For each field value change, the following recompute items are checked:

- Other field Init expressions
- Other control properties
- Event handlers triggered by an expression

If the value changes require a recompute event and the list indicates that the server should do the recompute, the client sends a command to the server and the server returns the computed values.

Field Level Validation

The client validates the end-user input by checking the picture, mask and range of the fields. The checked types are:

- Numeric

- Alpha
- Memo
- Date
- Time
- Logical

The mask is used to build and evaluate the field of the end-user's input.

An end-user can insert any string in the field. The field is only validated when it loses focus. The following conditions apply when the end-user enters a value into a field:

- The initial control value is the mask combined with the value of the field.
- The end-user can insert any character into the Alpha or Memo fields. If an Alpha or Memo field has a picture that consists only of upper or lower case characters, or only of numbers, the validation can be done on the fly. The same can be done with Numeric fields that consist of number slots ('#') only.
- The end-user can insert any character into Numeric, Date, or Time fields. After exiting from the field, the string is evaluated and validated by comparing it to the field picture and type.
- If the end-user input does not comply to the field mask, the previous value is returned.
- If the data received after evaluation cannot be correctly interpreted for the given field, it returns the previous value.
- If the picture has a range, the user's input is accepted only if the range is distinguishable. Otherwise, it returns the previous value.

- After the evaluation and validation of the data, the string appears in the input field. If the previous value is returned, a message appears on the status bar and the focus is returned to the corrected field.



For more information, refer to *How To...Working with Magic*, Building a Browser-Based Task.

Creating a Batch-Based HTML Program

Conventional Internet Application Flow

Conventional internet applications, do not maintain a non-stop connection with the eDeveloper enterprise server. Internet applications more closely resemble batch processes, as when an HTML page is generated by the eDeveloper server in the background. When the page generation is completed, including images, dynamic SQL data, static texts, and so on, the

HTML page is sent to the client browser. At this point, the eDeveloper task ends. The HTML page becomes an independent entity on the client browser, having no connection to the eDeveloper enterprise server and the eDeveloper Request Broker.

Internet applications usually operate in page mode technology. A request to retrieve an HTML page is issued from a browser either by a hyperlink or a URL address.

A specific web server reads the request with embedded parameters. That web server then sends an HTML file back to the browser, which in turn can send information back to the web server.

Upon receiving the request containing the client parameters from a browser, the eDeveloper enterprise server generates and sends an HTML file directly to the client browser through the Internet requester in use.

The eDeveloper HTML Form Editor

The eDeveloper HTML form editor lets you construct HTML pages; by editing, browsing, manipulating hyperlinks, designing tables and frames, and using tags. The eDeveloper HTML form editor supports many controls including

- Text
- Rich Text
- Push Button
- Image
- Combo List Box
- Check Box
- Radio Button
- Table Control

eDeveloper provides the means to generate HTML pages with embedded application data at deployment, letting you integrate static text with dynamic SQL data and graphical objects.

A Java applet control included in the HTML form palette allows external Java applets to be easily integrated at deployment. In this way, client functionality and interfaces can be distributed across the Internet while application logic remains secure and isolated. The eDeveloper Form Editor control palette also includes an ActiveX control that enables ActiveX objects to be easily integrated within eDeveloper. You can therefore take advantage of the growing number of available ActiveX components. The eDeveloper HTML form editor frees you from dealing with HTML tags and storing HTML files.

The eDeveloper Enterprise Server and Other HTML Editors

Seamless integration with external HTML editors provides merge processing between the eDeveloper HTML form and the external HTML authoring tool.

Where necessary, you can continue to develop or support existing HTML files with the HTML tool of their choice while gaining the benefits of eDeveloper development and deployment environments.

The eDeveloper Frame Set Editor

The eDeveloper Frame Set editor lets you divide a browser window into two or more windows, each displaying a different document. Note that each document can be the result of a combination of different HTML editors.

Frame sets in an HTML document can cause a web page to appear to be divided into multiple scrollable regions. For each frame within a frame set, the eDeveloper Frame Set editor can assign an eDeveloper Internet application, a name, a border, scrolling, and so on.

The eDeveloper Frame Set editor frees the developer from tasks such as writing HTML tags and storing HTML files. Application Partitioning

The Benefits of Application Partitioning

Partitioning an application has several potential benefits:

- Environment Independence

Application partitioning provides users a distribution of application services through a network of servers. Partitioning allows for the support of multiple hardware platforms, operating systems, networks, GUIs, Internet, and database management systems. Other advantages include:

- Utilization of an optimized server
- Reduced network traffic
- Improved resource allocation and load balancing
- Parallel execution of tasks

- Increased maintainability

Application partitioning provides increased application functionality by providing application access to many remote users. This functionality lets you reuse the same services for multiple applications. Other advantages include:

- Centralized code management
- Deploying multiple application configurations

- Utilizing existing software and legacy systems without a need to port or rewrite.
- Enhanced Reliability

Application partitioning provides for application component fail over, and automated version consistency.

The Call Remote Command

The Call Remote command lets you call to a remote service from an eDeveloper program. The return status of the Call Remote command can be used to receive the Request ID sent to the Magic Request Broker, or for an error message in case of an error. The calling program can pass parameters to the remote service and can receive values from it. The Call Remote command can be executed in a synchronous mode (Wait) or asynchronous mode (No wait).

Note that in the special case in which the remote service runs on the same system as the calling application, the Call Remote command allows requests among eDeveloper applications.

The *Call Remote* operation defines the service, program, arguments, and parameters that can be executed. Zoom from the Call Remote line to access the Call Remote dialog, and define the parameters required for the remote execution.

The Call Remote dialog contains the following parameters:

Parameter	Meaning
Service	The name of the Service from the Service repository. You can select a Service from the Services list.
Program Name	The public name of the program to be run.

Parameter	Meaning
Result File	Zoom from the Result File field to the Windows Open dialog to specify the file name for the return results. All result data is written to that result file.
Return Code	Zoom from this field to the Variable list to select the name of the numeric return code variable. If positive, the value returns as the request identification of the request.
Reason Code	Zoom from this field to the Variable list to select the name of the Reason Code variable. If positive, the value returns a reason string for the returned request.
Message ID	Zoom from this field to the Variable list to select the name of the message ID Code variable. If positive, the value returns a message identification of the request.
	Zoom from the string field to the Expression Rules repository to set an expression for the return value of the Message ID.
Priority Exp	Zoom from this field to the Expression Rules repository to select the expression that determines the conditions of the Call Remote execution.

The Call Remote operation line includes a Yes/No expression for the Wait field that appears in place of the Form field, to specify whether the client is required to wait until the program execution ends before continuing.

The program arguments are passed in the same way as is with the Call Program operation.

Synchronous Execution vs. Asynchronous Execution

The flow of messages among the Requester Client, the Magic Request Broker and the eDeveloper Runtime engine depend on whether the request is synchronous (Wait) or asynchronous (No Wait).

When a synchronous (Wait) request is received from a client, the Magic Request Broker issues a Request ID for that request. The MRB then assigns a free eDeveloper Runtime engine to execute the request, and sends both the Request ID and the Runtime Engine address to the client. Communication is then passed directly between the Requester Client and the eDeveloper Runtime engine. When the request is completed, the eDeveloper Runtime engine sends the request results directly to the Requester Client, and notifies the Magic Request Broker that it has become available. During this request process, the client must wait for the runtime engine to return the request results.

When an asynchronous (No Wait) request is received from a requester client, the Magic Request Broker issues a Request ID. The Requester Client does not wait for the request results before resuming the operation. The MRB does not return a runtime engine address to the client, nor do the client and runtime engine communicate directly. The Requester Client must query the Magic Request Broker for the request results through the use of its Request ID.

Processing Synchronous Requests

When the Magic Request Broker receives a synchronous request from a request client, it responds as follows:

- Assigns a Request ID to the request client

- Finds a free eDeveloper Runtime Engine and sends the Request ID and runtime engine address to the request client

When the Magic Request Broker returns the Request ID and runtime engine address, communication continues directly between the request client and the runtime engine without any additional MRB intervention. Once the request is completed, the runtime engine will notify the Magic Request Broker that it is available to process the next client request.

Processing Asynchronous Requests

When the Magic Request Broker receives an asynchronous request from a request client, it responds as follows:

- Assigns a Request ID to the request client
- Finds a free runtime engine, and sends the client's request to it
- The Magic Request Broker receives notification when the request execution is completed, and reassigns the runtime engine to the next client request

The Requester Client must queue the Magic Request Broker to receive the request results.

Dynamic Assignment of Partitions

eDeveloper Version 9 provides for a dynamic assignment of partitions by dragging and dropping arrow connections from a server to a service.

The Visual Connection screen displays services on the right of the screen assigned to a list of servers shown on the left of the screen. Servers and Services entered in the Server and Service repositories always appear.

Reassign a service to another server by clicking a service icon and dragging to a new server.

Setting Up an eDeveloper Partitioned Application

Choosing What to Partition

The first step in creating a partitioned application is choosing which components of the application run on the server system or systems. The criteria for making the choice are the components that would most benefit in performance and maintainability, if they were to run on the server. Note that the decision whether a program or task runs in the Requester Client or in the system need not necessarily be made when the application is designed, but the design must take this into consideration. Only background-mode tasks are applicants for partitioning.

Runtime Behavior

When a Call Remote command is encountered during runtime, the runtime engine checks for a valid server, and then passes the requested command to the eDeveloper Runtime requester. If the operation's Wait field is set to No or evaluates to No, eDeveloper will continue. If the Wait field is set to Yes or evaluates to Yes, eDeveloper waits for the completion of the request or until a time-out failure occurs. After the request has been completed, the variables that were sent are recomputed, and the flow of the program continues.

The following table defines the error codes and messages that can be returned:

Code	Error Mnemonic	Error Message	Meaning
4	NO-RESULT		The requested task did not return output to the requester output media (IO).
101	BAD-ARGS	Remote Call Error - Invalid Remote Call	Application or program names were not entered for the remote call.

Code	Error Mnemonic	Error Message	Meaning
102	CNCT-MRB-REFUSED	Remote Call Error - Connection to Broker Refused	Wrong host name or port for the Broker. Verify that the broker is connected to the port on the specified machine.
103	APP-NOT-FOUND	Remote Call Error - Application Not Found	The engine does not support the Application; or the Appl=entry in MGREQ.ini file was specified and the required application did not appear in the list (not relevant to the eDeveloper 9 Client).
104	APP-IN-USE	Remote Call Error - Application in Use	All the engines that support the application are busy serving other requests; or the engine does not support the application. The Broker Timeout value may have expired. For the eDeveloper Client, the timeout value is in the Server repository. From the MGREQ.ini file, the time out value is in the Timeout parameter.
106	RT-NOT-RESPOND	Remote Call Error - Runtime Not Responding	The Server Timeout parameter in the Magic.ini file or the MGREQ.ini file has expired. The server engine did not send an I-AM-ALIVE message during the specified time interval.

Code	Error Mnemonic	Error Message	Meaning
107		Connection reset by the enterprise server	This eDeveloper partitioning message is given when one or the following situations occur: (1) The enterprise server was aborted abnormally during the execution of a request, or (2) The connection was reset due to a network connection failure.
110	REQUEST-TimeOUT	Remote Call Error - Request Timed-out	The Request Timeout entry has expired in the Magic.ini or the MGREQ.ini file. The requested task was not completed in the specified time interval.
130	APP-OPEN-FAIL	Remote Call Error - Failed To Open Application	The server engine could not open the application control file.
131	PRG-NOT-FOUND	Remote Call Error - Program Not Found	Cannot find the public name of the program specified in the application.
133	ACCESS-DENIED	Remote Call Error - Access Denied	Invalid user name. No rights assigned to the user. No right to execute program.

Code	Error Mnemonic	Error Message	Meaning
135	LIMITED-LICENSE-HTTP	Remote Call Error - License Limited to Internet Requests	Server engine is licenses to execute requests from internet requesters only.
136	LIMITED-LICENSE-CS	Remote Call Error - License Limited by Requests Count	Server engine is licensed to execute a limited number of requests.
137	REQUEST-REJECT	Remote Call Error - Request Rejected	Server engine cannot execute the request due to a timing problem of switching from Runtime to toolkit modes.

Code	Error Mnemonic	Error Message	Meaning
138	RT-ERROR-MSG		<p>During execution of a request, the executed program, or subprograms, displayed error messages such as SQL or locking errors.</p> <p>If the executed program failed to complete, these error messages were trapped by the enterprise server & sent back to the requester. If the executed program was executed successfully, despite the error messages, the program's output is returned, overriding any error messages.</p> <p>When the requester is an Internet requester, the error messages are sent to the remote browser.</p> <p>When the requester is a command line requester, the error messages are displayed in the console.</p> <p>When the requester is an eDeveloper 9 engine, this error message is not displayed.</p>
146	BIND-HOST-NOT-FOUND	Remote Call Error - Bind Failed - Unknown Host	The Local Host entry in the MGREQ. ini file must specify the local internet address.

Code	Error Mnemonic	Error Message	Meaning
147	CNCT-HOST-NOT-FOUND	Remote Call Error - Connection Failed - Unknown Host	The Broker address must explicitly contain the internet address (i.e. 88.0.184/3001).
150	CNCT-REFUSED-MRB	An enterprise server could not connect to the Broker	An enterprise server couldn't connect to the Broker. Check if the Broker was started, and that the host name of the Broker, as known in the computer of the enterprise server (ping <mrhost>), belongs to the correct IP address (as known in the computer of the Broker-ping <mrhost>). Reproduce the problem with log "enabled" in mgreq.ini (log = req.log Y R), in the directory of the enterprise server.
151	CNCT-RESET-BY-REQ	During execution of a request, the connection between the requester & the enterprise server was reset	As a result, no output was sent back from the enterprise server to the requester. Check the requester in the client machine & the enterprise server. If possible, reproduce the problem with log "enabled" in mgreq.ini (log = req.log Y R) in the directories of the requester & of the enterprise server.

Code	Error Mnemonic	Error Message	Meaning
200	RQ-FATAL	Remote Call Fatal Error	Misc. Errors.
201	TCP/IP Not Initialized	Code Partitioning Error -	TCP/IP services were not installed. Install TCP/IP.
202	ERR-OPEN-RESULT-FILE		This is a two-part problem: (1) The requester asked that the output be written into a file (either using a field in the call remote dialog box in an eDeveloper 9 client, or using keywords in mgreq.ini or mgrqcmdl -file=), and (2) The combined file name (directory and/or file name) is illegal.

Command Line Requester

The Command Line Requester is a requester management program that allows for:

- Executing remote eDeveloper services
- Managing the Magic Request Broker and the eDeveloper Runtime engines (to start and stop the MRB or eDeveloper Runtime engines)
- Querying the Magic Request Broker regarding the status of specific requests that were issued to eDeveloper Runtime Engines
- Querying the Magic Request Broker regarding the status of the eDeveloper Runtime engines

The Command Line Requester is activated by selecting the mgrqcmdl.exe file.

The Command Line Interface

The Magic Request Broker can be queried by using the Command Line requester, which is mentioned in the Command Line Requester section above as one of three types of eDeveloper requesters. The Command Line requester is the MGRQCMDL.EXE program, which can be run in a DOS-Prompt window. Like other eDeveloper requesters, the Command Line requester uses the MGREQ.ini file to determine the address of the Magic Request Broker to which it communicates.

The full syntax tree for using the Command Line requester can be viewed by using the MGRQCMDL command without parameters. Command line requests are described below.

- To view the eDeveloper engines that are available for service to the MRB, use the command:

```
mgrqcmdl -query=rt
```

- To view the eDeveloper applications that are available for service to the MRB, use the command:

```
mgrqcmdl -query=app
```

If the MRB is not running, an error message is issued:

Command Line Requester: Connection to broker refused (error -102)

If the MRB is up but the Travel Agency application is not available, the display is as above, but the Travel Agency application is not be shown.

- To load an additional eDeveloper engine, use the command:

```
mgrqcmdl -exe=RuntimeName
```

where *RuntimeName* is the name of an entry in the [MRB EXECUTABLES LIST] section of the MGRB.INI file.

- To stop any eDeveloper engine, use the command:

```
mgrqcmdl - terminate=EngineId - timeout
```

where *EngineId* is the engine's host and port number.

When the engine is instructed to terminate, it terminates gracefully without any new contexts being opened in the terminating engine. Requests

for a new context are directed to other available engines. If another engine cannot be found, the request is rejected. The terminating engine continues to process existing contexts, and will terminate after the last context is properly closed or after the set timeout has passed.

When the timeout value equals zero, the server engine waits indefinitely to terminate. The timeout default value is zero.

The same command line interface can be used to query the MRB regarding its queue of requests, and to change request priorities or remove them from the queue.

Multi-Threading

For eDeveloper Version 9, background enterprise servers are multi-threaded. Each thread accesses a different Runtime context, and does not interact with other threads.

Shared Resident Tables

Resident tables are shared for all contexts, which improves the performance of each context by eliminating the need to load and maintain separate copies of resident tables for each context string.

Environment Settings

Maximum Number of Concurrent Requests

Under the Partitioning Web tab, from the Environment Settings dialog, the Maximum Concurrent Requests setting is displayed. You can enter the number of threads that the enterprise server is allowed to create. The user is only limited by server capacity, and the type of user license. The Magic.ini and Command Line name is `MaxConcurrentRequests`.

Closing an Application

The enterprise server allows a request to close the current application and open another application only if there are no other requests.

INIPut and INIGet

Magic.ini is stored for each Runtime context in a separate memory area. The INIPut function for a specific Runtime context is written to the Magic.ini file only if it is specifically requested by using the following flag:

INIPut (<assignment>,<force write>) (default - N)

Security

The user can add or remove user rights to each thread through the use of the Runtime (RT) functions. Runtime functions are described in the Runtime Expressions section below.

TCP/IP Ports

Each enterprise server uses one TCP/IP port.

Broker and the Generic Messaging Layer

No interoperability between eDeveloper Version 9 and any previous Magic versions is possible.

Main Programs

A separate copy of the Main Program is available for each Runtime context.

Transactions

Each thread acts as a separate process, independent of the other threads.

Database Files

Each thread acts as a separate process, independent of the other threads.

Locking

Each thread acts as a separate process. However, locking between threads is shared in memory. Each enterprise server process use one terminal number.

Shared Resources

The following resources are shared by every Runtime thread:

- DBMS connections
- External devices

Runtime Functions

The following Runtime functions are available:

- RqExe - Requests a Magic Request Broker to load a new entry from a predefined list of executables.
- RqRtTrm - Terminates an enterprise server.
- RqRtInf - Returns the current number of busy threads, the maximum number of threads allowed by license, and the most frequently used threads.
- DiscSrvr - Disconnects the current thread that is no longer required by eDeveloper.
- DbReload - Reloads a resident table and replaces the copy of the resident with the new copy for all contexts.
- DbDel - Clears the content of a resident table, but does not delete any of the physical records from the table. The DEBDel function is applicable also for enterprise servers.

Calling External Programs

- Call a DLL or a Userproc/UDF

External DLLs need to be compatible with a multi-thread environment before they are called.

- enterprise server Termination

If a termination request is sent from a requester, while one or more requests are still being processed, the termination request invokes a flag that determines whether the request should wait. If the termination request does not wait, the requester receives a warning message.

Syntax:


```
RqRtTrm(..., 'T'log | 'F'log)
mgrqcmdl -ter=server/port -wait
```

eDeveloper Monitor Application

The eDeveloper V9 installation includes an eDeveloper application for monitoring the Magic Request Broker. In addition to being a useful monitoring tool for the MRB, the application can be used as an example of the use of the eDeveloper V9 Requester functions.

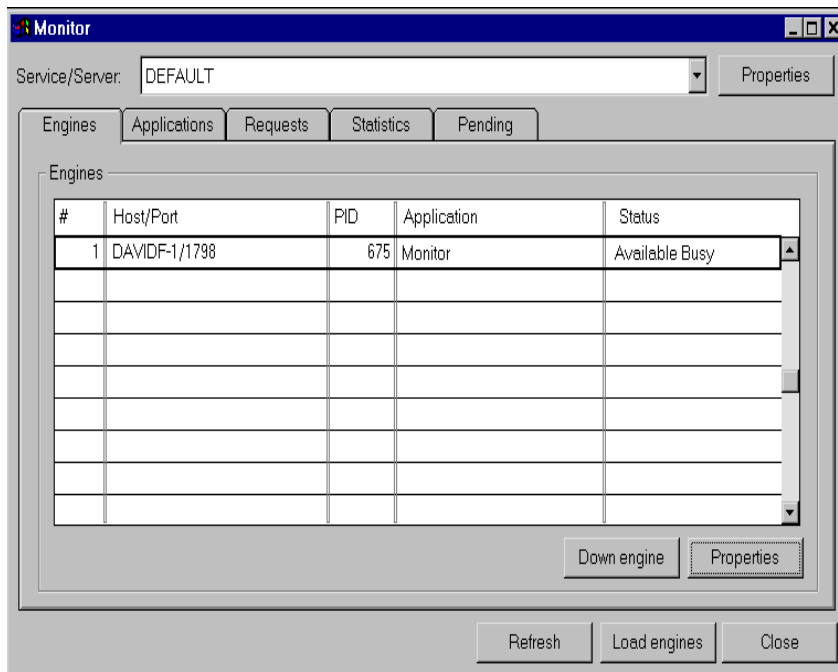


Figure 19-3 The Monitor Application

The Monitor application lets you select a service, and offers the following functions:

- Display a list of server engines and their properties
- Display the list of applications supported by the server engines

- Display the request queue
- Display the list of pending requests
- Display statistics on the requests that were handled by the Magic Request Broker since its startup
- Start a new server instance
- Shut down a server instance

Request-Related Functions

The following table provides a short description of the functions that support application partitioning. For a full description of each Application Partitioning function, refer to Chapter 8, Expression Rules.

Function	Description
GetParam	Gets global variables.
RqExe	Requests a MRB to load a new entry from a predefined list of executables on the MRB local computer.
RqLoad	Provides statistical information about the load of one or all services of a single broker.
RqQueDel	Deletes an entry in the Service Queue.
RqQueLst	List of pending requests in the queue.
RqQuePri	Resets the priority for a pending request in the queue.
RqReqInf	Provides information about a request through a list of values that are generated from the Queue or from the Broker's history log. Executes only when RqQueLst or RqReqLst is called before.

Function	Description
RqReqLst	Returns the number of request entries, specified by a range of request identifications from the broker's log.
RqRtApp	Returns information about one application supported by one or more Runtime engines registers to the broker.
RqRtApps	Provides the number of applications supported by one or all enterprise servers associated with the broker.
RqRtInf	Gives information about a specific enterprise server associated with the broker.
RqRts	Gives the number of enterprise servers associated with the broker.
RqRtTrm	Terminates an enterprise server or all the enterprise servers associated with a requester.
RqStat	Returns a simple, numeric value indicating the status of a single request.
SetParam	Sets global variables.

Magic includes a variety of utilities designed to help you to develop, modify, maintain, and move your eDeveloper applications.

In this chapter:

• Application Wizard
• Program Generator
• Check Syntax Utility
• Get Definition
• Cross Reference
• Export-Import
• Flow Monitor and Debugger
• Profiler
• OEM2Ansi Utility
• ODBC Check Driver
• MakeKey Utility
• Table Conversion Utility
• Magic Flat File
• Print Data Wizard
• Tools Infrastructure
• Documentation Template Facility

Application Wizard

The Application Wizard lets developers easily access the eDeveloper development environment and quickly create a working application.

The Application Wizard lets you create data objects, interactive programs, reports, and batch programs. Click Application Wizard on the Tools menu to start creating an eDeveloper application.

Automatic Program Generator

eDeveloper's Program Generator is designed to create programs automatically. The Program Generator generates programs for:

- Online data entry and maintenance
- Exporting data from the database file to an operating system text file
- Importing data from a text file to a database file
- Printing reports
- Internet
- Browser clients

Program Generator Properties for Database Tables

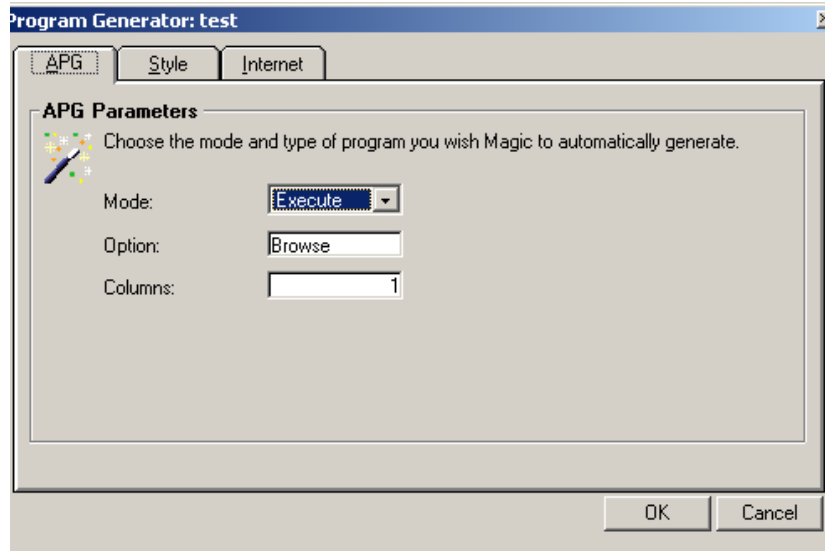


Figure 20-1 The Program Generator

APG Tab

Mode

The Program Generator operates in either of two modes: the execute mode or the generate mode. The execute mode creates and runs a temporary program. This program is not saved after running. The generate mode generates a program, saves it, and appends it to the Program repository. The Generate mode does not automatically run the generated program. Execute is the Mode property default.

Note: The Mode parameter can only be accessed when a Table Repository row is selected.

Option

The Option parameter states the program's function. Six options are available: Browse, Export, Import, and Print, Internet, Browser Client.

The Browse option runs or generates programs designed for online data entry and maintenance.

The Export option generates a program to export the data contained in each selected database file. The data is exported to an operating system text file.

The Import option imports a program in Execute mode or generates a program designed to import data in Generate mode contained in an operating system text file. The data is imported to the database file. The structure of the operating system text file is created by the Program Generator while using the Export option. The Export and Import options are designed to generate a pair of programs that export and import the same data.

The Print option prints an eDeveloper data file or generates a program designed to print an eDeveloper data file.

The Internet option generates programs designed for Internet data entry and maintenance. For eDeveloper Programs only. Not available for Execute mode in the Table repository.

The Browser Client option generates a Browser-based program. You can apply an independent template for the generated program. Not available for Execute mode in the Table repository.

Column

The Column field displays the number of columns (for database rows) from the selected entry. You can change the order of the columns by changing the third column value. A zero (0) value omits the column altogether. You cannot change the column order of a linked table.

Generate Forms

The Caption field defines how each program will be generated: Query, Output, and Both for Internet option only.

Query - the APG generates a skeleton program containing the HTML form used to query the repository. The generated program can be used to either edit the query form in the HTML form editor, or to save the form to a file using the Save as HTML command.

When generating for Query, you can also choose to automatically save the form to an HTML file.

Output - the APG generates a program that, when called from the appropriate HTML form using the Internet requester, produces an HTML report to the requester. The HTML report contains all records of that form.

Both - the APG generates a program that contains both the Query form and the HTML report form. When the generated program is called from the Query form, the program produces an HTML report to the requester. The report contains all records that fall into the range defined in the calling form.

Program Name

The name of the program as it appears in the Program repository.

HTML File Name

The name of the HTML file for the Browser client.

Template File Name

The Template file name that is associated with the HTML file of the Browser client.

Text File

The name of the operating system text file to be exported or imported.

Links

The Link property displays the Foreign Key selection list. From this list, you can define the Foreign Key for the generated program. For each selected Foreign Key, a Link operation is added to the generated program. The Link Query operation is generated when either the Main table or referenced table is assigned to a default database. A Link Inner Join is generated when both the main and referenced tables are assigned to a RDBMS database.

The Link property appears when you generate a program from a table with a foreign key.

Style Tab

Display

You can choose the following options to display records on the screen, Line and Screen.

- Line mode displays records as rows. Each row corresponds to a record. Many rows are displayed per screen.
- Screen mode displays one record per screen.

Style

For Screen mode, you can display as:

- 3-D
- 2-D
- Original

Caption

You can determine if the program name is displayed on the form.

Use Model

You can assign an existing Form model to the generated program.

Form Size

The Form Size property lets you specify the display of the GUI form created by the Automatic Program Generator when the mode is set to Execute and the option to Browse. The Form Size property has the following options:

- As Model – The form width and height is defined as specified in the attached model.
- As Content – The form width and height is according to the result content on the screen. Note that in Line mode, the height of the form will be the same As Model.

- As Content within MDI – The form width and height will be according to the result content on the screen, however, each measurement will not be greater than the MDI measurements.

Note: If the content width is greater than the MDI, the content width will be the same as the MDI. If the content width is smaller than the MDI width, the form width will be the same as the content width.

Internet Tab

Query Title

The Query Title property defines the title and heading of the HTML query form.

Query Wallpaper

The Task Query Wallpaper property contains an image that will be used as the wallpaper image of the HTML query form. The rules regarding the location of Internet Application images apply.

Output Title

The Output Title parameter defines the title and heading of the HTML output form.

Output Wallpaper

The Task Output Wallpaper property defines an image to be used as the wallpaper image of the HTML output form. The rules regarding the location of Internet application images apply.

Save Query Form As

The Save Query form As property defines the file name of the query form to be created. This is the same form that can be found in the generated program when the APG generates for Query or Both. If the field is left empty, no file is created.

Program Generator Properties for a Program Entry

Option

The Option property states the program's function. Six Tasks are available: Browse, Export, Import, Print, Internet, Browser client.

The Browse option generates programs designed for online data entry and maintenance.

The Export option generates programs designed to export the data contained in each selected database file. The data is exported to an operating system text file.

The Import option generates a program designed to import data contained in an operating system text file.

The Print option generates a program designed to print an eDeveloper data file.

The Internet option generates programs designed for Internet data entry and maintenance for eDeveloper Programs only.

The Browser Client option generates a Browser-based program. You can apply an independent template for the generated program.

Main Table

The eDeveloper table from the Table repository for which you are creating a program.

You must select the name of the database file in order to activate the Program Generator from the Program repository.

If you activate the generator from the task tree, eDeveloper provides the original database table by default, but you can access the column and change the table.

In the Main Table property, either enter the table's sequential number from the Table repository or zoom into the Table list to select a table entry.

Generate Forms

The Generate Forms field defines how each program is generated: Query, Output, and Both.

- Query fields

The Query Fields property defines which fields are to be included in the generated query program and the task form.

- Output fields

The Output Fields property defines which fields are to be included in the generated output program, and are displayed in the HTML report returned by the program.

Internet Tab

Query Title

The Query title property defines the title and heading of the HTML query form.

Query Wallpaper

The Task Query wallpaper property defines the wallpaper image of the HTML query form. The rules regarding the location of Internet application images apply.

Output Title

The Output title property defines the title and heading of the HTML output form.

Output Wallpaper

The Task Output wallpaper property defines the wallpaper image of the HTML output form. The rules regarding the location of Internet application images apply.

Save Query Form As

The Save query form as property defines the file name of the query form to be created. This is the same form that can be found in the generated program

when the APG generates for Query or Both. If the field is left empty, no file is created.

Check Syntax Utility

This utility lets you check models, tables, tasks, programs, help screens, and menus for incorrect syntax.

eDeveloper's rapid development method lets you fill in repositories without the delays incurred by online syntax and logical checks. It is your responsibility to run the syntax checker to verify the syntax of the program.

Checker Message Categories

The checker results are divided into three categories:

- Error - Identifies incorrect syntax that may result in an unexpected behavior.
- Warning - Identifies incorrect syntax that may not necessarily result in an unexpected behavior.
- Recommendation - Provides a better syntax alternative.

For more information about setting the checker level, see The Toolkit Checker Minimal Level environment setting in the Environment dialog box under Preferences in Chapter 2, Settings.

Checker Results

Checker results are displayed in the Checker Results window, as shown in Figure 20-2.

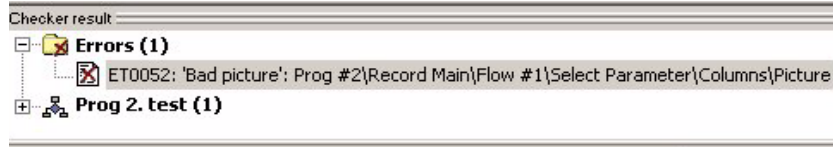


Figure 20-2 Checker Results Window

The Checker Results window can be docked to the eDeveloper MDI screen or displayed as a floating window. Checker results can also be combined with the property sheet and navigator.

Checker results are displayed in a data tree format, and the result display group selection determines the first level of the tree. The check syntax results can be displayed by:

- Type - Grouped by the checker message type: Error, Warning, or Recommendation. Checker messages are sorted by the order in which errors are found by the syntax checker.
- Object - Grouped by eDeveloper object: Models, Tables, Programs and Subtasks, Help screens, and Menus. Checker messages are sorted by the order in which errors are found by the syntax checker.
- Object and Type - Grouped by both the eDeveloper object and the checker message type.

For information about displaying the checker syntax results, see the Group Checker Messages By environment setting in the Environment dialog box under Preferences in Chapter 2, Settings.

The Check Syntax Process

When the Check Syntax utility is running, a message box appears, as shown in Figure 20-3.

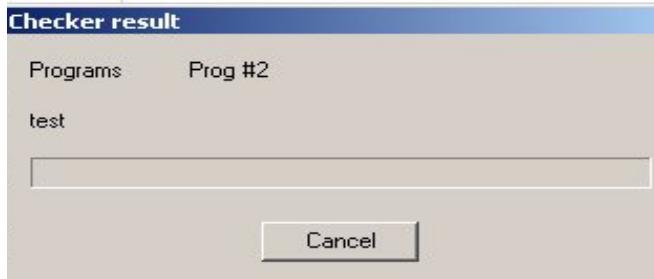


Figure 20-3 Syntax Checker in Process

If the checking process is canceled before completion, the Checker Results window displays the incorrect syntax found when the Check Syntax utility was stopped.

When the check syntax process has been completed, one of the following prompt messages is displayed:

- For incorrect syntax at the Error and Warning levels:
Check Syntax completed. Please refer to the Checker Results window.
- For incorrect syntax at the Recommendation or Unused Objects levels:
Program is OK. Please refer to the Checker Results window.
- When incorrect syntax has not been found:
Program is OK.

Note: If the Jump automatically to First Item environment setting is set to No, eDeveloper does not highlight the first checker result message entry and does not park in the field where the error occurred.

Unused Objects

When the syntax checker finds an unused object, eDeveloper prompts you to select one of the following commands:

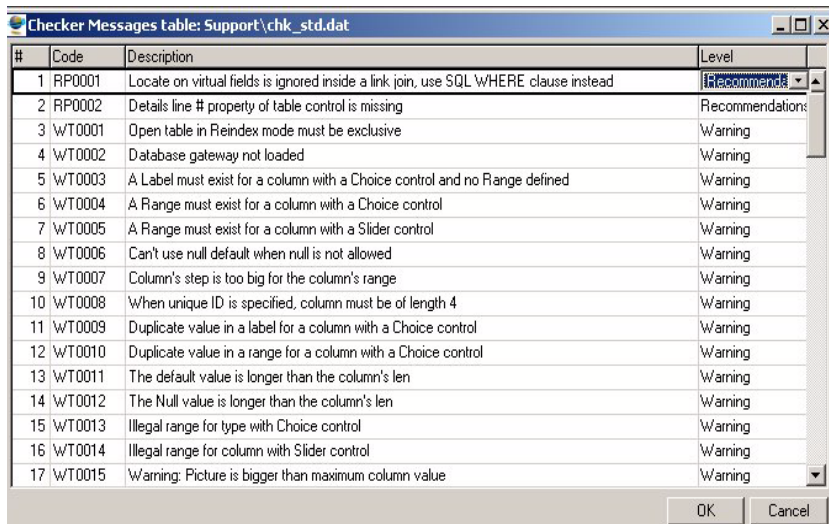
- Erase - Erases the current object expression.
- Erase All - Erases the current object expression and all other unused expressions of the same object.
- Skip - Keeps the current object expression.
- Skip All - Keeps the current object expression and all other unused expressions found in the check process until the next check is done.

Messages about unused expressions are displayed in the Checker Results window.

Checker Messages Table

The Checker Messages table lets you customize the level of each message for the checker. You can set a message as an Error, Warning, Recommendation, or to be ignored by the checker. All checker messages are listed in the table with the exception of error messages.

Click Checker Messages in the Settings menu to open the Checker Message table, as shown in Figure 20-4.



Checker Messages table: Support\chk_std.dat

#	Code	Description	Level
1	RP0001	Locate on virtual fields is ignored inside a link join, use SQL WHERE clause instead	Recommendation
2	RP0002	Details line # property of table control is missing	Recommendation
3	WT0001	Open table in Reindex mode must be exclusive	Warning
4	WT0002	Database gateway not loaded	Warning
5	WT0003	A Label must exist for a column with a Choice control and no Range defined	Warning
6	WT0004	A Range must exist for a column with a Choice control	Warning
7	WT0005	A Range must exist for a column with a Slider control	Warning
8	WT0006	Can't use null default when null is not allowed	Warning
9	WT0007	Column's step is too big for the column's range	Warning
10	WT0008	When unique ID is specified, column must be of length 4	Warning
11	WT0009	Duplicate value in a label for a column with a Choice control	Warning
12	WT0010	Duplicate value in a range for a column with a Choice control	Warning
13	WT0011	The default value is longer than the column's len	Warning
14	WT0012	The Null value is longer than the column's len	Warning
15	WT0013	Illegal range for type with Choice control	Warning
16	WT0014	Illegal range for column with Slider control	Warning
17	WT0015	Warning: Picture is bigger than maximum column value	Warning

OK Cancel

Figure 20-4 Check Message Table

You cannot enter additional checker messages or remove existing messages.

The information displayed in the Check Messages table is kept in an external Chk_std.dat file. Specify the file path in the Checker Messages table file environment setting in the Environment dialog under the External tab. If the file location is not defined, eDeveloper applies its own default settings.

Get Definition Utility

The Get Definition utility enables loading of table definitions directly from the DBMS's Data Dictionary into eDeveloper's Table repository. Single or multiple table definitions can be loaded from a database. The utility does not synchronize definitions between eDeveloper and the DBMS's Data Dictionary. The developer makes such synchronization of data definitions manually.

When using an existing database, it is often preferable to read the existing table descriptions for the RDBMS data dictionary tables than to redefine those descriptions in eDeveloper. This reduces the risk of error and makes the development process quicker and easier. The Get Definition option is available from the Option menu. Note: The Get Definition utility works only for SQL gateways.

Loading Tables

You can load either a single table definition or multiple tables. The menu will be active if an entry in the repository has been assigned to one of the SQL databases

To load a single table:

1. Access the Table repository.
2. Place the cursor on a new line.
3. Choose the database.
4. Enter the name of the table to be loaded in the DB Name column.
5. Click the Options menu. The Get Definition option is enabled.

6. Click on the Get Definition option. The Loading Window is displayed.
7. The Loading Window closes automatically. The table now has the appropriate columns, keys, and foreign keys.

To load several tables at once:

1. Place the cursor on the title line (#) and select Get Definition from the Options menu.
2. The Load Table Definition window appears.
3. Park on the Database field and zoom.
4. Choose the database you want to load from.
5. Park on the Tag Tables combo box and select a value.
6. Select *All*, if you want to load all the database tables accessible to the database, as defined in the Database Table with user name and password, or specify *Several* for multiple tables.
7. The Table Selection window opens.
8. Highlight the desired item.
9. Press the space bar. A check appears in the Select field to show that the table has been selected for loading.
10. Press the space bar on each table you want to load.
11. Press the Select button to close the window.
12. Click OK to accept the operation.

Note the following about Get Definition behavior:

- When loading existing table definitions, eDeveloper uses the most appropriate eDeveloper attribute for each database column.
- For float data type, eDeveloper uses the default picture from the DBMS repository.

Get Definition of a View

SQL also offers view definitions, which can be accessed in the same way as the table definitions are accessed. An SQL view is a virtual table defined by a

query. An SQL view is accessible as a table but does not physically contain rows. You can access an SQL view in query mode, and in some cases depending on the RDBMS, update them just as if they were tables in the RDBMS.

A view does not contain an index. Therefore when loading a view, you need to define a virtual unique index. This is because eDeveloper must have a position for the table.

In Oracle or Informix, which has a ROWID, if the view is based on one table, the ROWID can be obtained, and the position can be ROWID. Otherwise, the virtual unique index is necessary as the positional index.

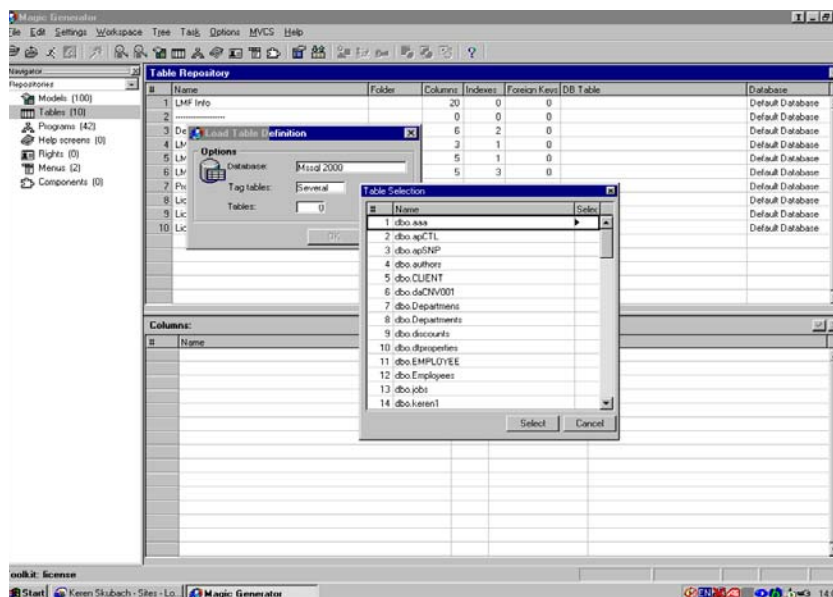


Figure 20-5 Selecting a Database File

Cross Reference Utility

The Cross Reference utility provides information about where an entity such as a model, column, or program is used. An example of where you might want to

use the Cross Reference utility would be to obtain a list of programs that refer to a certain column, index, or table.

The Cross Reference utility lets you find information about the following object types:

- Model
- Table
- Index
- Program
- Help screen
- Right
- Menu
- Modal
- Event
- Component
- Expression
- Form
- I/O field

The results are displayed in the Cross-Reference tab of the Navigation pane and can be printed. You can also delete the results from the result set.

The Location From Where to Cross Reference an Object

The following table explains what objects can be cross referenced from which eDeveloper repositories.

Object to be Cross-Referenced	From eDeveloper Repository
Tables	Programs Choice controls Choice control models Foreign keys
Columns	Programs Indexes Foreign keys Choice controls Choice control models
Indexes	Programs Choice controls Choice control models Foreign keys
Programs	Programs Models/control model Tables/column Menus Task Execution/ Call Program command Virtual field Control properties Subforms

Object to be Cross-Referenced	From eDeveloper Repository
Help Screen	Table fields Program/ real and virtual fields Form properties Controls Menus Field model Control model Form model
Prompt Help	Table fields Program/ real and virtual fields Controls Menus Field model Control model
ToolTip	Table fields Program/ real and virtual fields Form properties Controls Field model Control model
Rights	Models Tables Programs/Tasks Rights literal Helps Menus Application events Components
Menus	Programs

Object to be Cross-Referenced	From eDeveloper Repository
Field model	Tables/columns Programs/tasks (virtual fields, arguments)
Control model	Tables/columns Programs/tasks (virtual fields, arguments) Controls Field models
Task model	Programs
Help model	Help
Form model	Forms
Components	Components can be cross-referenced as an entire object or single items of the component can be cross-referenced.
Main Program events	Programs
Task events	Program tree

You can cross-reference an object from a selected object entry in an eDeveloper repository by clicking **CTRL+X** or by selecting **Cross Reference**

from the Options menu. The Cross Reference (X-ref) dialog box appears as shown below.

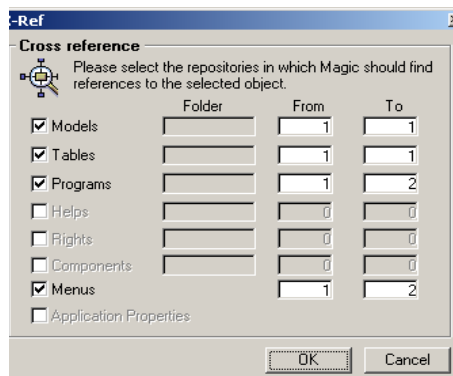


Figure 20-6 Cross-Referencing an Object

Results for a cross reference are displayed by repository name in the Cross Reference (X-ref) navigator pane. The cross reference results are displayed in a composite form.

[illegible]

Figure 20-7 Cross-Reference Results

Each object is displayed with the following information:

- Repository name
- Entry name
- Entry number - in brackets

You can click on the last node (the node which has no other nodes) to display the corresponding repository in the Workspace pane and park on the selected entry.

Cross references have the following characteristics:

- You can delete a cross reference result.
- You can issue a cross reference search from any result entry.
- When the cross reference search has been completed, automatically the Navigator pane opens with the cross reference result. The result is highlighted in grey.
- When the application is closed, the cross reference results are stored in the user's personal directory by application name and file extension (.xrf). This file is in a binary format.

Deleting a Cross Reference

To delete a cross references:

- On the **Navigator** pane, click **X-Ref** and select a cross reference. Press **F3** to delete the cross reference.

Searching for a Cross Reference

To search for a cross references to an item:

1. In an eDeveloper repository, select an entry and press **CTRL+X**.
2. In the **X-Ref** dialog box, select the repositories for eDeveloper to search for related objects.

3. Click **OK**. The results are displayed in the Cross Reference tab of the Navigation Pane.

Note: When you activate a second cross reference search, the cross-referenced object becomes the root in the result tree.

Saving Cross References

To save cross reference results:

- From the **File** menu, click **Cross-Ref Result** and then click **Save Result**.

Printing Cross References

To print cross reference results:

- From the **File** menu, click **Cross-Ref Result** and then click **Print Result**.

Changing the Maximum Number of Cross-Referenced Results

To change the maximum number of cross reference results in the Navigation pane:

1. From the **Settings** menu, click **Environment**.
2. From the **Preferences** tab, enter a number representing the maximum number of cross reference results in the **Maximum number of X-ref results** environment setting.

Export-Import Utility

The eDeveloper Export-Import utilities provide a convenient way to move or convert application components. The services provided include:

- Porting. You can export application components in order to import them into another eDeveloper application across physical sites and platforms.
- Export and re-import the same application in order to reorganize the application file. Note that the operation creates a new application file.
- Produce automatic documentation.

The Import utility imports application components that have been exported using the Export utility. After the import operation, the imported components are appended to the proper repositories. For example, if you imported a database table named Customers, the table will appear at the end of the current application's Table repository. Or, if you imported a program named Customer list, the program will be appended to the Program repository.

When moving from Version 7 to Version 9 using the Export/Import utilities, eDeveloper converts the values for the cache as described below:

Table Properties - If any cache size was defined in Magic 7, it will be converted to Position and Data. If no cache was defined in Magic 7, it will be converted to No Cache.

Task Cache Strategy - The Task Cache Strategy setting is set according to the Cache Strategy in the Table Properties dialog of the of the Main table

Task DB Table - If cache was set to No, it will remain No.

If cache was set to Yes or if an expression was given, the setting is taken from the Cache Strategy in the Table Properties dialog.

The maximum cache memory size property in the Magic.ini file in Version 7 is omitted in Version 9.

To perform an export/import operation, select *File/Export-Import* (**SHIFT+F10**). The Export/Import dialog opens, as shown in Figure 20-8.

The Export/Import dialog contains a number of properties. eDeveloper is sensitive to property selection, and allows access to a property in the dialog if it is consistent with previous selections.

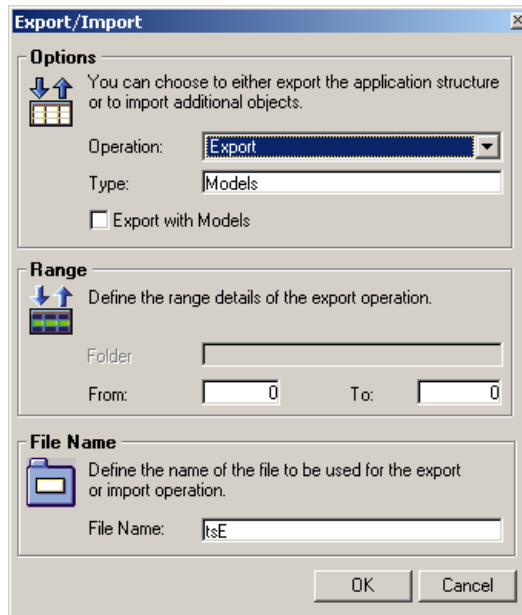


Figure 20-8 Export/Import Dialog

The Export/Import Dialog Box

Magic Version 9 exports documents as ANSI only.

The Import utility detects the import of documents from previous Magic versions. When a document from a previous version is detected, then eDeveloper treats the file as written in OEM. eDeveloper then translates the document to ANSI by using the OEM2ANSI translation file.

Following is the list of properties in the Export/Import dialog, and their possible settings.

Operation

The available operation options are: Export, Export Document, Import.

The property defaults to Export if the utility is activated for the first time. In subsequent cases, it defaults to the operation last performed.

- Exports application components.
- The Export Document option produces a self-documenting report that lists the various components of the entity you exported. eDeveloper's export document utility uses a special template file, DOC_STD.ENG by default, which uses a special description language to set the listing format. The DOC_STD.ENG file can be edited by text editors to customize it for your needs, or by the special Documentation Template facility provided with eDeveloper. An additional DOC_EXT.ENG file produces a more detailed report of the eDeveloper repositories. For details about the Documentation Template feature and the description language for its files, refer to the Documentation Template Facility section in this chapter.
- Imports application components that were exported via Magic Version 7 or higher.

Note: if you import your application and get an error stating that the CALL-MODAL is an unexpected token, just delete that text from the input file.

Type

Type is the application component to be exported.

The available Export options cover all the application components. You can click on the property to open a combo box to see the Export options and to select one. They are:

- Models: exports the Model repository.
- Tables: exports the Table repository.
- Programs: exports the Program repository.
- Help Screens: exports the Help Screen repository.

- Rights: exports the Rights repository.
- Menus: exports the application menus.
- Components: exports application components
- Application Data: exports the application's control data.
- Application: The Application option exports the entire application from one component to another. Therefore, whenever import is performed on an entire application divided into components, the order is as follows:
 - Help screens
 - Models
 - Tables
 - Programs
 - Menus
 - User IDs
 - Rights
 - Application Data

Note: Be sure to import complete components, in order to maintain internal cross-references. eDeveloper will maintain references between repositories and within repositories, as long as the import is done in one session. If the Program repository is split into two or more parts, only the first part imported will maintain cross-references between programs. Importing programs to an existing application may also cause loss of internal cross-references.

Export with Models

The Export with Models property is displayed when you select the Export operation with the Type field set to Models, Tables, Programs, or Help Screens. When the check box is marked, eDeveloper exports your object models with the application. If you do not mark the check box, eDeveloper will export the application without your object models, and the application will either point to

an equivalent model on the end-users computer or will use the application's default system model.

From...To...

If you are exporting models, tables, programs, or help screens, you can indicate a range so as to export only some of them. For example, if you want to export the first three database tables (those appearing on the first three lines of the Table repository), indicate From No.=1 and To No.=3.

From either range field you can zoom to the Table list, the Program list, or the Help list, according to the selection you made in the Type property. Figure 20-9 illustrates the Table list.

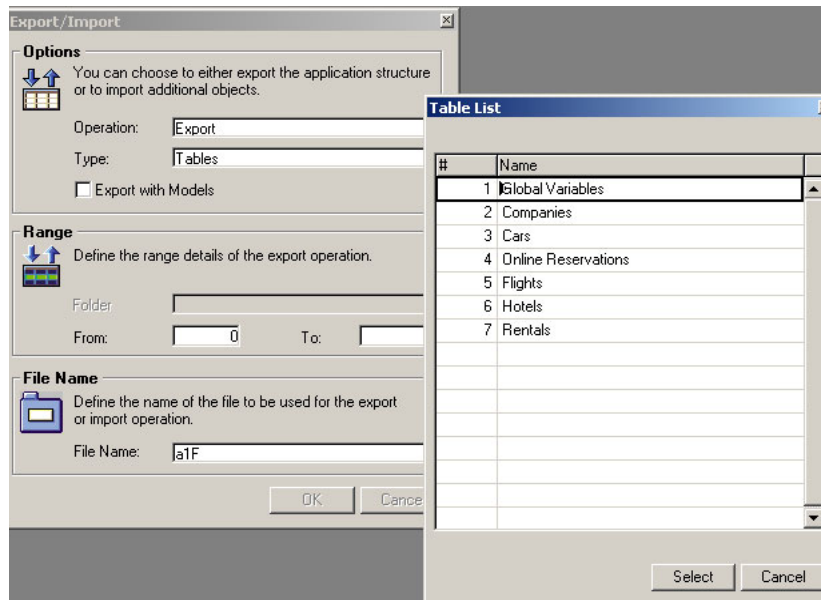


Figure 20-9 Export/Import File List

Folders

- Exporting

Exported repository entries will always be exported with their folders. The exported document will include the folder name.

- Importing

You can import a folder structure by selecting Yes in the Import Folder Structure property of the Export/Import Utility dialog. If the folder already exists, the imported entry appears as the last entry entered in the folder.

If you select No, the imported object will be created in the eDeveloper Imported folder.

File Name

The disk file that will hold the exported components, or the file to be imported.

In an export operation, eDeveloper automatically determines the name of the file, as follows:

- The first part is the application prefix, as defined in the *Settings/Applications* repository.
- The prefix is followed by a letter representing the Model property. It is the selection letter of the exported component's name, such as E for Models or F for Tables.

Flow Monitor/Debugger

The Flow Monitor lets you monitor the eDeveloper runtime engine and see how a program is executed in the runtime environment. It helps you understand program execution and identifies problems that may exist in the program logic.

The Flow Monitor utility lets you log and display the command flow (for example, tasks, levels, operation flow, and so on) that occurs when executing a program.

The Debugger utility lets you set break points in the flow. When the Debugger comes to a break point, the flow stops. You can then review each step of that operation, skip that sequence of steps, or ignore the break point.

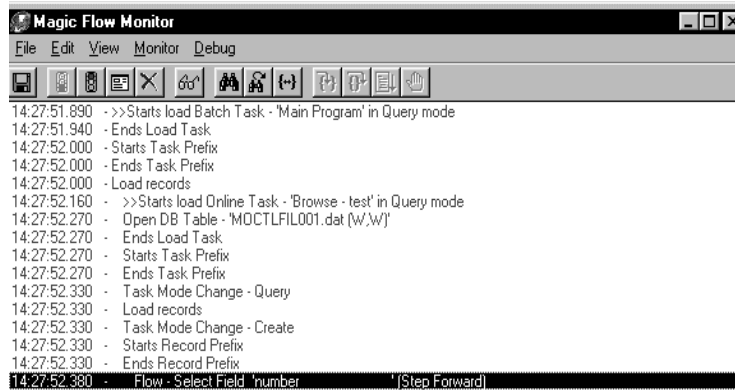


Figure 20-10 The eDeveloper Flow Monitor

The Flow Monitor/Debugger utility runs on a separate thread, parallel to eDeveloper. The Flow Monitor receives activity messages from the eDeveloper Engine.

Flow Monitor Toolbar

The Flow Monitor buttons are displayed below.



Saves the flow



Starts the Flow
Monitor



Stops the Flow
Monitor



Opens the Activity Filter dialog.
Displays the filters that you can set for the Flow Monitor.



Clears the Flow Monitor



Opens the Variables dialog. Displays the program's variables



When the Flow Monitor/Debugger comes to a break point, the Step-In command lets you review a sequence of commands one line at a time.



When the Flow Monitor/ Debugger comes to a break point, the Step Over command lets you skip over from one command sequence to another.



When the Flow Monitor/Debugger comes to a break point, the Continue command lets you ignore the break point.



Sets a break point in the eDeveloper program.

Flow Monitor Message Group Filters

You can specify the activities to be displayed by selecting the following filters:

Task	Task Message registers task activities (For example: open task, close task, task events)
Levels	Flow Level Message registers the task levels For example: record prefix, record suffix, and record main.
Dataview	Dataview Message registers data view actions For example, range, locate, and so on.
Flow Operations	Operations Message registers the flow of operations For example: Select, Link, Block and so on.
Gateways	Gateway Message registers gateway activities For example, gateway connections, SQL statements, and so on.
Transaction Cache	Registers the DML Statements and data content stored in the Transaction Cache of the deferred transactions.

Recompute	Recompute Message registers the activities that cause values to be recomputed. For example: updating a value.
Monitor Browser Tasks Activity	When this filter is selected, you can specify the context information to be displayed in the flow monitor.
Triggered Events	When this filter is selected, the flow monitor displays every event that is processed by the eDeveloper engine.
Log Messages	Registers the log actions. Start - registers the beginning of a compound action (for example, a task prefix). Stop - registers the ending of a compound action.

Flow Monitor Properties

- For activities that have Start/End messages, the user can block either message or both of them.
- You can view the program variables (for example, record number, last reservation, date, and so on) and display them on the flow dataview.
- You can set specific break points for the program. The break points defined are displayed in the Break Points dialog when the program is implemented.
- When a break point is set, The Flow Monitor/Debugger will stop the flow execution at that point. The user can also set a break point by using the FlwMtr function in any given time during the implementation of the program. A counter can also be set to a break point so that the flow will stop after a certain number of times.

- When a break point is encountered, the user can step into, step over, or continue the flow. Step Into lets the user review the sequence of steps one step at a time. Step Over lets the user jump over one sequence series to the next. Continue lets the user ignore the immediate break point.

Flow Monitor Utility for a Server

The Flow Monitor/Debugger can also operate on a server in the Runtime module. Flow monitor messages can be written in the log file, MGFlwMtr.LOG.

To load the Flow Monitor in background mode or on a server, enter the following setting in the Magic.ini file:

```
[MAGIC_FLOW_MONITOR]
LoadMonitor=Y
Monitor2File=Log file name
```

Flow Monitor message filters can be defined in the MGFlwMtr.INI file under the Application Name section, for example [MY APP], by using the keywords listed below. These keywords determine what flow monitor messages appear in the MGFlwMtr.LOG.

For each application, you can define the Flow Monitor message filters.

Keyword Value	Description
BeginMSG=Yes	Begin Message registers the beginning of a sequence of command steps For example, Block
EndMSG=Yes	End Message registers the end of a sequence of commands steps. For example, End Block.
TaskMSG=Yes	Task Message registers task activities (for example: open task, close task, task events, and so on)

Keyword Value	Description
FlowMSG=Yes	Flow Message registers the task levels For example: record prefix, record suffix, and record main.
ViewMSG=Yes	Data View Message registers data view actions For example, range, locate, and so on.
RecomputeMSG=Yes	Recompute Message registers the activities that cause values to be recomputed. For example: updating a value.
OperationsMSG=Yes	Operations Message registers the flow of operations For example: Select, Link, Block and all other Operations.
LogMSG=No	Log Message registers the log actions.

Flow Monitor Support for the Browser Client

eDeveloper provides flow monitor functionality for the browser client. The changing foreground color differentiates between server-side operations and browser client-side logic operations. Foreground colors can be set from the Flow Monitor panel.

The Remote Flow Monitor

The Remote Flow Monitor lets you monitor eDeveloper applications that run on a remote host.

You can start the Remote Flow Monitor by selecting the shortcut in the start\Magic 940 eDeveloper menu group or by clicking the Remote_Monitor.exe executable file, located in the Development folder.

The Remote Flow Monitor is an executable file, independent of the eDeveloper engine, that uses TCP/IP to communicate with the remote eDeveloper engine.

To use the Remote Flow Monitor, the environment settings in the host engine must be set to Remote Flow Monitor=Yes and a Remote Flow Monitor port must be specified.

When the the Remote_Monitor.exe executable file is activated, the Remote Flow Monitor client application appears, as shown in Figure 20-11.

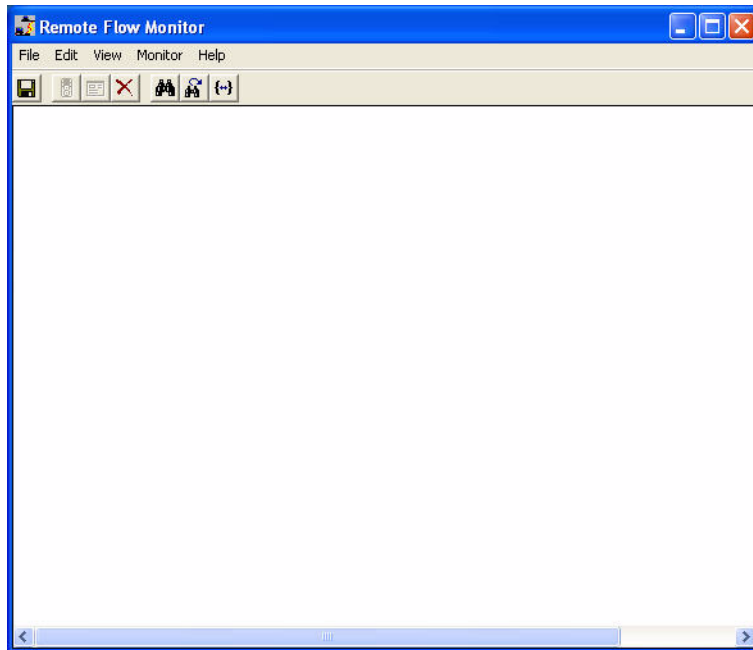


Figure 20-11 Flow Monitor Client

The Remote Flow Monitor client application lets you:

- Define the remote application to monitor
- Start the monitor
- Restrict access
- Save the monitor data
- View statistics
- Set the activity filter
- Define message colors
- Select a specific context

- Set the cache size
- Stop the monitor

Defining the Remote Application to Monitor

Click Setting from the File menu to select the remote application to monitor. The Setting Application dialog opens, as shown in Figure 20-12.

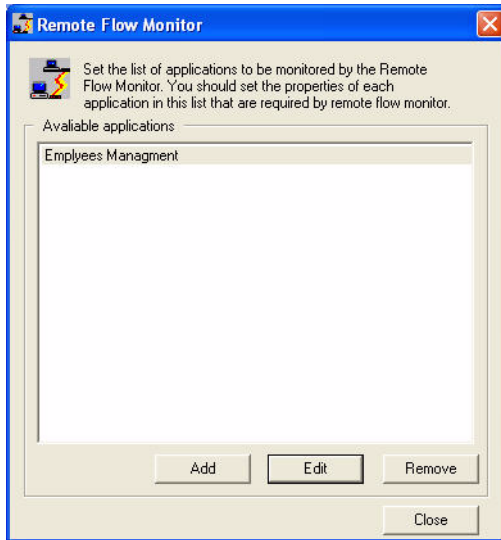


Figure 20-12 The Remote Flow Monitor

From this dialog, you can:

- Add new applications to monitor. When you click Add, the Application Details dialog appears.
- Edit the information of the available applications. Edit is enabled only when eDeveloper is parked on the application name. When you click Edit, the Application Details dialog appears.
- Remove an application from your list. Remove is enabled only when eDeveloper is parked on the application name.

You can add or edit an application from the Application Details dialog box, as shown in Figure 20-13.

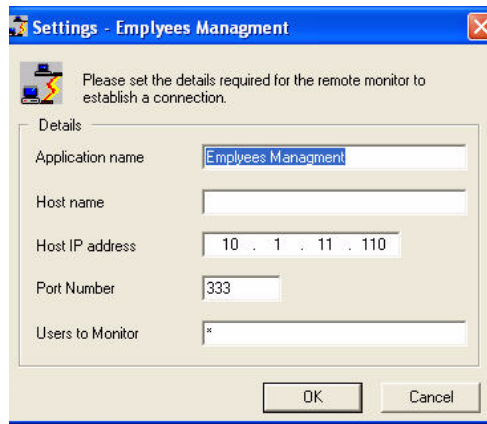


Figure 20-13 Remote Application Details

The Remote Application details are described below:

- **Application Name** - Enter the application name. This name should match the name of the application you are monitoring.
- **Host Name** - The name of the machine on which the application is running. If the host internet protocol is specified, you cannot set the host name.
- **Host IP Address** - The Internet Protocol (IP) of the machine on which the application is running. If the host name is specified, you cannot set the host IP.
- **Port Number** - The port number that the eDeveloper engine uses to monitor connections.
- **Users to Monitor** - This field enables you to filter the messages sent by the server. You can specify several different users, separating their names by commas. Enter an asterisk (*) to instruct the flow monitor to receive all messages.

Starting the Remote Flow Monitor

To start the Remote Flow Monitor, select an application from the Monitor menu. eDeveloper prompts you to enter a user name and password, as shown in Figure 20-14.



Figure 20-14 Starting the Remote Flow Monitor Login Window

When the information is verified, the server begins sending flow monitor messages to the flow monitor client. The information is filtered on the server side according to the information provided on the flow monitor client.

You cannot start the flow monitor without a verified user name and password.

An error message is displayed when the flow monitor fails to connect to the remote engine, as shown in Figure 20-15.



Figure 20-15 Remote Flow Error Message

Restricting Access

You can restrict Remote Flow Monitor access to users with flow monitor rights by specifying the rights in the Flow Monitor Right application property of the Application Property dialog under Security. For more information, see Chapter 15, Application Properties.

Only one Flow Monitor client is allowed to monitor the application at a time. A monitor request by other clients is rejected when the application is being monitored.

Saving the Monitor Data

You can save the result data from the monitor into a text file for use after the monitor is closed. Click Save from the File menu or the Save icon to save the result data from the monitor to a text file. In the Save dialog box, specify the file path and name, and click Save.

Performance Statistics

The monitor can display statistics about the performance of the monitored application. Click Statistics from the View menu. The Statistics window appears, as shown in Figure 20-16.

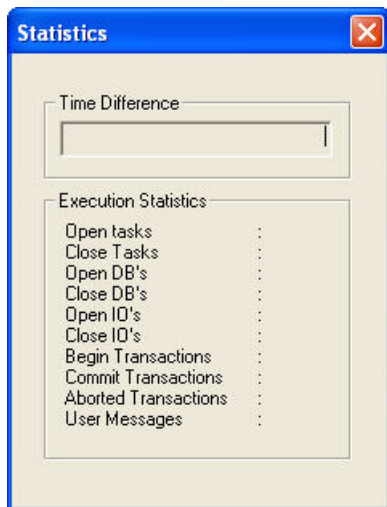


Figure 20-16 Statistics

The Time Difference field displays the amount of time that has elapsed from one selected monitor message to another. Click the monitor message where you want to start and then click the monitor message where you want to stop. The execution statistics display the number of specified actions (such as open

tasks, closed tasks, open databases, and closed databases) that occur between the two selected monitor messages.

Matching the Start and End of a Flow

The flow of an eDeveloper program contains a start and end point. For example, the monitor displays the start of the record suffix and the end of the record suffix. When a line position in the monitor is selected at the beginning of a flow, you can jump to the end of the flow by clicking Match Start/End from the Monitor menu. In our example above, eDeveloper will move the highlight line to the end of the record suffix. This functionality lets you view and understand what is being executed in a specific section of the flow.

Activity Filter

Click Options from the View menu to set monitor data filters. The Activity Filter dialog appears, as shown in figure 20-17.

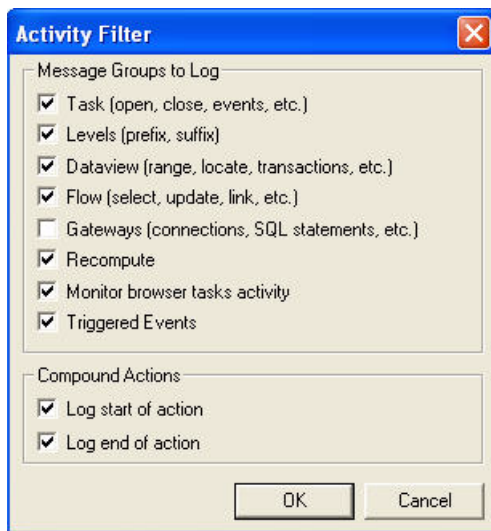


Figure 20-17 Activity Filter

For more information about the Activity Filter, see the Flow Monitor/Debugger section in this chapter.

Color Management

You can customize how the monitor messages are displayed by specifying colors for the different message types, as shown in Figure 20-18.

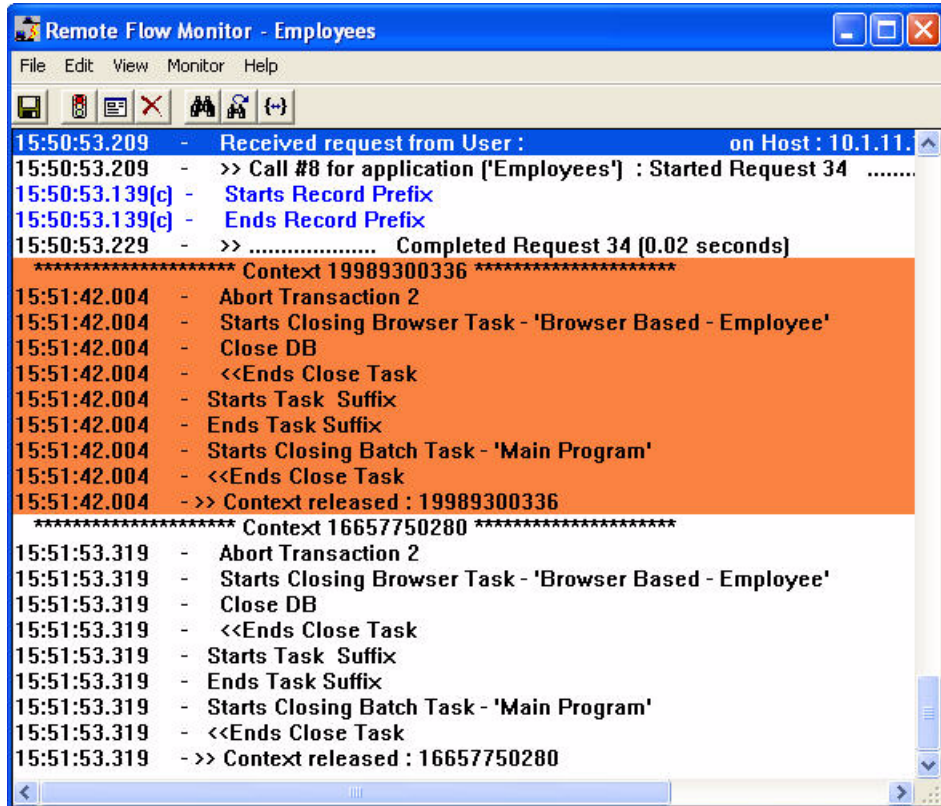


Figure 20-18 Selected Colors for Monitor Data

The monitor message types are:

- Engine activity - Select a color for server-side operations.
- Browser activity - Select a color for client-side operations, such as browser-based programs.
- Context activity - Select a background color to display a change of context. The selected background color will replace the default color, white.

Context Manager

Click Context Manager from the View menu to select a specific context that you want to monitor. The Context Manager window appears, as shown in Figure 20-19.

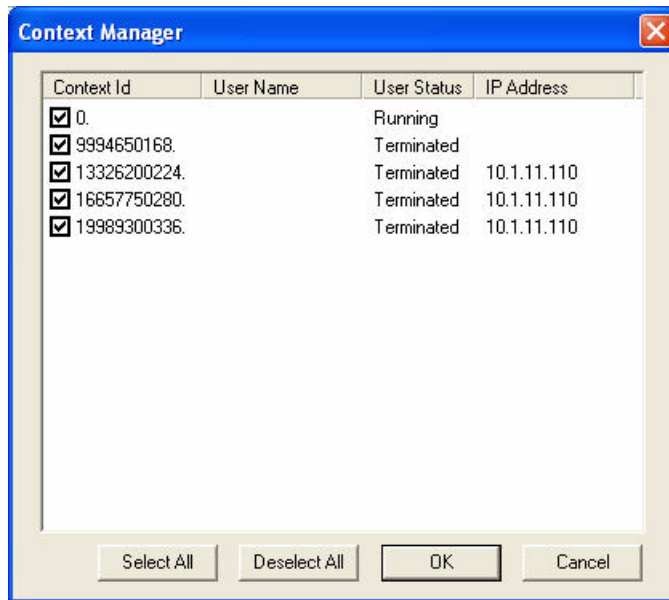


Figure 20-19 Context Manager

You can view the context number of the browser task in the Internet Explorer's address bar.


The Context Manager window has the following columns:

- Context ID - Select to display the context and the context number.
- User Name - Displays the name of the user running the context.
- User status - Displays the status of the context, whether the context is running or whether it has been terminated.
- IP Address - Displays the IP Address of the context.

Setting the Cache Size

The Remote Flow Monitor stores the incoming message codes in the client cache. You can specify the size of this cache by clicking Cache Size from the View menu. The minimum size is 100 KB and the maximal size is 9999 KB. The cache mechanism of the Remote Flow Monitor is First In First Out (FIFO).

Stop the Monitoring

To stop monitoring, click Stop Monitoring from the Monitor menu or click the Stop  icon.

The Profiler

The Profiler collects performance and coverage information on your application and lets you analyze that information to learn about the runtime characteristics of your application. You can then use this data, including information on task usage, elapsed times, compute-bound tasks, I/O-bound tasks, and so on, to help you identify and resolve bottlenecks in the application.

Improving performance using the Profiler consists of four steps:

1. Collect data
2. Analyze results
3. Focus on problem areas
4. Fix problems

Profiler Operation

To activate the Profiler, you must set up an operating system variable.

For Windows or UNIX this is MGPROF.

For VMS this is MAGIC\$TRACE.

For example, in DOS, enter set MGPROF=1 at the system prompt.

Profiler Output

The Profiler creates two output files for each program that is run. The first file is the Program Execution Trace file that includes information about the Call tree in the application (including events), and exact timing about the various elements in the task execution. The second file includes information about files used in the tasks. The profiler output files are regular ASCII files that can be edited, printed, or even read into a spreadsheet program for the analysis phase.

Program Execution Trace File

The Program Execution Trace file displays the calling sequence of the program. The file consists of a line for the beginning of each task and a line for the end of each task. The hierarchy of the tasks is shown through indentations from the left margin.

The Program Execution Trace file contains the following columns:

#	Column Name	Contents	Remarks
1	#	Line Number	
2	Task Number	Program Number in the Program repository, or a fully qualified Task Number for tasks.	
3	Task Name	Program name or Task name.	
4	Start/End Task	Start - for the beginning of task execution. End- for the end of task execution.	This column is displayed in a hierarchical order to show the calling sequence.

#	Column Name	Contents	Remarks
5	Event	Event - if the task was executed as an event.	
6	Total Task Time	Total elapsed time from beginning to end of task.	Elapsed time includes the elapsed time for all called tasks.
7	Resident	Y - if the task is resident.	
8	Sort Time	Elapsed time for sorting the dataview.	
9	# of Open Files	Number of files that are defined in the task.	
10	Task Open Time	Elapsed time to load the task for execution.	Task open time includes the time to open the files for the task.
11	Task Close Time	Elapsed time to close the task after execution.	Task close time includes the time to close the files of the task.
12	Open Files Time	Elapsed time to open the tables for the task.	
13	Close Files Time	Elapsed time to close the tables of the task.	
14	# of Updated Records	Number of View updates.	
15	Average Update Time	Average time for updating a single dataview.	

#	Column Name	Contents	Remarks
16	# Load Records	Number of View loads.	
17	Average Load Time	Average time for loading a single dataview.	

The output table name of the Program Execution Trace table is MGPROFT.LOG.

Opened Files Trace File

The Opened Files Trace file lists the tables that are defined within a task and that eDeveloper opens for any reason, and also that eDeveloper is forced to open or reopen because of a change in access and share modes.

The Opened Files Trace file contains the following columns:

#	Column Name	Details
1	#	Reference to the line number in the Program Execution Trace file.
2	Task Number	Program number in the Program repository, or a fully qualified Task number.
3	File	File number.
4	New	New- if this is the first time the file was opened.
5	Original Open Mode	Access, Share, and Open Mode as defined in the parent task.
6	Open Mode	Access, Share, and Open Mode as defined in the task.

The output name of the Opened Files Trace file is MGPROFM.LOG. To change the file name in DOS or UNIX use the command
set MGTCNF=<file name>
and in VMS use the command
set MAGIC\$TRACE_CNF=<filename>.

Note: All measured times are in 55 milliseconds resolution.

The OEM2ANSI Utility

A conversion utility that lets you convert an entire series of external text files from ANSI to OEM and from OEM to ANSI. The purpose of this utility is to convert various files used in applications that may have different standards of character mapping, for example, exporting a text file from an eDeveloper Version 9 application to an application of a previous Magic versions.

The ODBC Check Driver Utility

The ODBC Check Driver utility is included with the ODBC driver. Running this program tests the ODBC data source and prints a list of functions that are produced by the driver. The test will tell you whether or not the driver can be used with the eDeveloper Database Gateway for ODBC.

For more information, refer to Chapter 25, SQL Considerations.

The MakeKey Utility

The MakeKey utility lets you create a feature license that can be checked out and in by other users of an eDeveloper application.

The MakeKey utility works best when run under the installed license server directory. This utility prompts the user to enter the information required for creating a new license entry.

The modules for the feature license implementation are the LMChkOut function, LMChkIn function, and the MakeKey utility.

The **Makekey.exe** file is provided with the installation of eDeveloper. This file will take you through a series of steps needed for creating your new license entry.

The Table Conversion Utility

Table conversion is a process eDeveloper starts automatically if the physical structure of a table is modified and the data table exists. For example, changes such as adding a column to a table, removing a column from a table, changing a column's attribute, changing index specifications, or modifying a type used in a table, each automatically launch the table conversion process.

It is important to note that if you change an index definition from non-unique to unique, the conversion process deletes all the rows that meet the duplicate index condition.

The table conversion process starts as you attempt to exit from a table row in the Table repository, or exit from the Table repository, or as a result of a Model repository conversion sweep.

eDeveloper maintains the structure of the table as it exists before the changes are made to it, along with the new structure, until the table row is left or until the table repository is left. If you cancel the conversion process, an inconsistency between the physical and logical structures of the table may remain. In this case, eDeveloper will not be able to perform an automatic conversion at a later stage and may report an error the next time it accesses the table.

The conversion process consists of the following stages:

1. eDeveloper opens the Confirm Convert Operation dialog. Select Yes to convert the table automatically. If you Cancel the operation no conversion will take place (in which case you may have to undo the modifications manually).
2. If you confirm the convert operation, and depending on the database used, eDeveloper opens the Confirm Backup Operation dialog. If you want to preserve a copy of the original table (as it was before it was modified), select Yes. The process creates a BCK file (for example, in **REM: GET WIN EX** DOS, DMBCK001.DAT) in the current directory, provided there is room on the disk.
3. If you confirm the convert operation, eDeveloper displays the Table Convert dialog. In this dialog you can request eDeveloper to rearrange the database table according to one of its previously existing indexes. Index 0 arranges the rows in their physical sequence. If you want to arrange the table according to a index, either type the index's sequential number (from the Index repository), or zoom to the Index list to select a index. It is advisable to rearrange a table according to the index that is most frequently used, to enhance performance when scanning the table or in any I/O operation with the selected index.
4. After you select Yes in the Table Convert dialog, eDeveloper carries out the conversion. A Convert window appears and displays the number of rows converted so far. When the window disappears, the

table is fully converted.

5. When possible, eDeveloper uses underlying database utilities for table conversion, for example using the SQL ALTER statement when the underlying database is SQL. Refer to the Magic Guide to the SQL.

Magic Flat File

The Magic Flat File (MFF) lets you deploy a database-independent eDeveloper application file. The application, which is created as a binary file, can be stored anywhere on the user's computer system and is used for deployment only.

MFF files are meant to be used in a multi-platform environment. For example, you can transfer an MFF from Windows NT to UNIX without importing or exporting the application.

You can save an existing eDeveloper application as an MFF by clicking the Save as MFF command from the File menu.

Print Data Wizard

For online tasks, you can enable the Print Data Wizard by setting the Allow Print Data task control property, in the Task Control dialog, to Yes. When the Print Data Wizard is enabled, the end-user can print data displayed on the screen at runtime.

Runtime Operations

The Print Data Wizard lets you specify how to print data displayed on the screen at runtime by defining the:

- Output Type
- Column Order
- Delimiters

Output Type

Click Print Data (**CTRL+G**) from the Options menu in runtime to start the Print Data Wizard. The Print Data dialog box opens, as shown in Figure 20-20.

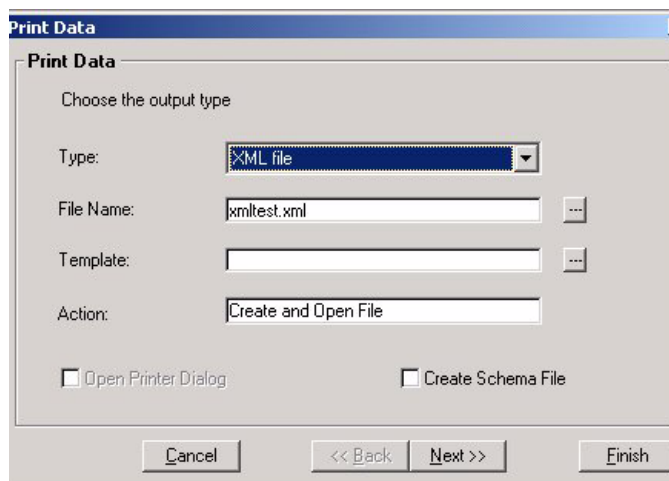


Figure 20-20 Print Data Dialog

You can print the data displayed on the screen in HTML, XML, or Text format. Data can be sent to the printer, printed to file, or opened in an external application. To send data to the printer, select the Create and Print file action and Open Printer Dialog.

When printing data to a file, select the Create action and specify the file path name. If no path is specified, the file is created in the Magic home directory. Search for the file path name by zooming from the field.

You can display the data in an external application assigned to the data's file type by selecting the Create and Open File action. For example, an HTML file that has a .DOC extension will be opened in Microsoft Word.

You can specify a template for an HTML or XML file. For an XML template, select an XSL file. Template files specified in this field override the default files selected in the Print Data HTML and XML Template environment settings. If this field is left blank, eDeveloper automatically uses the default HTML and XML template files.

You can create an XML Schema Definition (XSD) file when you specify the data to be displayed in XML format. When the Create Schema File check box is selected, eDeveloper creates the XSD file in the location specified in the Print Data XML Template environment setting. The XSD file has the same name as the XML file. For more information, see XSD Data Type.

Click Finish to print the data without specifying the column order or delimiters.

If you want to set the column order and delimiters before the data is printed, click Next to open the Column Selection dialog box, as shown in Figure 20-21.

Column Order

If you do not want to print data from a column, enter 0 next to the column name.

You can change the column order by entering a different order number next to the column name. eDeveloper adjusts the column order accordingly.

Click Finish to print the dataview with the selected column order.

For text files, you can continue the process and click Next to select the delimiter character that appears in the data printout. eDeveloper uses the comma and single quotation mark for the default value and string delimiters.

Delimiters and String Identifiers

You can select a delimiter for the text file or enter a custom delimiter by selecting Other in the Delimiter and String Identifiers dialog, as shown in Figure 20-21

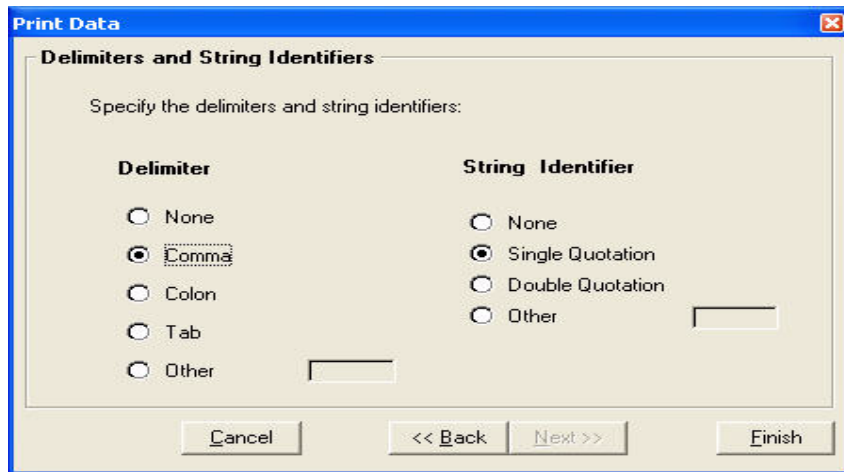


Figure 20-21 Delimiters and String Identifiers

You can also select a string identifier, a single or double quotation mark, or enter a custom string identifier by selecting Other.

Click Finish from the Delimiter dialog box to print the data as defined.

The printed data always appears in table format, even if invoked from a screen mode task. Printing data is not available for multi-marked selections.

Runtime Behavior

eDeveloper prints the current screen control value for the:

- Radio button
- Edit
- List
- Combo box
- RTF

If the Check box is selected, eDeveloper prints True. If it is not selected, eDeveloper prints False.

If a control has an expression that does not provide a control name, the end-user cannot print the control value.

If there is no data, the Print Data Wizard prints:

No data printed.

If eDeveloper cannot find the template or there is an error in it, the Print Data Wizard prints:

Error in the printout template.

If there are no style definitions, the print data output will not have a defined style.

XML Template Structure

When XML is specified as the print data output type, eDeveloper displays the data in an XML template where the XML tags are labeled according to the control name.

The XML template file should include a reference to the XSL file. Magic uses the default template file when no template file is specified. If a template file is specified, eDeveloper adds a reference to the XSL file, as displayed below:

```
<?xml-stylesheet type="text/xsl" href="sample.xsl" ?>
<Print_data>
</Print_data>
```

eDeveloper enters the records between the <Print_data> tags and the <Record> tag for each record, as shown below:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<Print_data>
<Record>
<Emp_ID>1</Emp_ID>
<Name>David</Name>
<Email>david@mse.com</Email>
</Record>
<Record>
<Emp_ID>2</Emp_ID>
<Name>Dan</Name>
<Email>dan@mse.com</Email>
</Record>
<Record>
<Emp_ID>3</Emp_ID>
<Name>John</Name>
<Email>John@mse.com</Email>
</Record>
</Print_data>

```

XSD Data Type

When the Create Schema File check box is selected, eDeveloper creates the data in the XML file according to the proper XSD data types, described below.

Magic Attribute	XSD Data Type
Alpha	string
Logical	boolean

Magic Attribute	XSD Data Type
Numeric without floating point	integer
Time	time
Date	date
Memo	string

Note:

eDeveloper's date and time format is converted to the XSD date and time format.

The XSD data type is determined according to the picture. For numeric, date, and time, the picture can contain only numbers and dots. If the picture contains other characters, eDeveloper creates the XSD file with the String data type.

XSD Header

The XSD header is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by eDeveloper 9.4 (http://www.magicsoftware.com)-->
<schema targetNamespace="http://www.magicsoftware.com/printdata94"
xmlns="http://www.w3.org/2001/XMLSchema">
```

Elements

All elements the user selected for the XML file should be represented in the XML Schema Definition. When the element name contains an illegal character, it is replaced with an underscore, as shown below.

```
<xs:element name="element_name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Note:

The `element_name` is replaced by the control name.

The **xs:type** is set according to the conversion table in conjunction with the control attribute.

The **value** is replaced with the control picture.

The Record Complex Type is specified as:

```
<xs:complexType name="Record">
  <xs:sequence>
    <xs:element ref="element1"/>
    <xs:element ref="element2"/>
    <xs:element ref="element3"/>
    <xs:element ref="element14"/>
  </xs:sequence>
</xs:complexType>
```

XML Example

Elements are replaced with element names in the order they appear in the XML file, as displayed below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by eDeveloper 9.4 (http://www.magicsoftware.com)-->
<schema targetNamespace=http://www.magicsoftware.com/printdata94
  xmlns="http://www.w3.org/2001/XMLSchema">
```

```

<xs:element name="EMPLOYEE_ID">
  <xs:simpleType >
    <xs:restriction base="xs:integer">
      <xs:maxLength value="5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="FIRST_NAME">
  <xs:simpleType >
    <xs:restriction base="xs:string">
      <xs:maxLength value="30"/>
    </xs:restriction >
  </xs:simpleType>
</xs:element>
<xs:element name="HIRE_DATE">
  <xs:simpleType >
    <xs:restriction base="xs:date">
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="LAST_NAME">
  <xs:simpleType >
    <xs:restriction base="xs:string">
      <xs:maxLength value="30"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="PHONE_NUMBER">
  <xs:simpleType >
    <xs:restriction base="xs:string">
      <xs:maxLength value="15"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:complexType name="Record">
  <xs:sequence>
    <xs:element ref="EMPLOYEE_ID"/>
    <xs:element ref="FIRST_NAME"/>
    <xs:element ref="LAST_NAME"/>
    <xs:element ref="PHONE_NUMBER"/>
    <xs:element ref="HIRE_DATE"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

HTML Template Structure

When HTML is specified as the print data output type, eDeveloper displays the data in the HTML template below:

```
<html>
<head>
<title> </title>
</head>
<body>
  <MGTABLE>
</body>
</html>
```

You can define the MGTABLE style tags, RowStyle and ColumnStyle, in the HTML Template as described below:

- RowStyle:
 - All - eDeveloper creates a specific style for each row and title.
 - EvenAndOdd - eDeveloper creates two styles, an MG_Even_Row and an MG_Odd_Row.
 - Equal - eDeveloper creates the same style for all rows and titles.
- ColumnStyle:
 - All - eDeveloper creates a specific style for each column.
 - Equal - eDeveloper creates the same style for all columns.

The example below has four columns in the table. The MGTABLE is defined as < MGTABLE RowStyle=ALL ColumnStyle=ALL >:

```
<table border="1" width="100%" class="MG_TABLE">
<THEAD>
  <tr >
```



```

<td width="25%" class="MG_TITLE1">t1</td>
<td width="25%" class="MG_TITLE2">t2</td>
<td width="25%" class="MG_TITLE3">t3</td>
<td width="25%" class="MG_TITLE4">t4</td>
</tr>
</THEAD>
<TBODY>
<tr class="MG_ROW1">
<td width="25%" class="MG_DATA1">d11</td>
<td width="25%" class="MG_DATA2">d12</td>
<td width="25%" class="MG_DATA3">d13</td>
<td width="25%" class="MG_DATA4">d14</td>
</tr>
<tr class="MG_ROW2">
<td width="25%" class="MG_DATA1">d21</td>
<td width="25%" class="MG_DATA2">d22</td>
<td width="25%" class="MG_DATA3">d23</td>
<td width="25%" class="MG_DATA4">d24</td>
</tr>
</TBODY>
</table>

```

The eDeveloper Version 9.4 template tags below replace the template tags from previous Magic versions.

- System Date and System Time (##/##/#### HH:MM) replace MGDate.
- The Owner parameter specified in the Magic.ini file replaces MGOwner.

- The eDeveloper logged-in user replaces MGUser.

Tools Infrastructure

The Tool Infrastructure lets you create your own tools, wizards and batch processes that can run from a dynamic menu in toolkit mode. The Tools menu is displayed only when an application is open.

Building the Menu

The Tools menu definition is loaded with the eDeveloper toolkit. Any modification of the [TOOLS_MENUS] section in the Magic. ini file takes effect only in the next session.

The menu entry order in the Tools menu and submenus are determined by the order the menu entries are entered in the [TOOLS_MENUS] section.

The Tools menu entry is defined in the Magic.ini file by the following syntax:

Menu Name = menu type, menu caption, parent menu name, MFF path\command, access key, pre-operation command file, post-operation command file, image for tool number and tool group

If the Tools menu syntax is invalid, the option is not displayed in the menu. If there are no valid tool entries found, the Tools menu is not displayed.

Menu Type

You can select a character below to specify a tool type:

A=Application	This option creates a menu entry that activates an eDeveloper application in a flat-file format.
O=OS Command	This option creates a menu entry that activates a defined OS command.
M=Submenu	This option creates a submenu entry point.

A=Application	This option creates a menu entry that activates an eDeveloper application in a flat-file format.
S=Separator	This option creates a menu separator.

Menu Caption

The menu caption lets you define a menu name.

The menu caption can include an ampersand character (&) to set the accelerator key. The menu caption is mandatory for Application, OS command, and Submenu entries but is not required for the Separator.

Parent Menu Name

A string defining the parent menu. If the Parent Menu Name parameter is empty, the menu caption appears in the Tools menu.

MFF Path and Command

When the menu type is Application, the MFF Path/Command parameter must specify the location of the application flat file.

Before a flat-file application is loaded, the current application will be closed. The general information about the application is set as global parameters.

If no MFF path is defined, the menu entry is regarded as invalid and will not be part of the menu structure.

When the menu type is OS Command, the MFF parameter specifies the OS command to be executed.

Logical names are supported for both the MFF Path and Command parameters.

Access Key

This parameter defines the access key of the defined menu entry. This is an optional parameter that applies only to Application and OS Command menu types.

Pre-Operation Command File

The file name that lists a sequence of operations to be performed before the current application is closed. For information about the command operations, see Operation Commands on page 1287. This parameter supports logical names.

Post Operation Command File

The file name that lists a sequence of operations that will be performed before the current application is reloaded after the flat-file application closes. For information about the command operations, see Operation Commands on page 1287. This parameter supports logical names.

Tools Menu Example

[TOOLS_MENUS]

Menu1 = A,&Search and Replace

Tool,,Add_On\SearchReplace.mff,Ctrl+W,pre.txt, post.txt

Menu2 = O,&Notepad, ,notepad.exe magic.ini,

Menu3 = S,,,,,

Menu4 = M,S&ub Menu,,,,

Menu5 = M,&My Wizards,Menu4,,,

Menu6 = A,Wizard &1,Menu5,Add_On\wizard1.mff

Menu7 = A,Wizard &2,Menu5,Add_On\wizard2.mff

Menu8 = A,Wizard &3,Menu5,%Tools%wizard3.mff

The example is displayed in Figure 20-22.

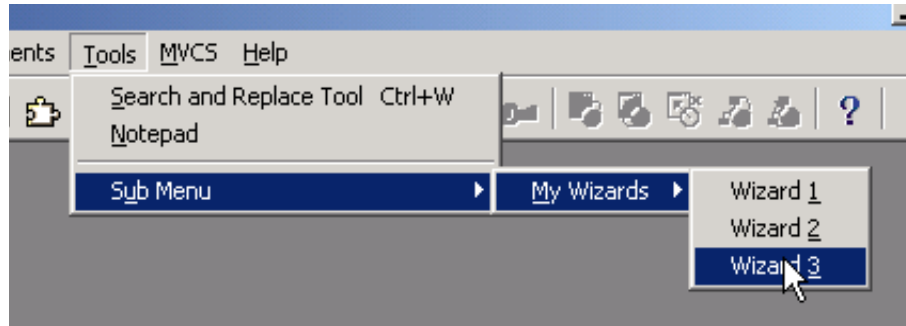


Figure 20-22 Tools Menu Example

Activating the *Search and Replace Tool* menu option results in the following steps:

1. Runs the operations listed in the pre.txt operation file (for example, exporting the application).
2. Closes the current application (for example, customers.mcf)
3. Opens the SearchReplace flat file in the current eDeveloper engine.
4. When the SearchReplace.mff closes, eDeveloper automatically returns to the original application, customers.mcf.
5. Returns the list of operations in the post.txt operation file (for example, importing an application).

When you select the Notepad menu option, Windows Notepad opens with the Magic.ini file.

Operation Commands

The syntax for pre-operation and post operation commands are:

operation, main information, number, switches

The pre-operations and post operations are:

- Export - An internal export of an application

- Document - A document export of an application
- Import - importing an eDeveloper application
- MFF - Saving an application as a flat file
- Application - Switching between defined applications
- Getdef - An SQL Get Definition operation
- OS - An OS command
- Simulate - Simulating keyboard strokes (for example, parking on a repository object)

Export Operation

This operation has the following parameters:

Export, <file name>, <switches>

File Name

A valid file name and path for the export file. The file name can support logical names.

Switches

Optional instructions for the Export Document operation.

- Repository=X

This switch specifies which repository should be exported. Select the character representing the repository as listed below:

- M - Models
- T - Tables
- P - Programs
- H - Help screens
- R - Rights
- U - Menus

- C - Components
- D - Application data (properties)
- A - Application

- Range=

This switch sets the range of objects that can be exported within a defined repository.

The range of values are defined by the From and To values, separated by a hyphen (for example, 13-18).

When a single entry is exported, the parameter has only one entry number or the same entry number for the From and To values (for example, 7 or 7-7).

When the range is not defined or invalid, eDeveloper exports the selected repository.

Range values are irrelevant when exporting Menus, Rights, Application data or an entire application.

You can export the parked object by using the Range switch. When the Range is set to Range=@, the engine exports the parked object. This switch is handled only in conjunction with the Repository switch.

For example: **Export, myobject.exp, Repository=A (All), Range=@**

To export a handled object of a specific repository, the Repository switch must indicate the repository.

If the engine did not park on an object or handle an object within a specified repository, the result file will be empty.

- WithModels=Y\N

When this switch is set to Y, exporting tables and programs will include the models used in the export file. The default value is N.

For example: Export, %Path%AllTables.exp, Repository=T WithModels=Y

Export Document Operation

This operation has the following parameters:

Document,<file name>,<switches>

File Name

The file name and path used as the destination file for the output result.

Switches

Optional instructions for the Export Document operation.

- Repository=X

This switch specifies which repositories can be exported. Select the character representing the repository as listed below:

- M - Models
- T - Tables
- P - Programs
- H - Help screens
- R - Rights
- U - Menus
- C - Components
- D - Application data (properties)
- A - Application

- Range =

This switch sets the range of objects that can be exported within a defined repository.

The range of values are defined by From and To values, separated by a hyphen (for example, 13-18).

When a single entry is exported, the parameter has only one entry number or the same entry number for the From and To values (for example, 7 or 7-7).

When the range is not defined or invalid, eDeveloper exports the selected repository.

Range values are irrelevant when exporting Menus, Rights, Application data or an entire application.

You can export the parked object by using the Range switch. When the Range is set to Range=@, the engine exports the parked object. This switch is handled only in conjunction with the Repository switch.

For example: **Export, myobject.exp, Repository=A (All), Range=@**

To export a handled object of a specific repository, the Repository switch must indicate the repository.

If the engine did not park on an object or handle an object within a specified repository, the result file will be empty.

- IncludeComponents=Y\N

When this switch is set to Y, exporting the defined repository include entries from the Components repository.

- InternalValues=Y\N

When the switch is set to Y, exporting the defined repository displays its values as fixed internal eDeveloper values, not translated through the Mgconstw file.

- TemplateFile=

The file name and path of the document template used for the Export Document operation.

If TemplateFile is not defined, the export document uses the value entered for the Document Template environment setting. This switch supports logical names.

For example: Document,%Path%Table.exp,Repository=T InternalValues=Y TemplateFile=My_Doc_std.eng

Import Operation

This operation has the following parameters:

Import, <file name>, <switches>

File Name

The file name and path used as the source file for the import process.

Switches

Optional instructions for the Import operation:

- ImportFolderInfo=Y/N

When the switch is set to Y, the selected objects are imported into their defined folder. The default value is N.

- OverwriteMainProgram=Y/N

When the switch is set to Y, the imported Main Program overwrites the existing one. The default value is N.

- OverwritePulldownMenu=Y/N

When the switch is set to Y, the imported pulldown menu definition overwrites the existing one. The default value is N.

- OverwriteContextMenu=Y/N

When the switch is set to Y, the imported context menu definition overwrites the existing one. The default value is N.

- OverwriteTarget=[repository entry number]

You can use this switch to import a specific export object to replace the target object defined by the OverwriteTarget switch for a repository.

For example: **Import, myobj.exp, Repository=P, OverwriteTarget=17**

If the target does not exist, the Import operation handles the import object as if the switch has not been set, placing the imported object at the end of the repository.

The Main program cannot be overwritten with this switch. You can only overwrite the Main Program by using the OverWriteMainProgram switch.

Menus, Application Data, and Rights are always imported in their entirety, and the OverwriteTarget switch is ignored.

If the target object is found, the ImportFolderInfo switch is ignored.

- AutoCheckOut=Y/N

When this switch is set to Y and the application is defined in a team development environment, the imported repositories are automatically checked

out.

For example: Import, a.exp, ImportFolderInfo=Y OverwriteMainProgram=Y OverwritePulldownMenu=N

Save as MFF Operation

This operation has the following parameters:

MFF, <file name>

File Name

A valid file name and path used as the MFF destination file.

Open Application Operation

This operation has the following parameters:

Application, <Application number>

Application Number

The application identifier from the application list. Any consecutive operation is performed on the most recently opened application.

Get Table Definition Operation

This operation has the following parameters:

Getdef, <database name>, <table name>, <switches>

Database Name

The name of the database that determines the table definition. This name should be defined in eDeveloper Database settings.

Table Name

The full table name, including the owner name.

FolderName Switch

The name of the folder where the table is created.

For example: Getdef, Mssql, dbo.GA_table, FolderName = folder1

This switch is optional.

OS Command Operation

This operation has the following parameters:

OS, <OS command>, <switches>

OS Command

The OS command to be executed.

Switches

Optional instructions for the OS Command operation are:

- Wait=Y/N

If set to Y the next operation waits until the OS Command operation is completed.

- Show =[Hide, Normal (default), Maximize, Minimize]

This switch determines how the DOS prompt window should be displayed.

For example: OS, notepad, wait=Y show=hide

Simulate Operation

This operation has the following parameters:

Simulate, <collection of instructive tokens>, <switches>

The three tokens below instruct the eDeveloper engine to either park on an object or simulate keyboard strokes. A semicolon is used to separate the token values.

- WIN - This token instructs the eDeveloper engine to park on a repository object.

Each repository is defined by a string as follows:

- MDL_REP - Models

- TBL_REP - Tables
- PRG_REP - Programs
- HLP_REP - Help screens
- RGT_REP - Rights
- MNU_REP - Menu
- CMP_REP - Components

These strings are suffixed by a number representing the object number in the repository.

For example: Simulate, WIN=MDL_REP#15, parks on the fifteenth model entry in the Model repository.

- TXT - This token instructs the eDeveloper engine to simulate keystrokes.

Simulate, TXT= <text>, enters the text at its current location.

For this token, <free text> starts from the character immediately following the = sign. A space is considered a character.

- KEY - This token instructs the eDeveloper engine to simulate a key combination. The supported key combinations are:
 - F1 ... F12 - As is, prefix by Shift+ or Ctrl+ or Alt+
 - Ctrl+<any character>
 - Home - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - End - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - Up - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - Down - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - PgUp - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - PgDn - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - Tab - As is or could be prefixed by Shift+ or Ctrl+ or Alt+
 - Esc - As is or could be prefixed by Shift+ or Ctrl+ or Alt+

- Del - As is or could be prefixed by Shift+
- Back - As is or could be prefixed by Alt+
- Enter - As is or could be prefixed by prefix by Ctrl+
- Space - As is or could be prefixed by Ctrl+
- Ins - As is or could be prefixed by Shift+ or Ctrl+
- Plus
- Minus
- Clear
- Help

For example: Simulate, KEY=SHIFT+F10 ; KEY=Enter

Global Parameters Information

A running tool application can retrieve information about the current eDeveloper application. Use the GetParam function to retrieve preset global values about the handled application through the global value names listed in the table below.

Parameter Name	Value
MG_ApplicationIDX	The entry number of the application in the systems list
MG_ApplicationPrefix	The application prefix
MG_ApplicationName	The name of the application
MG_CurrentObject	A number representing the internal serial number of the object on which the developer is parked.
MG_CurrentObjectISN	The number representing the internal serial number of the object on which the developer is parked.

Parameter Name	Value
MG_CurrentObjCheckedIn	The engine returns True when the last parked or handled object is checked in, or False when the last parked or handled object is checked out. When the application is not defined for team development, the setting does not display a value.
MG_CurrentObjCheckedOut	The engine returns True when the last parked or handled object is checked out or False when the last parked or handled object is checked in. When the application is not defined for team development, the setting does not display a value.
MG_CurrentRepository	<p>A number representing the repository that developer was parked on prior to activating the tool application.</p> <p>The Repository identifiers are:</p> <ul style="list-style-type: none"> 1 – Models 2 – Tables 3 – Programs 4 – Help Screens 5 – Rights 6 – Menu 7 – Components 9 - Application Properties

Parameter Name	Value
MG_TeamDev	Y when the application is in team development mode. N when the application is not in team development mode.
MG_ToolEntry	A string representing the INI entry of the activated tool.
MG_TotalModels	The total number of models, not including components For example, if an application has 20 models this global parameter returns the number 20
MG_TotalTopModels	The total number of models set with no folder
MG_TotalTables	The total number of tables, not including components
MG_TotalTopTables	The total number of tables set with no folder
MG_TotalPrograms	The total number of programs, not including components
MG_TotalTopPrograms	The total number of programs set with no folder
MG_TotalHelpScreens	The total number of help screens, not including components
MG_TotalTopHelpScreens	The total number of help screens set with no folder
MG_TotalRights	The total number of rights, not including components
MG_TotalTopRights	The total number of rights set with no folder
MG_TotalComponents	The total number of components

Parameter Name	Value
MG_TotalTopComponents	The total number of components set with no folder

Automatic Processing

The toolkit engine can perform automatic operations in the background and toolkit modes. Operations are listed in a text file defined in the Magic.ini file in the [MAGIC_ENV] section as AutomaticProcessingSequenceFile=file name

When an eDeveloper generator engine loads in background mode, it automatically executes the operations listed in the AutomaticProcessingSequence file.

If the engine is in background mode and is not loaded as a server, it will automatically shutdown after the last listed operation.

Automatic processing can be useful for running a series of automatic operations at specified times.

The name of the AutomaticProcessingSequence file can be defined by using a logical name.

Automatic Processing Sequence Example

To automatically export application number 2, add AutomaticProcessingSequenceFile=c:\tools\operations.txt to the Magic.ini file.

Enter the following lines in c:\tools\operations text file:

```
Application,2,
Export, %Path%MyApplication.exp, Repository=A WithModels=Y
```

Create a shortcut with the proper ApplicationStartup environment setting, user name and password for executing the automatic processing file.

An example of a shortcut target is:

```
C:\program files\magic 940\MGgenw.exe /ActivateRequestsServer=N /  
ApplicationStartup=B /user=guest /password=guest /  
AutomaticProcessingSequenceFile=AutoProcess1.txt
```

Monitor Utility

The Monitor Application utility lets you monitor your enterprise servers for a selected broker. You can open the utility by double-clicking the MGRQMonitor executable file from the 9.40 subdirectory under Magic or by selecting **Monitor** from the context menu when you right-click the Magic Broker icon on the Windows status bar. The windows that constitute the Monitor Application screen are described below.

Enterprise Servers

This window displays all active servers. You can load or close an enterprise server by clicking **Load Enterprise Server** or **Shutdown Enterprise Server** from the **Actions** menu.

The columns for the Enterprise Servers window are:

Host/Port - Internet Protocol (IP) or host name

PID - The client's process identifier

Application - The application name

Status - The request status

Current Threads - The current number of requests

Peak Threads - The peak number of threads that the server requests from the broker

Max Threads - The maximum number of threads that the server requests from the broker

Request Count - The number of requests processed

Context Count - The number of contexts available

When shutting down an enterprise server, a warning box appears. If you click OK for a server that is running, the server shuts down. If the server is not running, the Monitor utility displays an error message.

Contexts

This window displays the context threads for the selected request broker. You can close a selected context by clicking **Terminate Context** from the **Actions** menu.

When closing a context, a warning box appears. If you click OK for an active context, the context is closed. If the context is not running, the Monitor utility displays an error message. When a context is closed, the cursor parks on the next context.

Requests

This window displays the requests issued to the server by the selected request broker. You can close the selected request by clicking **Delete Request** from the **Actions** menu.

The columns for the Request window are:

Application/Program - The application or program executing the request

Client - The Internet Protocol (IP) or computer name of the client submitting the request

Status - The request status, such as Failed Request, Completed Request, Request in Progress, or Pending Request

Submission Time - The time the request was submitted according to the broker

Elapsed - The time in seconds that elapsed since the request was submitted to the broker until the request was completed.

When deleting a request, a warning box appears. If you click OK for a request that is pending, it will be deleted. If the request has already been handled, it will not be deleted.

Statistics

This window displays the request statuses, which are listed below:

- Failed Request - A request that could not be completed
- Completed Request - A request that is finished
- In-Progress Request - A request being processed
- Pending Request - A request waiting for an available server or for other reasons
- Total Requests - All failed, in-progress, pending, and completed requests

The number of requests by status are displayed by a color-coded bar line. You can determine the status options that are displayed and their designated colors from the **Requests Filter** dialog. You can return to the default settings by clicking **Default**.

Applications

You can open a window displaying the available applications by clicking **Available Applications**.

Window Displays

You can determine the windows that are displayed by selecting the window options from the **View** menu.

The Monitor utility screen can appear as horizontal tiles or as cascading windows by selecting **Tile Horizontally** or **Cascade** from the **Windows** menu. Click **Default Layout** to return to the original screen display.

Changes to the screen can be displayed immediately by clicking **Refresh (F5)**. You can also set how often the Monitor utility refreshes the screen by clicking **Refresh Settings** in the **Options** menu. Enter the refresh time in seconds or the number of requests before eDeveloper refreshes the Monitor utility values.

Monitoring Servers

You can start monitoring by clicking **Start Monitoring** from the **Monitor** menu. The Monitor utility prompts you to select a request broker. The local broker is the default.

You can add or remove a request broker by clicking **Broker List** from the **Options** menu. The Broker List dialog appears.

Click **Stop Broker** to stop monitoring the status of your requests.

The Documentation Template Facility

The Documentation Template facility is used to create hard copies of the eDeveloper elements - such as tables, repositories, dialog boxes, and end-user forms - that are associated with the various eDeveloper structures for a particular application. These hard copies can serve as developer documentation.

Producing Template Documentation

To produce template-specified documentation, create a documentation template file as explained in the "Syntax" section. The file must be in text format, and its width cannot exceed 256 characters. Note, however, that the width of the documentation output is 132 columns.

You can use the default documentation template file, DOC_STD.ENG, or DOC_EXT.ENG as a guideline. These files, which will create documentation for all data items, are included as one of the configuration files in the eDeveloper package.

After you have prepared your documentation template file, you can run the export utility in Document mode, using the following procedure:

1. After opening your application, choose *Settings/Environment* and select the External files tab.
2. Use *Edit/Table Locate* or page down to get to the Documentation Template File property. Enter the name of your documentation template file.
3. Select *File/Export-Import* SHIFT+F10 to access the *Export/Import* dialog.
4. Select Export Document from the Operation combo box.

For the Type combo box, select the eDeveloper component you want to have documented (except for Application components). Your documentation template file must include a section for the component you select.

If your documentation template file has sections for several eDeveloper components, repeat steps 5 to 9, selecting a different eDeveloper component for the Type combo box each time.

1. If there are several occurrences of the type you have just selected, you can specify a subset of these items by zooming in the Range From and To properties to indicate the range of occurrences you want to document.
2. Type an output file name in the file name field. The documentation generator will choose the first letter of the output file name based on the component you have selected, placing the file in the same disk directory in which the application resides, according to the application prefix specified in the Application repository. You can override the generated first letter in the filename.
3. Press OK to confirm the export properties.
If the documentation template has a syntax error in the section you have specified, you will receive an error message at this point, and the process will terminate.
4. If you want to create documentation for other components, return to

the Type combo box and select the next type.

5. Leave eDeveloper by selecting *File/Exit System* (or choose *File/Shell to OS* to go out temporarily) and examine the documentation report(s) created.

You can interrupt the documentation generator at any point by pressing **ESC**.

The documentation generator uses the first entry in the Printer repository to obtain information such as the number of lines per page and the printer-control character set. You can access the Printer repository by selecting *Settings/Printers*.

Output can be directed to a printer directly by specifying the printer name as it appears in the Printer repository.

The application provided with eDeveloper is simple to use, and contains on-screen helps to assist in its usage.

Syntax - Documentation Template File

Each row of the documentation template file can contain both keywords and regular text that is printed as is. There are no restrictions as to where on the input line the keywords can appear. Similarly, both keywords and text can contain an unrestricted mixture of upper-case and lower-case type.

There are three types of command lines in the Documentation Template file: control lines, output lines, and comments. As noted below, each line begins with a special character that serves as a command-type indicator.

Control Lines

Control lines can be report section delimiters or condition indicators. They must begin with the # character.

Report Section Delimiters

The section delimiters signal either the beginning or the end of a report section for a given eDeveloper structure. All command lines for a particular section,

therefore, must appear within that section's delimiters. Similarly, the sub-section delimiters set off an eDeveloper sub-structure.

The section-start and sub-section-start delimiters take the form:

```
#Section Section_Name_Keyword, Parm1, Parm2, . . . , Parmn
```

where Parm indicates additional parameters. These parameters are tasks and must be separated by commas. Some are unique to a particular template section.

Section_Name_Keyword typically is derived from the eDeveloper data structure you want documented. For example, the beginning of the Type Table report section is indicated by:

```
#Section Type_Table,Top
```

Similarly, the section-end delimiter takes the form:

```
#End Section_Name_Keyword.
```

Sub-section command lines are nested inside section command lines. For example, if you want template documentation for the Task Properties and Task Control sub-sections of a Program Structure, the template input section will appear as:

#Section Program

Program Structure command lines -----

#Section Task_Properties

Task Properties command lines -----

#End Task_Properties

#Section Tasks_Control

Task Control command lines -----

#End Tasks_Control

Additional Program Structure command lines

#End Program

Condition Indicators

The condition indicators `If,' `Else,' and `Endif' set off text blocks that the documentation generator prints depending on the evaluations to true or false of the various condition indicators:

#If Condition

Block_of_Text_1

#Else

Block_of_Text_2

#Endif

Condition indicates a Boolean (logical) operator that compares a keyword's value with a constant or another keyword:

Symbol	Means
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Only one comparison can appear in any given line. However, since condition nesting is allowed, you can include compound conditions. For example, by placing each comparison on a separate line, you can tell the documentation generator to output text only when a series of conditions is met:

Output lines comprise the actual text blocks that will be printed by the documentation generator in each report section. There are three types of output lines: headers, data lines, and footers.

Headers

Headers correspond to report heading lines you want to appear in the documentation generator output. They can contain both keywords that are specific to particular sections as well as global keywords, which may be used anywhere in the documentation template file.

Each header must begin with the character 'H.'

Footers

The footer lines contain the section footer that you want to appear in the documentation report.

Footer lines, which can contain global keywords only, begin with the character 'F'.

Data Lines

All data lines must begin with the character 'D'. Data lines implies a loop to the documentation generator on all the elements of the data structure being output.

Data lines correspond to the actual data of the eDeveloper structure that you want to document. The information required depends on the report-section type, as noted below in "Documentation Template Sections."

All three output line types can contain printer control characters. You can use these, for example, to print wide reports on standard-size paper by specifying the character sequence that initiates compressed print.

Comments

Comment lines are internal notes that do not appear in the output report. Put any explanatory information that you do want the documentation generator to print in either the header or footer lines.

Comments must begin with the characters ';' or `**'.

Documentation Report Sections

There are three types of report sections:

- full screen - for entry screens, such as the Task Properties Screen. To receive an accurate copy of the screen from the documentation report generator, enter the relevant keywords across several data lines, as they appear in the actual screen.
- table - used to obtain a copy of a multi-line table, such as the Expressions repository. You supply a sample data line, including all pertinent keywords. The documentation report generator will output all of the table lines, using the sample provided as its guide.
- forms - used for text blocks, such as help screens or input forms. This section type generally does not have any special keywords other than screen-location variables. The documentation generator therefore will produce this type of report without requiring a sample output line or

screen from the user. As noted in the section on Special Parameters, the display block can accept the parameters: Frame (frame type) and Offset, which are unique to this sub-section.

Report Section Hierarchy

Shown below is a list of output report sections, and their hierarchy, that can be obtained from the documentation generator.

The command lines for lower-level reports (sub-sections) are nested within the higher level sections to which they belong.

Any section or sub-section can appear more than once in the input file. This feature should prove useful whenever you want two documentation copies of a particular eDeveloper structure, one showing more detail than the other.

Application

Model
repository

Table
repository

Index repository

Index Segments
repository

Foreign Key
repository

Help Screens
repository

Help Display Block

Menus
repository

Context Menu
Definition
repository

Menu Line
Description

Pulldown Menu
Definition
repository

Menu Line
Description

Application
Data

Application Events

Rights
repository

Program
repository

Task Properties
SQL Commands
SQL Where
Task Control
Local Variable
repository
DB Table
repository
I/O File repository
Sort repository

Events
repository

Events Parameters
repository

Expression
Rules
repository
Form
repository

Form Display Block
Field Location
Table (Displayed
Item Dialog)

Level Definition
repository

Operation
repository

Operations
Description

Keywords

Keywords are distinguished from other text by the % character. Each set consisting of a keyword plus parameter list is enclosed by the % character. That is, the % character will appear as the first character and as the last character of such sets:

%Keyword, Parm1, Parm2, . . . , Parmn%

Optional Parameters

There are three Optional parameters:

- *nn* - a number that defines the width of the field in which data represented by the keyword is to appear in the output report.
- *Wrap* - indicates that the data represented by the keyword output is to be continued in its field on the following line if the output does not fit in the field width that has been specified. If *Wrap* is not specified, the data represented by the keyword output will be truncated if it exceeds the field width specified.

Example: if the data represented by the keyword output is 23 letters long, and contains the line 'This is a sample output'

```
%Keyword,  
10,Wrap%
```

will produce

```
This is a  
sample  
out  
put
```

Specifying an input line of

```
%Keyword,  
10%
```

will produce

This is
a

The text “sample output” is truncated.

- Blank - tells the documentation generator to print blanks when the given numeric data represented by the keyword has the value zero.

Special Parameters

Some keywords can accept special parameters that modify their output values:

- Name - causes a field's description to be displayed instead of its code. Many file parameters in the various eDeveloper dialogs and repositories contain codes of indexes of elements in other eDeveloper repositories. The Name parameter causes the description of the code to be output instead of the code or index. For example, one of the keywords in the File Table Fields sub-section is Fld_Hlp, which outputs the number of the field's Help Screen. When the Name parameter is specified along with this keyword, the title of the Help Screen from the Help repository appears in the output report.
- Short / Long - appears after keywords used to print an expression's number. The Short and Long parameters tell the documentation generator to print the expression itself in short or long (expanded) form. An exception, however, can occur in the case of a keyword that represents a field whose value is `Yes,' `No,' or an expression number. When the field's value is either `Yes' or `No,' the actual field value will be displayed, even if you have instructed the documentation generator to print the expression itself.
- File - relevant only in the Field Location sub-section of Form repository. The File Parameter indicates the name of the file to which a particular field belongs.

Keyword Types

There are two types of keywords, global and local.

Global keywords can be used anywhere in the template file, e.g., Date and Time (the date and time, respectively when the report is printed).

Local keywords are relevant only in specific sections.

A section or sub-section that is nested within a higher-level section (such as a dialog invoked by a repository field - see the Report Section Hierarchy chart) can use the keywords associated with the higher-level section.

Section Delimiters and Keywords

The following describe the section delimiters and keywords.

Model Repository

Section delimiter: Model_Table

Keyword	Task & Special Parameters	Produces
EXD_MODEL_TBL_IDX		Model entry index number
EXD_MODEL_TBL_NAME		Model name
EXD_MODEL_TBL_FOLDER		Model folder
EXD_MODEL_TBL_CLASS		Model class
EXD_MODEL_TBL_ATTR		Model attribute

Table Repository

Section delimiter: File_Table

Keyword	Produces
Idx	Table entry index
File_Name	Table (File) description
File_Folder	Table folder
Fld_Cnt	Column (Field) counter

Keyword	Produces
Key_Cnt	Index (Key) counter
Size	Table (File) size
File_Path	Optional table name (filename)
File_DB	Physical database used
File_Acc_Key	Data file access index (key)
Encr	Table (File) encryption flag
File_DB_Info	Database information
Resident	Resident Memory repository (Table)
SQL_Hint	SQL Hint string
SQL_Cursor	SQL cursor
SQL_Check_Exist	SQL check
SQL_File_Type	SQL file type
SQL_Array_Size	SQL array size
SQL_Owner	SQL owner
SQL_Position	SQL position
SQL_Position_Key	SQL position key
SQL_File	SQL file
Cache_Strg	Cache strategy

There are no Optional parameters for this section's keywords.

Column Repository

Sub-section delimiter: Fields_Table

Keyword	Optional & Special Parameters	Produces
Fld_Idx		Column (Fields) repository Index
Fld_Name		Column (Field) Name
Type	Name	Column (Field) Type Number/Name
Attr		Column (Field) Attribute
Picture		Column (Field) Picture
Range		Column (Field) Range
Fld_Hlp	Name	Help Screen Number/ Name
Fld_Prompt	Name	Prompt Help No./ Name
Fld_Sel_Prg_Nr	Name	Select Program No./ Name
Fld_Sel_Prg_CI		Column (Field) Prop's Sel Pgm Call
Fld_Modif		Column (Field) Prop's - Modifiable
Fld_Trans		Column (Field) Prop's - Translate
Fld_Stor_As		Column (Field) Prop's - Stored as
Fld_Size		Column (Field) Prop's - Size

Keyword	Optional & Special Parameters	Produces
Fld_Type	File_Name_Size	Database type
Fld_User_Type	File_Name_Size	Database user type
Fld_DB_Def		Column (Field) Prop's - Column's definition
Fld_DB_Name		Column (Field) Prop's - Database Name
Fld_DB_Info		Column (Field) Prop's - Database Information.
Null-Allow		Column (Field) Prop's - Allow Nulls
Null_Val		Column (Field) Prop's - Calc Value
Null_Dsp		Column (Field) Prop's - Displayed string
Null_Default		Use Null as a default (Yes, No)
Default		Null default value
DB_Default	File_Name_Size	Database default value
CTRL		Number = type of control
Ret_Act		Return action
Rb_Cols		No.of radio button columns
Sld_Step		Slider step
VISUAL_LineS		No. of lines

Keyword	Optional & Special Parameters	Produces
VISUAL_STYLE		3D or 2D
VISUAL_BORDER		Thick, Thin, None
VISUAL_MULTI_Line		Yes/No
VISUAL_COLOR		Color number
VISUAL_FONT		Font number

Index Repository

Sub-section delimiter: Keys_Table

Keyword	Produces
Key_Clustered	Clustered Index (Yes, No)
Key_Constraint	Index constraints
Key_Hint	Index Hint string
Key_Idx	Index (Keys) Table Index
Key_Name	Index (Key) Name
Key_Type	Index (Key) Type (Unique or Non-unique)
Key_Dir	Index (Key) Properties - Direction
Key_Rng_Mod	Index (Key) Properties - Range Mode
Key_DB_Name	Index (Field) Properties - Data Base Name
Key_DB_Info	Index Properties - Database Information
Key_Vir_Real	Index Properties - Real variables
Seg_Cnt	Index (Key) Segments Counter

Index Segments Repository

Sub-section delimiter: Keys_Seg_Table

Keyword	Optional & Special Parameters	Produces
Seg_Idx		Index (Key) Segments Table Index
Seg_Fld	Name	Index (Key) Segments Field Index/Name
Seg_Size		Index (Key) Segment Size
Seg_Dir		Index (Key) Direction

Foreign Key Repository

Section delimiter: FKeys_Table

Keyword	Optional & Special Parameters	Produces
FKey_Idx		Foreign Key Table Index
FKey_Name		Foreign Key Name
R_Tab_Name		Referenced Table Name
R_Tab_Idx		Referenced Table Index
PKey_Name		Public Key Name

Keyword	Optional & Special Parameters	Produces
PKey_idx		Public Key Index

Foreign Key Segment Repository

Section delimiter: FKeys_Seg_Table

Keyword	Optional & Special Parameters	Produces
FKey_Seg_Idx		Foreign Key Segment Table Index
CTab_Col_Idx		Current Table Column Index
CTab_Col_Name		Current Table Column Name
RTab_Col_Idx		Referenced Table Column Index
RTab_Col_Name		Referenced Table Column Name

Component Repository

Keyword	Optional & Special Parameters	Produces
Comp_Idx		Component Table Index
Comp_Name		Component Name
Comp_Folder		Component Folder

Keyword	Optional & Special Parameters	Produces
Comp_Desc		Component Description

Component Models

Section delimiter: Comp_Models

Keyword	Optional & Special Parameters	Produces
Model_Idx		Model Index
Model_Name		Model Name
Model_Remark		Model Remark

Component Tables

Section delimiter: Comp_Tables

Keyword	Optional & Special Parameters	Produces
Table_Idx		Table Index
Table_Name		Table Name
Table_Remark		Table Remark

Component Programs

Section delimiter: Comp_Programs

Keyword	Optional & Special Parameters	Produces
Program_Idx		Program Index
Program_Name		Program Name

Keyword	Optional & Special Parameters	Produces
Program_Remark		Program Remark

Component Helps

Section delimiter: Compo_Helps

Keyword	Optional & Special Parameters	Produces
Help_Idx		Help Index
Help_Name		Help Name
Help_Remark		Help Remark

Component Rights

Section delimiter: Comp_Rights

Keyword	Optional & Special Parameters	Produces
Rights_Idx		Rights Index
Rights_Name		Rights Name
Rights_Remark		Rights Remark

Component Events

Section delimiter: Comp_Events

Keyword	Optional & Special Parameters	Produces
Events_Idx		Events Index
Events_Name		Events Name

Keyword	Optional & Special Parameters	Produces
Events_Remark		Events Remark

Component Properties

Keyword	Optional & Special Parameters	Produces
Comp_ID		Component Identification
Appl_File		Application File
Flat_MCF		Flat MCF Deployment
Revision		Revision
Load_Imm		Load Immediately
Help_File		Help File
Help_Key		Help Key
Mci_File		MCI File
Appl_Prefix		Application Prefix
Color_File		Color File
Color_YN		Use Color File
Font_File		Font File
Font_YN		Use Font File

Help Screen Repository

Section delimiter: Help_Screens

Keyword	Produces
Hlp_Idx	Repository (Table) Index

Keyword	Produces
Hlp_Name	Help Screen Name
Hlp_Folder	Help folder
Hlp_Row	Help Screen row location on screen
Hlp_Left	Help Screen column location on screen
Hlp_Height	Help block height (in UOM)
Hlp_Width	Help block width (in UOM)
Hlp_Clr	Help Screen color number
Hlp_Top	Help block top position (in UOM)
Hlp_Win_Type	Help type: Internal/ Prompt/ Windows
Hlp_Win_File	Windows Help file name
Hlp_Win_Command	Windows Help command
Hlp_Win_Key	Windows Help key

Keyword	Produces
Hlp_Title_Bar	Help has Title Bar? Yes/No
Hlp_Sys_Menu	Help has Sys Menu? Yes/No
Hlp_Font	Font number
Hlp_Win_Prompt	Prompt line
Hlp_Frame_Type	Frame type: Thin/Thick/ None

Help Display Block

Sub-section delimiter: Help_Block

This sub-section reports the contents of the Help Screen.

Rights

Section delimiter: Rights

Keyword	Produces
Idx	Repository (Table) Index
Name	Right's name
Folder	Right's folder
Key	The Right's Index (Key)
Public	The Right's Public Flag

Context Menus

Section delimiter: Context_Menus

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Menu_Type		Menu Type
Menu_Name		Name in Menu
Menu_Sh_Key		Menu Short Index (Key)
Menu_Hlp	Name	Menu Help Screen
Menu_Acc_Right		Menu Access Right
Menu_Level	Name	Menu Level and Name
Menu_Cnt		Menu entry counter

Context Menu is a table-type section, refer to Documentation Report Sections above. The way in which each line in the Context Menu repository is processed depends on its sub-section type. These types are unique to the Menu structure.

Entry Type 1: Program

Sub-section delimiter: POP_Prog

Keyword: Mnu_Prg_Nr

Optional parameter: Name

Produces the invoked program's number or name.

Entry Type 2: Exit

Sub-section delimiter: POP_Exit

Keyword	Specifies
Mnu_OS_Cmd	OS command to be executed
Mnu_Wait	Wait for the started task? Yes/No
Mnu_Show	Show the started task? Yes/ No

Entry Type 3: System

Sub-section delimiter: POP_System

Keyword: Mnu_Act

Produces the action's description.

Entry Type 4: Line

Sub-section delimiter: POP_Line

No special keywords.

Entry Type 5: Menu

Sub-section delimiter: POP_Menu

Keyword: Sub_Lines

Produces: number of sub-lines

Pulldown Menus

Section delimiter: Pulldown_Menus

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Menu_Type		Menu Type

Keyword	Optional & Special Parameters	Produces
Menu_Name		Name in Menu
Menu_Sh_Key		Menu Short Index (Key)
Menu_Hlp	Name	Menu Help Screen
Menu_Acc_Right		Menu Access Right
Menu_Pos_If_Lst		Menu position if last entry
Menu_level	Name	Menu Level and Name
Menu_Cnt		Menu entry counter
Menu_Prompt_Help		Menu Prompt Help
Menu_Is_Pulldown		Pulldown Menu
Menu_Tool_Image		Menu Tool Image
Menu_Tool_Tip		Menu Tool Tip
Menu_Tool_Group		Menu Tool Group
Menu_Tool_Number		Menu Tool Number
Menu_Checked		Menu Checked
Menu_Visible		Visible Menu
Menu_Enabled		Enabled Menu

Pulldown Menu is a table-type section, refer to Documentation Report Sections on page page 1309. The way in which each line in the Pulldown Menu repository is processed depends on its sub-section type. These types are unique to the Menu structure.

The sub-section definitions for Pulldown Menus can be found above in the "Context Menus" section, with the "POP_" prefix replaced with "PDM_," e.g., PDM_Prog instead of POP_Prog.

Application Properties

Section delimiter: CTL_Data

Keyword	Optional & Special Parameters	Produces
Alpha_Nul_Def		Alpha Null value as a default (Yes, No)
Alpha_Def_Val	Name_Size	Alpha default value
Num_Nul_Def		Numeric Null value as a default (Yes, No)
Num_Def_Val		Numeric default value
Bool_Nul_Def		Logical Null value as a default (Yes, No)
Bool_Def_Val		Logical default value
Date_Nul_Def		Date Null value as a default (Yes, No)
Date_Def_Val		Date default value
Time_Nul_Def		Time Null value as a default (Yes, No)
Time_Def_Val		Time default value
Memo_Nul_Def		Memo Null value as a default (Yes, No)
Memo_Def_Val	Name_Size	Memo default value
MVCS_Right_Key	Name	Force MVCS security key
HTML_Style_File	File_Name_Size	HTML Style file
Internet_Root	File_Name_Size	Internet Development file root

Keyword	Optional & Special Parameters	Produces
Logo_Scr	Name	Help Screen used as Logo Screen
Startup		Application startup mode (Toolkit, Runtime)
Pr_Attr_File		Application Print Attributes repository (file) names
Clr_Def_File		Application Color definition repository (file)
Nul_Clc		Application Null Arithmetic
Pub_Right_Key		Public Rights Access Index (Key)
Super_Key		Super Right Index (Key)
Appl_Key		Application Access Index (Key)
Fnt_Def_File		Font Definition repository (file)
Fore_St_End		Yes/No Force Start/End prog
Kbd_Map_File		Keyboard mapping repository (file)

The Application Data section has a sub-section for NULL values defaults.

NULL Value Defaults

Sub-section delimiter: Null_Data

Keyword	Produces
Nul_Alpha_Val	Null default value for Alpha data type
Nul_Num_Val	Null default value for Numeric data type
Nul_Bool_Val	Null default value for Boolean data type
Nul_Date_Val	Null default value for Date data type
Nul_Time_Val	Null default value for Time data type
Nul_Memo_Val	Null default value for Memo data type
Nul_Alpha_Dsp	Null display string for Alpha data type
Nul _Num_Dsp	Null display string for Numeric data type
Nul_Bool_Dsp	Null display string for Boolean data type
Nul_Date_Dsp	Null display string for Date data type
Nul_Time_Dsp	Null display string for Time data type
Nul_Memo_Dsp	Null display string for Memo data type
Nul_BLOB_Dsp	Null display string for BLOB data type
Alpha_Def_Value	Alpha Default Value
Num_Def_Value	Numeric Default Value

Keyword	Produces
Bool_Def_Value	Boolean Default Value
Date_Def_Value	Date Default Value
Time_Def_Value	Time Default Value
Memo_Def_Value	Memo Default Value

Application Events

Sub-section delimiter: CTL_Events

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Hot_Key		Hot Key
Elaps_Time		Elapsed Time
Nam_Exp	Short, Long	Expression number returning the Elapsed Time
Call		Call type
Prog_Task	Name	Number/Name of the called program
Parm_Cnt		Parameter counter
Event_Cnt		Events repository (Table) entry counter
Action		Action that triggers event

Program Repository

Section delimiter: Program

Keyword	Optional & Special Parameters	Produces
Prg	Name	Number/Name of the current report Program
Prg_Folder		Program Folder
Public	Public_Name	Public Name of the Program
Task	Name	Level or Name of the currently reported Task
Task_Id	Name	Name of the currently reported Task (or of the Task and its parent tasks as specified in the form Program.Task_Parent1. (...) Task_ParentN.Task_Name
Upd_Date		Date of the Last Update
Upd_Time		Time of the Last Update
First_Prg		Flag that indicates if the currently reported Program is the first (1 = yes; 0 = no)
Last_Prg		Flag that indicates if the currently reported Program is the last (1 = yes; 0 = no)

Keyword	Optional & Special Parameters	Produces
Batch_Tsk		Returns 1 if task is batch or 0 if task is online

Task Properties

Sub-section delimiter: Task_Properties

Keyword	Optional & Special Parameters	Produces
Alo_Abort		Allow Abort
Main_Display	Short, Long	
Task_Name		Task Name
Main_File	Name	Number/Name of the Main Table (File)
Key	Name	Number/Name of the Sort Index (Key)
Key_Exp	Short, Long	Number of the Expression that returns the Index (Key) Number
Tsk_Typ		Task type (online, batch)
Init_Mod		Initial task mode
Ini_Mod_Exp	Short, Long	Number of the Expression that returns the initial mode
Sel_Tab	Short, Long	Selection repository (Table)

Keyword	Optional & Special Parameters	Produces
Res_Tsk		Flag indicating whether the Task is resident (1=yes; 0 = no)
Icon_Name		Icon File Name
Task_Size		Returns the size of the task
Icon_Name		Returns the Icon file name
Popup_Menu		Attached Popup menu
Sql_Tsk		Yes/No: Main table (file) is SQL
Limit_To		Row Limit expression
Trans_Mode		Transaction Mode
Trans_Begin		Transaction Begin
Lock_Strg		Locking Strategy
Error_Strg		Error Behavior Strategy
Var		By Variable
Exd_Tsk_Prop_Ret_Val_Exp		Return value expression
Exd_Tsk_Prop_Chunk_Sz_Exp		Chunk size expression
Exd_Tsk_Prop_Exit_Url		Exit URL

SQL Command

Sub-section delimiter: SQL_Cmd

Cmd	Optional & Special Parameters	Produces
Db_Name		Database where SQL command is executed
Res_Tab		Result repository (table) of the SQL Command
Cmd		SQL command text
Inp_Prms_Cnt		Input parameter count
Out_Prms_Cnt		Output parameter count
Ret_Code	Name	Return code variable identifier/name

SQL Command Input Parameter Table

Sub-section delimiter: SQL_Cmd_Inp_Prms

Keyword	Optional & Special Parameters	Produces
Parm_Idx		Parameter repository (table) index number
Parm_Var	Name, File	Variable number / name used as an input parameter / table (file) to which variable belongs

Keyword	Optional & Special Parameters	Produces
Parm_Exp	Short, Long	Expression number of text

SQL Command Output Parameter Table

Sub_section delimiter: SQL_Cmd_Out_Prms

Keyword	Optional & Special Parameters	Produces
Parm_Idx		Parameter repository (table) index number
Parm_Var	Name, File	Variable number / name used as an input parameter / table (file) to which variable belongs
Parm_Exp	Short, Long	Expression number of text

Range and Locate

Keyword	Optional & Special Parameters	Produces
Rng_Exp		Range Expression
Rng_Ord		Range Order
Loc_Exp		Locate Expression
Loc_Ord		Locate Order
Pos_Exp		Position
Usage		Usage

Keyword	Optional & Special Parameters	Produces
SQL_Exp		Magic SQL Expression
DB_SQL		DB SQL

SQL Where Clause

Sub-section delimiter: Where_Clause

Keyword	Optional & Special Parameters	Produces
Where_Clause		SQL Where Clause

Task Control

Sub-section delimiter: Task_Control

Keyword	Optional & Special Parameters	Produces
Opn_Tsk_Win	Short, Long	Open Task Window
Clos_Tsk_Win	Short, Long	Close Task Window
Fg_Win	Short, Long	Foreground Window
Ref_Win	Short, Long	Refresh Task Window
Flip_Fld	Short, Long	Flip Record (Field) on Input
Cyl_Rec	Short, Long	Cycle Record Main
Conf_Upd	Short, Long	Confirm Update
Loc_Exp	Short, Long	Locate Expression Number
Loc_Ord		Locate Order

Keyword	Optional & Special Parameters	Produces
Rng_Exp	Short, Long	Range Expression
Rng_Ord		Range Order
End_Tsk	Short, Long	End Task
End_Chk		End Check
For_Rec_Suf	Short, Long	Force Row (Record) Suffix
For_Rec_Del	Short, Long	Force Row (Record) Delete
Form_Recs		Form Row (Records)
Lock_Strg		Locking Strategy
Skip_Lock		On Locked Row (Record)
Skip_Fail		On Failed Row (Record) Access
Alo_Opt	3, Short, Long	Allow Tasks
Alo_Mod	3, Short, Long	Allow modify
Alo_Cre	3, Short, Long	Allow create
Alo_Del	3, Short, Long	Allow delete
Alo_Qur	3, Short, Long	Allow query
Alo_Loc	3, Short, Long	Allow locate
Alo_Rng	3, Short, Long	Allow range
Alo_Key_Ch	3, Short, Long	Allow key change
Alo_Sort	3, Short, Long	Allow sorting
Alo_IO	3, Short, Long	Allow Input/Output files
Alo_Rpr	3, Short, Long	Allow reports

Keyword	Optional & Special Parameters	Produces
Alo_Kopt	3, Short, Long	Allow key optimization (Yes, No)
Alo_Qlct	3, Short, Long	Allow locate query (Yes, No)
Cache_Strg		Cache Strategy

Local Variable Repository

Sub-section delimiter: Virtual_Fields

Keyword	Optional & Special Parameters	Produces
Fld_Idx		Repository (Table) Index
Fld_Name		Column (Field) Name
Line		Line
Prg		Program
Task		Task
Type		Column (Field) Type
Attr		Column (Field) Attribute
Picture		Column (Field) Picture
Range		Column (Field) Range

Keyword	Optional & Special Parameters	Produces
V_Fld_Hlp	Name	Column (Field) Properties - Help Screen Number
V_Fld_Prmt	Name	Prompt Help Number/Name
V_Fld_Sel_Prg_Nr	Name	Select Program Number/Name
V_Fld_Sel_Prg_CI		Select Program Call
Null_Allow		Value of the Allow Nulls Parameter
Null_Val		Null Value for Calculation
Null_Dsp		Null Display String
Null_Allow		Allow Null values
Null_Val		Calculated Null value
Null_Dsp		Displayed Null value
Null_Default		Null as a default value (Yes, No)
Default		Null default value
Ret_Act		Return Action
SLD_Step		
Translate		Data native translation
Visual_Line		No. lines
Visual_Cols		No. columns

Keyword	Optional & Special Parameters	Produces
Visual_Style		3D or 2D
Visual_Border		Thick, Thin, None
Visual_Goin		Yes/No
Visual_Multi_Line		Yes/No
Visual_Color		Color no.
Visual_Font		Font no.
CTRL		No. representing control

DB Tables Repository

Sub-section delimiter: DB_Files

Keyword	Optional & Special Parameters	Produces
Idx		Table Index
File	Name	File Number/Name
Acss		Access Method
Share		Share Task
Open		Open Task
Nam_exp	Short, Long	Number of the Expression that returns the filename
Cache_Exp		DB file cache size expression

I/O File Repository

Sub-section delimiter: IO_Files

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Name		Table (File) Name
Media		type of Media
Page		type of Paper
Header		Header
Footer		Footer
Copies		Number of Copies
Expcopies		
Orient		Page Orientation
Prn		Printer Name
Acss		Access Method
Frmt		I/O Format
Nam_Exp	Short, Long	Number of the Expression that returns the tablename
Rows		Number of rows
Pdlg		Open a printer dialog when printing with a graphic printer? Yes/No
Tmpl_Nam		Template name
Tmpl_Nam_Exp	Shrt, Long	Template Name expressions

Keyword	Optional & Special Parameters	Produces
Prefix		Token prefix
Suffix		Token suffix

Sort Repository

Sub-section delimiter: Sort_Table

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Var	Name	Variable Number/ Name
Size		Variable size
Dir		Sort Direction column

Task Event Repository

Sub-section delimiter: Events_Table

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Hot_Key		Hot Key
Elaps_Time		Elapsed Time
Nam_exp	Short, Long	Expression number returning the Elapsed Time

Keyword	Optional & Special Parameters	Produces
Scope		Event Scope (Task or Structure)
Call		Call type
Prog_Task	Name	Number/Name of the called program
Lock		Event Lock parameter
Parm_Cnt		Parameter counter
Event_Cnt		Events repository (Table) entry counter
Action		Action that triggers event
Exd_Evnt_Desc		Event description
Exd_Evnt_Type		Event type
Exd_Evnt_Trigger		Event trigger
Exd_Evnt_Force_Exit		Event force exit

Task Event Parameter Repository

Sub-section delimiter: Event_Parms

This repository contains the parameters passed to the called Program / Task.

Keyword	Optional & Special Parameters	Produces
Parm_Idx		Parameter repository (Table) Index Number

Keyword	Optional & Special Parameters	Produces
Parm_Var	Name, File	Variable Number/ Name used as a parameter/table to which Variable belongs
Parm_Exp	Short, Long	Expression Number

Expression Rules Repository

Sub-section delimiter: Exp_Table

The Expressions Table combines the Program Variable (fields) Table and the Program Expressions Table.

Keyword	Optional & Special Parameters	Produces
Idx		Repository (Table) Index
Fld	Name, File	Column (Field) Number/Name/File
Exp	Short, Long	Expression number / Short or Long description
Exp_Cnt		Number of Expressions in the repository
Fld_Cnt		Number of Columns in the Repository (Table)

Form Repository

Sub-section delimiter: Forms_Table

This section reports the following Form repository information:

- Display Block information
- The location of columns in each form in a Form repository sub-section
- The layout of forms belonging to a particular class

The Form repository documentation display depends on whether the program I/O is routed only to the interactive window or to other output devices as well. eDeveloper reserves Class 0 for the interactive window. A program can also have forms that are used for output reports or files; each corresponding output device is allocated a unique non-zero class number. For each particular report on a given output device, various report sections can be processed by different tasks. The documentation generator will print the report's entire Display Block contents at the lowest task level that outputs records to a non-zero-class device. Display Block parameters, however, are displayed at each task level that outputs records for a particular report:

- When the Display Block class is 0, corresponding to the interactive window, the Display Block parameters are reported in a one-entry table, and the Display Block contents are reported immediately afterwards. The value returned by the Disp_Area keyword is 1.
- When the Display Block class is non-zero, and a lower level task also has a Display Block of the same class, then the Display Block parameters are printed in a one-entry table, but the Display Block contents are not shown at the present task level. The value returned by the Disp_Area keyword is 0.
- When there is more than one Display Block whose class is non-zero, and no lower level task has a Display Block of the same class, all the Display Blocks of the same class are reported in one Form repository. Their

combined Display Block contents appear immediately after the table. The value returned by the Disp_Area keyword is 1.

Keyword	Optional & Special Parameters	Produces
Idx		Form repository (Table) Index
Dsp_Blkc		Display Block Title
Class		Block class
Area		Block area type
Rows		Block height
Cols		Block width
Color		Block color
Hlp	Name	Assigned Help Screen Number/Name
Disp_Area		Flag indicating whether the contents of the Display Block are reported right after the current Display Block record or in a lower level task
First_Dsp_Blkc		Whether the record of the currently reported Form is the first
Last_Dsp_Blkc		Whether the record of the currently reported Form is the last
Uom_Type		Unit of measurement
Vert_Factor		Vertical factor
Hor_Factor		Horizontal factor
Show_Grid		Show form's grid? Yes/No

Keyword	Optional & Special Parameters	Produces
Grid_X		Size of grid X
Grid_Y		Size of grid Y
Interface_Type		GUI/Text based/HTML Forms and Documents, and Java forms
Att_Txt_Form	Name	Index/name of attached text form
Is_Child		Form is child? Yes/No
Title_Bar		Form has Title bar? Yes/No
Title_Bar_Exp	Short, Long	Form has Title bar - expression
Sys_Menu		Form has Sys menu? Yes/No
Min_Button		Form has Min button? Yes/No
Max_Button		Form has Max button? Yes/No
Use_Best_Plt		Form uses Best palette
Font		Font number
Frame_Type		Frame type: Thick/Thin/None
Frame_Left		Frame left position (in UOM)
Frame_Top		Frame top position (in UOM)
Frame_Width		Frame width (in UOM)
Frame_Height		Frame height (in UOM)

Keyword	Optional & Special Parameters	Produces
Frame_Left_Exp	Short, Long	Frame left position - expression
Frame_Top_Exp	Short, Long	Frame top position - expression
Frame_Width_Exp	Short, Long	Frame width - expression
Frame_Height_Exp	Short, Long	Frame height - expression
Form_Placement_Left		0-100 percent
Form_Placement_Top		0-100 percent
Form_Placement_Width		0-100 percent
Form_Placement_Height		0-100 percent
Hyper_Lnk		Form hyperlink
Hyper_Lnk_Type		Form hyperlink type
Magic_App		eDeveloper Application form name
Public	Public_Name_Len	eDeveloper Hyperlink form public name
Param		eDeveloper Application form parameter names
URL		Hyperlink URL
URL_Exp	Shrt, Long	Hyperlink URL Form Expressions
DST_Frame	Name_Size	Hyperlink forms destination frame
Ctrl_Name	Name_Size	Hyperlink forms control name
Cntxt_Vars		Context variables

Keyword	Optional & Special Parameters	Produces
Use_Cookies		Use Cookies
Domain_Restr		Domain restriction
Expr_Date_Exp	Shrt, Long	Expiration date expression
Expr_Time_Exp	Shrt, Long	Expiration time expression
Secured		Secured Cookies
Wallpaper	File_Name_Size	Wallpaper file name
Wallpaper_Exp	Shrt, Long	Wallpaper file name expression
Submit_Form		Submit form (Yes, No)
Img_Border_Val		Defines a border value for an image control
Img_Border_Exp		Determines an expression for an image control border value.
Color_Exp		Color number expression
Ctrl_Hyper_Lnk		Controls hyperlink
Ctrl_Hyper_Lnk_Type		Controls hyperlink type
Ctrl_Magic_App		eDeveloper Application control name
Ctrl_Public	Public_Name_Len	Hyperlink eDeveloper public program
Ctrl_Parm		eDeveloper Hyperlink Control Parameters
Ctrl_URL		Hyperlink URL
Ctrl_URL_Exp	Shrt, Long	Hyperlink URL expression

Keyword	Optional & Special Parameters	Produces
Ctrl_DST_Frame	Name_Size	Hyperlink Destination frame
Ctrl_Ctrl_Name	Name_Size	Hyperlink control name
Stable_Rows		Static Table rows
Stable_Columns		Static Table Columns
Stable_Border		Static Table border width
Stable_Spacing		Static Table cell spacing
Stable_Padding		Static Table cell padding
Follow_Text		Follow text formatting
Hspace		Horizontal spacing
Vspace		Vertical spacing
Text_Type		Text type
Indent		Indent level
Int_Attr		Internal HTML Attribute
Int_Attr_Exp		Internal HTML expression
Is_Int_Attr_Exp		Is Internal HTML Attribute expression
Ext_Attr		External HTML attribute
Ext_Attr_Exp		External HTML Attribute expression
Is_Ext_Attr_Exp		Is External HTML Attribute expression
Inc_HTML_File		Include an external Head file
Inc_HTML_Exp		Include a Head file expression

Keyword	Optional & Special Parameters	Produces
Hidden_Var		Hidden variable
Para_Align		Paragraph alignment
Fixed_Width		Width property
Alt_Text		Alternate text property
DSP_Type		Display type

Form Display Block

Sub-section delimiter: Form_Block

This sub-section reports the contents of the Form Display Block.

Field Location Repository

Sub-section delimiter: Flds_Loc_Table

For each Display Block in the Form repository, this sub-section creates a repository of field locations.

Keyword	Optional & Special Parameters	Produces
Model		Model settings for field locations
Idx		Repository (Table) Index
Fld	Name, File*	Column (Field) Name
Exp	Short, Long	Number of the Expression whose result is displayed
Attr		Column (Field) Attribute

Keyword	Optional & Special Parameters	Produces
Picture		Column (Field) Picture
Pic_Exp	Short, Long	Number of the Expression that returns the Column (Field) Picture
Top		Column (Field) row location
Left		Variable (Field) column location
Height		Column (Field) width
Width		Column (Field) height
Allow_Inp	Short, Long	Input allowed
Must_Inp	Short, Long	Input required
Color		Column (Field) Color
Color_Exp	Short, Long	Number of the Expression that returns the Field Color
Exp_Win	Name	Number/Title of the Form for the expansion window
Hlp	Name	Number/Title of the assigned Help screen
Prompt	Name	Number/Title of the assigned Prompt Help screen

Keyword	Optional & Special Parameters	Produces
Sel_Prg_Nr	Name	Select Program Number/Name
Sel_Prg_Cl		Called Program method
Multi_Lin_Edt		Is Edit control a multi-line control? Yes/No
Visible		Control visible? Yes/No
Enable		Control enabled? Yes/No
Font		Font number
Ver_Align		Vertical alignment: Top/Center/Bottom
Hor_Align		Horizontal alignment: Left/Center/Right
Style		Control style: 2D/3D-raised/3D-sunken
Password		Is Edit control password protected? Yes/No
Img_Button		Is Push button an image? Yes/No
Border_Style		Border: Thin/Thick/None
Is_Hor_Sld		Is slider horizontal? Yes/No

Keyword	Optional & Special Parameters	Produces
Choice_Rng		No. of choices in Radio button
NW_Se_Line		Is Static control a line from NW to SE? Yes/No
Ne_Sw_Line		Is Static control a line from NE to SW? Yes/No
Hor_Line		Is Static control a horizontal line? Yes/No
Ver_Line		Is Static control a vertical line? Yes/No
Rect		Is Static control a rectangle? Yes/No
Round_Rect		Is Static control a round rectangle? Yes/No
Ellipse		Is Static control an ellipse? Yes/No
Def_Img_Name		Image file name (for Image control and Image button)
Img_Style		Image style: Tiled/Copied/Scaled...
Top_Exp		Control top position-expression
Left_Exp		Control left position-expression

Keyword	Optional & Special Parameters	Produces
Height_Exp		Control height - expression
Width_Exp		Control width - expression
Text		Text attached to the control
Combo_Lines		Number of lines in combo box
Fix_Size_Table		Set table size
Title_On_Every_Page		Places a title on every page
Font_Exp		Font expression
Range_Exp		Range expression for choice
Scroll_Bar		Table scroll bar
Line_Divider		Table - raw divider
Vert_Scroll		MLE vertical scroll
Horz_Scroll		MLE horizontal scroll
Allow_CR		MLE allow CR in data
Allow_MIQ		Allow modify in query
Placement_Left		0-100 percent
Placement_Top		0-100 percent
Placement_Width		0-100 percent
Placement_Height		0-100 percent
Img_Effect		Image effect
Tab_Side		Location of tabs

Keyword	Optional & Special Parameters	Produces
Choice_Cols		Radio button no. of cols.
OLE2_Class		OLE class
OLE2_Display_As		Display OLE as...
OLE2_Store_As		OLE Content...
OLE2_Auto_Link		Yes/No
OLE2_Frame_Type		Thick, Thin, None
Name		Field name
Ret_Act		Return action
Sld_Step		Slider step
Ctrl		Returns the name of the control. If in IF statement, returns the value: 10- Edit control 20- Push control 30- Choice control 31- Radio button control 32- Combo box control 33- Tab control 34- List control 40- Check control 50- Image control 60- Static control 70- Slider control 80- Table control 90- OLE control 100- Line control

* File is a special parameter indicating the file to which the Variable belongs.

Level Definition Repository

Sub-section delimiter: Level_Def_Table

This template section, which documents the Task Flow repositories, reports the Task's Break Level Definition repository and includes a sub-section for each level's Operations repository.

Keyword	Optional & Special Parameters	Produces
Idx		Level Definition repository Index
Var	Name	Change Level Variable Index*
Pref		Prefix Operation Count
Main		Main Operation Count
Suff		Suffix Operation Count
Trans		Transaction Value
Abort		Transaction Error Value
Level_Cnt		Number of entries in the Break Level Operations repository

* When the current Level is Record or Task, Var displays nothing.

Operation Repository

Sub-section delimiter: Oper_Table

Keyword	Optional & Special Parameters	Produces
Curr_Flow		Current flow - current Change Level (e.g. Record Main)
Idx		Operation repository Index
Oper_Cnt	Name	Operation Number/ Name
Flow_Dir		Flow direction
Flow_Mode		Flow mode
Cond_Exp	Short, Long	Operation Condition
Rec_Main		Flag indicating whether the current Task Flow is Record Main or not

The Operations repository sub-section is a repository-type section, see Documentation Report Sections on page page 1309 above. Each repository entry is handled as one of the sub-sections unique to the Operations repository:

Operation 0: Remark

Sub-section delimiter: OP_Remark

Keyword: Remark

Produces: remark contents

Operation 1: Select

Sub-section delimiter: OP_Select

Keyword	Optional & Special Parameters	Produces
Fld_Type		Column (Field) Type: Real/Virtual
Fld	Name	Table (File)
Ini_Exp	Short, Long	Initial Value Expression
Rng_Min_Exp	Short, Long	Range Min. Expression
Rng_Max_Exp	Short, Long	Range Min. Expression
Loc_Min_Exp	Short, Long	Locate Min. Expression
Loc_Max_Exp	Short, Long	Locate Min. Expression

Operation 2: Verify

Sub-section delimiter: OP_Verify

Keyword	Optional & Special Parameters	Produces
Verif_Exp	Short, Long	Expression to be verified
Verif_Mod		Verification mode

Operation 3: Link

Sub-section delimiter: OP_Link

Keyword	Optional & Special Parameters	Produces
Link_Type		Link Type
File	Name	Number/Name of the Linked Table (File)
Key	Name	Number/Name of the Linked Table Index (File Key)
File_Dir		Table (File) Access direction
Ret_Var		Variable receiving the return code

Operation 4: End Link

Sub-section delimiter: OP_End_Link

No special keywords.

Operation 5: Block

Sub-section delimiter: OP_Block

Keyword: Exd_Op_Block_Type (Block type)

Operation 6: End Block

Sub-section delimiter: OP_End_Block

No special keywords.

Operation 7: Call

Sub-section delimiter: OP_Call

Keyword	Optional & Special Parameters	Produces
Call_Type		Call type (Task or Program)
Prog_Task	Name	Number/Name of the called Program/Task (if the operation calls a Program or a Task)
Call_Exp	Short, Long	Expression Number/Name used when the operation calls by an Expression (for "Call By Expression" or "User Exit")
Parm		Number of Parameters passed
Form	Name	Form Number/Name
Ser_Priority_Lock		
Call_Lock		Lock Value for the Call Operation
Wait		Wait (Yes, No)
Service	Name_Size	Service name
Prog_Name		Program name
Res_File	File_Name_Size	Result file name
Ret_Code	Name	Return Code variable

Parameters List

Sub-section delimiter: Parm_List

Keyword	Optional & Special Parameters	Produces
Parm_Idx		Parameter repository (table) Index Number
Parm_Var	Name	Number/Name of Variable used as a Parameter/Row (File) to which Variable belongs
Parm_Exp	Short, Long	Expression Number
Parm_Nam		Parameter Name

Operation 8: Evaluate Exp

Sub-section delimiter: OP_Eval_Exp

Keyword	Optional & Special Parameters	Produces
Eval_Exp	Short, Long	Evaluated Expression
Ret_Var	Name, Table (File)	Number/Name/Table (File) of the return code.

Operation 9: Update Operation

Sub-section delimiter: OP_Update_Var

Keyword	Optional & Special Parameters	Produces
Fld	Name, Table (File)	Number/ Name/Table (File) of the Updated Column (Field)
Upd_Exp	Short, Long	Expression used for updating
Upd_How		Evaluation mode
Upd_Undo		Undo operation

Operation 10: Output Form, Output Merge

Sub-section delimiter: OP_Out_Form or OP_Out_Merge

Keyword	Optional & Special Parameters	Produces
Form	Name	Form Number/Name
IO_File	Name	I/O File Number/ Name
IO_Page_Form		Output page format
Param_Idx		Output Merge parameter name
Parm_Tag	Name, File	Output Merge parameter tag

Keyword	Optional & Special Parameters	Produces
Parm_Var	Name, File	Output Merge parameter variable
Parm_Exp	Short, Long	Output Merge parameter expression
Picture		Picture
Pic_Exp		Picture Expression

Operation 11: Java/Text Form

Sub-section delimiter: OP_In_Form

Keyword	Optional & Special Parameters	Produces
Form	Name	Form Number/ Name
IO_File	Name	I/O File Number/ Name
IO_DIm		Input File Delimiter type
IO_DIm_Ch		Delimiter character

Operation 12: Browse on Exp

Sub-section delimiter: OP_Browse

Keyword	Optional & Special Parameters	Produces
Brow_Exp	Short, Long	Tablename (Filename) Expression
Brow_Edt		Edit-scan mode
Form	Name	Form Number/Name

Operation 13: Exit on Exp

Sub-section delimiter: OP_Exit_on_Exp

Keyword	Optional & Special Parameters	Produces
Ex_Prog_exp	Short, Long	External program expression
Ex_Wait		Wait for the started task? Yes/No
Ex_Show		Show the started task? Yes/No
Ret_Var	Name, File	Number/Name/File of the return code

Operation 14: Raise Event

Sub-section delimiter: Op_RaiseEvent

Keyword	Optional & Special Parameters	Produces
Name		Name
Wait		Wait

Keyword	Optional & Special Parameters	Produces
Arg		Arguments
Cond_exp		Conditional Expression
Flow_Mode		Flow Mode
Flow_Dir		Flow Direction
Dest_CTX_Name		Destination context.

Section Parameters

Top - skip to a new page when starting the section and print the section header.

For sections that report the contents of Display Blocks:

Frame - the type of frame that surrounds the Display Block contents; the parameter syntax is:

Frame = *T*

where *T* is the frame type, chosen from the valid eDeveloper Display Block frame types. When a special character is used for the frame, the parameter syntax is:

Frame = Oc

where O is the letter, standing for "other," and c is the character to be used for the frame.

Offset - the offset in the output line containing the Display Block contents.

Example:

```
#Section Help_Block, Frame= T, Offset=3
```

Special Global Keywords

- Page - the current page number

- Line - the current line number
- Date - the current date
- Time - the current time
- Owner - the system owner
- Version - current eDeveloper version number
- App - the application number

App has the following special parameters:

Parameter	Meaning
Name	show the application name instead of its number
Path	show the application's file prefix, including the full path
Pref	the application's two-character file prefix
Ctl_File	the name of the application's control file
Rep_File	the name of the application's RPR file
DBMS	the application's DBMS type
D_B	the application's database

Simple Network Management Protocol

21

The Simple Network Management Protocol (SNMP) governs network management and monitors network devices and their functions. Applications and various eDeveloper modules can send trap messages to a pre-configured Network Management Station (NMS), also referred to as an SNMP monitor, which lets the system administrator query and control an application or eDeveloper module when alarms, failures, or other exceptional events occur.

In this chapter:

- | |
|--|
| • SNMP Implementation |
| • Network Management Station Query from eDeveloper |
| • NMS Management Options |
| • Installation and Configuration |

SNMP Implementation

The Simple Network Management Protocol (SNMP) can only be implemented for eDeveloper if the SNMP agent is installed on the operating system.

The eDeveloper SNMP extension, the Mgsnmp.dll file, is placed in the eDeveloper root during installation.

eDeveloper supplies two Management Information Base (MIB) files, Magic.mib and Magic_trap.mib, located in the Magic Support directory. The MIB files are compiled according to the Network Management Station (NMS) version installed. It is recommended to copy the MIB files to the NMS server.

You can set various values for the SNMP parameters, defined in the Mgreg.ini and Mgrb.ini files, to specify when the Magic Requester and Magic Requester Broker send trap messages.

eDeveloper Requester Settings

The Mreq.ini settings below can be specified under the [SNMP] section.

EnableAgent = Y, N

This property determines whether the eDeveloper SNMP agent extension is active. If EnableAgent = Yes, the NMS is enabled to monitor eDeveloper.

The EnableAgent setting is automatically set to Yes during installation only if the eDeveloper SNMP extension is installed.

EnableTraps = Y, N

This property determines whether eDeveloper sends trap messages to the SNMP monitor.

NMSAddress

You can enter the Host name or TCP/IP address of the server that has the NMS installed.

Version = 1,2

You can specify the NMS version.

Community

Enter a community name. The community name acts as a password to the NMS.

ReportStatusCode

You can select the error codes that are sent to the SNMP monitor. They can be displayed as a list separated by commas, a range of values, or as both; for example: 103,104,105, or 102-107,109. To monitor a full range of error codes, you can enter 0-9999.

Although the error codes are negative values, they are displayed as positive values to prevent confusion when the error codes are displayed as a range of values.

Magic Request Broker Settings

The Mgrb.ini settings below can be set under the [SNMP] section.

AverageWaitTime = N seconds

You can enter the average time in seconds that a request waits for an available enterprise server. If the average wait time exceeds the AverageWaitTime value, an SNMP trap message is sent to the Network Management Station (NMS).

AverageProcessTime = N seconds

You can enter the average time in seconds for a request to be processed in an enterprise server. The threshold is checked for all registered enterprise servers each time a request is sent. If the average process time exceeds the AverageProcessTime value, a trap message is sent to the NMS.

DelayThresholdTraps = N in minutes

You can enter the time in minutes that a request of a specified type has to wait between threshold traps to avoid flooding the Network Management Station (NMS). The default value is one minute.

Environment Settings

The SNMP Database Connections Utilization Threshold environment setting, on the External tab in the Environment dialog box, lets you enter a percentage number that determines how many open connections are permitted for a DBMS before eDeveloper sends a trap message.

For example, if the Maximum Connection property is set to 10 for a specified DBMS and the SNMP Database Connections Utilization Threshold is set to 50, eDeveloper sends a trap message to the NMS when 5 connections (50% of 10) to the DBMS have been opened.

For more information, see the SNMP Database Connections Utilization Threshold environment setting in Chapter 2, Settings.

SNMPNotify Function

This function lets the eDeveloper application send an application-defined trap message to an NMS. For more information, see the SNMPNotify function in Chapter 8, Expression Rules.

Other Traps

When an alarm or error occurs outside of the application development process, a trap message is sent to the NMS.

Examples of the types of trap messages sent are:

- A thread crash or fatal error of eDeveloper or the eDeveloper gateway.
- Termination due to the license limit.

- The application cannot connect to a specified DBMS.
- Database connection threshold was exceeded.
- An enterprise server was stopped and started. The trap message contains the address of the enterprise server.
- An enterprise server aborted its connection to the Magic request broker.

Network Management Station Query from eDeveloper

eDeveloper SNMP implementation provides various query options for the enterprise server and request broker from the NMS.

Enterprise Servers (QUE=RT)

When you define a QueryRTTable query, the NMS displays a list of query options. Use the keywords in the table below to define the selected view:

#	Query Option	MIB Node
1	Unique Identifier	EntServerEntryIndex
2	Enterprise server's host name	EntServerHost
3	Enterprise server's port number	EntServerPort
4	Name of the opened application	openedApplication
5	Enterprise server priority	EntPriority
6	Running thread counter	runningThreads
7	Peak thread count value	peakThreads
8	Current context count value	runningContexts
9	Peak context count value	peakContexts
10	Number of requests served	requestsServed

#	Query Option	MIB Node
11	Number of execution errors	executionErrors

Requested Query (QUE=QUE)

When you define a QueryQueTable query, the NMS displays a list of pending requests with the following information. Use the keywords in the table below to define the selected view:

#	Query Option	MIB Node
1	Unique identifier	EntServerQueEntryIndex
2	Application name	application
3	User name	userName
4	Priority number of the request	priorityNum
5	Request identifier	requestId
6	Client host name	clientHostName
7	Client process identifier	clientProcessId
8	Request submitting time	requestSubmittingTime

Loaded Query (QUE=LOAD)

When you define a QueryLoad query, the NMS returns the following statistical information about the broker. Use the keywords in the table below to define the selected view.

#	Query Option	MIB Node
1	Broker's host address	brokerHost
2	Broker's port number	brokerPort
3	Average wait time	averageWaitTime
4	Total number of requests submitted to the Magic Request Broker	totalRequestsNumber

#	Query Option	MIB Node
5	Total number of pending requests	pendingRequestsNumber
6	Total number of requests in progress	inProgressRequestsNumber
7	Total number of completed requests	completedRequestsNumber
8	Total number of failed requests	failedRequestsNumber

NMS Management Options

eDeveloper SNMP implementation provides various management options for the enterprise servers and request broker from the NMS. Use the keywords in the table below, which are in italics, to enter Set commands from the eDeveloper Request command line.

Set Commands	MIB Node
Stop enterprise server:	shutdownEntServerRequest
<i>host/port</i>	shutdownEntServer
<i>graceful</i>	graceful

Due to the limitations of some Network Management Station software that allow for only one parameter, the following syntax can be used for the Stop enterprise server: *host/port,T* (T denotes a graceful shutdown)

Set Commands	MIB Node
Start enterprise server: <i>application to load</i> <i>startup parameters</i>	startEntServerRequest exeEntryEntServer startupParameters
<p>Due to the limitations of some Network Management Station software that allow for only one parameter, the following syntax can be used for the Start enterprise server: <i>background,-LoadMonitor</i></p>	
Enterprise server priority: <i>enterprise server name</i> <i>priority</i>	priorityEntServerRequest priorityEntServer priority
<p>The priority number must be between 1 (highest) to 5 as specified by the Load Balancing Priority property.</p>	
<p>Due to the limitations of some Network Management Station software that allow for only one parameter, the following syntax can be used for the Enterprise server priority: <i>host/port,4</i> (priority number)</p>	
Reset peak thread count server: <i>enterprise server name</i>	resetPeakThreadCount resetEntServerThreadCount
Reset peak context server: <i>enterprise server name</i>	resetPeakContextCount resetEntServerPeakContext Count
Reset all number-of-requests: served counters <i>enterprise server name</i>	resetNumRequestServed resetEntServerNumRequest Served
Reset all number-of-execution: error counters <i>enterprise server name</i>	resetNumExecutionError resetEntServerNumExecutio nError

Set Commands	MIB Node
Reset all:	resetAllCounters
<i>enterprise server name</i>	resetEntServerAllCounters

i When the value of the host or port parameter is an asterisk (*), the set command affects all running enterprise servers.

Installation and Configuration

For eDeveloper to be SNMP-enabled, SNMP must be started from the operating system. When the operating system is SNMP-enabled, eDeveloper automatically installs the SNMP component in the typical installation. For example, in Microsoft® Windows®, the Microsoft® SNMP Agent service must be started.

When the SNMP option has been selected, the installation process automatically sets the EnableAgent setting to Yes. In Microsoft® Windows®, the SNMP service is restarted so that the changes to the eDeveloper requesters can take effect immediately.

Supported SNMP Agents

The supported SNMP agents are listed below.

- Windows® - Microsoft® SNMP Agent
- UNIX - Net-SNMP Agent
- iSeries - Native SNMP Agent

For each supported SNMP Agent, the following values should be configured:

- Security - The Community property should have Read and Write rights specified.
- Traps - The Host Trap destination should be defined. Usually, these values are the same as the values in the Mgreg.ini file.

Magic eDeveloper allows you to generate Enterprise JavaBeans (EJBs). Clients operating within a J2EE environment are able to view and activate programs as EJB methods.

This chapter explains the process of creating an EJB using eDeveloper's Component Builder Utility, and how it connects with eDeveloper's enterprise server.

In this chapter:

• J2EE Terminology
• Component Builder
• Enterprise JavaBean
• eDeveloper Configuration and Deployment

Terminology

The following is a list of standard terminology used in relation to the J2EE environment:

J2EE	Java 2 Enterprise Edition An architecture for developing, deploying, and executing applications in a distributed environment.
EJB	Enterprise JavaBeans These are server components written in Java that are accessible from various types of clients in a J2EE environment.
JAR	Java Archive file This file is used to store EJBs, clients and applications. Jar is the file name extension.
Container	A container is a runtime environment that controls enterprise beans and provides them with system-level services. EJBs run within containers.
DD	Deployment Descriptor An .xml file that describes the component.

**Stateless
session bean**

The container selects, and creates if needed, an instance of the EJB for each method invocation.

Data inside the EJB is maintained across method invocations, for example a connection to an external source.

However, a client using the EJB will not get the same instance for each method invocation, therefore the client cannot maintain any personal state across method invocations.

No context will be maintained inside the enterprise server across method invocations. A new context will be created for the duration of each request, as for any other request which is not a browser-based program

The Component Builder

eDeveloper lets you create EJB components using your existing programs. The Component Builder has enhanced functionality, which generates and packages the modules you want to deploy in a J2EE environment, and saves them in the form of a .jar file. This file will contain all the required modules including .class files & DDs.

Use of the eDeveloper broker is optional. If you make the broker active, it can be used to query the status of the enterprise servers, to load new enterprise servers, and to terminate existing enterprise servers.

Defining the EJB

When you create an EJB using the component builder you can only select programs. Models, tables and other objects that you can normally include

when building a component are not available. These objects are only available internally, when a program is invoked.

Naming

There are important naming issues to be aware of when creating the EJB component.

- The component's name is used as the class name.
Note: The component name is not used to locate the EJB. The JNDI (Java Naming and Directory Interface) name, which is set during deployment, is used for this purpose.
- Each program's public name is used as the method name.
- For each parameter, the parameter's name in the selected program is used as the parameter's name in the Java code.

The Component Builder validates the names of components, programs, and parameters.

- If the names are invalid, the component names and programs must be modified in the original application in which the Component Builder was activated.
- Names of invalid parameters are modified while the .jar file is being generated. For example, spaces are replaced with an underscore.

Returned values

Task Returned Value

- The task's Returned Value, which is specified in the Task Properties dialog box, is used as the method's returned value.
- An IO Requester type is disregarded and will not be returned to the EJB.

Compound returned values

- Compound returned values include arrays and structures. They are available by using user-defined XMLs, whose content is recognized both by the activated eDeveloper program and the client invoking the EJB.
- The enterprise server passes the XML to the EJB, which in turn passes it to the client as it is, without parsing the XML.
- You can prepare the XML file either using alpha/memo fields or BLOB. With the BLOB method you work with a text file and then load it using the File2Blob function.

EJB Component Builder Screen

The EJB Interface Builder enables an eDeveloper application to be exposed as an EJB in a J2EE enterprise server.

Click the EJB Interface from the Components menu to create a Jar file. The EJB repository opens, as shown in Figure 22-1.

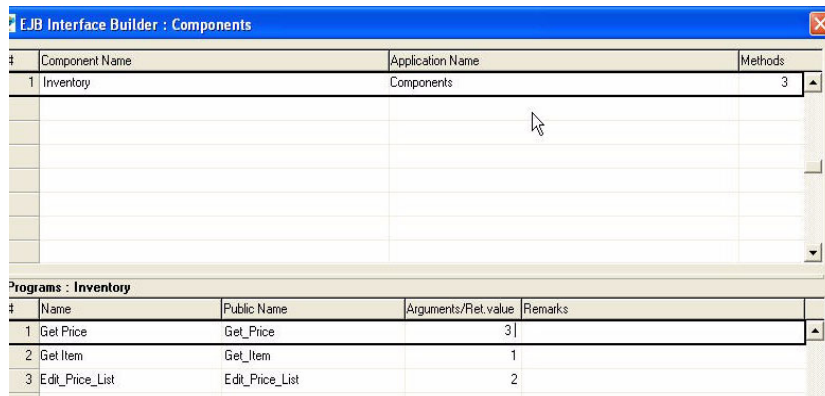


Figure 22-1 The EJB Component Builder Screen

In the lower half of the screen you select which programs to include by clicking Select Program. You can zoom from the Arguments/Ret. value column to open the Arguments dialog box as shown in 22-2. All the Select Parameter fields of the selected program are displayed in the table. They are later generated into the Java code.

#	Argument Name	Java Type	Class
1	Product Name	String	
2	Product Price	long	
3	Date	Object	

Return Value

Expression: Date

Java Type: Object Class: java.util.Date

OK

Figure 22-2 Arguments Dialog Box

The Argument name is taken from the parameters table of the selected program. The Java Type is calculated according to the field's picture defined in the selected program.

A Returned Value can only be an expression. The Expression column displays the textual image of the expression. The Java Type list only displays the allowed types according to the attribute or picture definitions defined in the program field or expression.

The **Precision** slider is only visible for expressions that can return a numeric value. The Java Type list and Expression field will be unavailable for numeric arguments and for programs without a returned value.

Comparing eDeveloper and Java types

The following table shows the different eDeveloper types with the corresponding Java types:

eDeveloper type	Java type
Alpha	String / StringBuffer
BLOB	String / StringBuffer / byte
Numeric	Byte / short / long / float / double
Logical	Boolean
Date/Time	String

- You can choose Java types for the parameters and returned values that are either **BLOB** or **Alpha** fields, or expressions that can return **BLOB** values such as File2Blb or **Alpha** fields (Str).
- To maintain accuracy, **Numeric** parameters must properly map the parameters of the activated program. The Component Builder assigns the Java type based on the parameter's attribute and picture, using the same rules used to assign a storage type to fields of database tables. You cannot choose another type.
- For a **Numeric** task's returned value, i.e. an expression returning a numeric value, the Component Builder does let you choose the Java type. If the selected type is **Float** or **Double**, you have to specify the picture.

The default type is **Double** with a **Precision** of 2. If **Precision** is 0, the Java Type changes to **Long**.

EJB Settings

From the Options menu, click Settings to select from the following J2EE enterprise servers:

- BEA WebLogic Version 5 or 6
- Sun Reference Version 1.3

- JBOSS
- Oracle Enterprise Server Version 9i
- Sun ONE Enterprise Server Version 7
- Fujitsu Interstage Enterprise Server Version 5
- WebSphere

Creating the JAR file

The process of generating the .jar file is similar to that of creating a regular Magic Component Interface file. From the Options menu, you can choose “Build Jar File” instead of the regular “Build Interface File” option.

The **Build Jar File** dialog box is displayed, prompting you for input, if any or all of the following cannot be found:

- **Intermediate files location**
Input is only required if the TEMP environment variable is not found.
- **J2EE Home Directory**
Input is only required if the J2EE_HOME environment variable is not found.
- **Java Home Directory**
Input is only required if the JAVA_HOME environment variable is not found.

You can either type or zoom to enter the correct path.

When the JAR generation process is complete the Component Builder validates that the JAR has been created successfully and displays a message with the following information:

Bean XYZ was packed into path\XYZ.jar, and is ready for deployment

The message also contains a **View Script** and a **View Log** button.



You must install JDK Version 1.3 or higher to be able to create the JAR file. If you are creating a JAR file for either WebLogic, WebSphere or Sun, you need to have these products installed on your machine.

EJB Configuration

The Magic Component Builder creates a Deployment Descriptor (DD) file that describes the newly created EJB and packs it into the JAR file. This file contains the information the bean needs to operate. The two types of settings are Environment and Resources.

EJB Environment Definitions

Magicenterprise servers

This setting, in the form of a string, indicates the enterprise servers with which the EJB can connect. Each name, delimited by ',' references a URL resource, as described in the Resources section on page page 1391.

Syntax: MAGICURL1[,MAGICURL2..]

Default: MAGICURL1

MagicApplication

This setting, in the form of a string, indicates the name of the eDeveloper application with which the EJB should connect.

Syntax: (name of the application)

Communication Timeout

This setting, which is numeric, determines the time, in seconds, to allow for connecting, sending or receiving requests between the EJB & the enterprise

server. The Timeout setting does not include the time spent in executing the request.

Syntax: nnn (<=0 is handled using the default value)
Default: 10

Resources

MAGICURL1

This setting represent a URL of the enterprise server. MAGICURL1 represents a single resource. You can however use additional resources if you allowed for this in the eDeveloper enterprise servers setting.

Default: http://localhost:1500

eDeveloper Configuration and Deployment

The EJB will only be able to work when interacting with an enterprise server that supports J2EE. This section outlines the different factors impacting on the EJB's interaction with the enterprise server.

Environment

An EJB created by eDeveloper can only be used with an eDeveloper enterprise server.

eDeveloper clients executing remote calls will not be allowed to reference J2EE-type servers from the services table. If an EJB does attempt to reference another server during checker and remote call activation, the following error message will be displayed:

J2EE type servers cannot be used to send remote calls

Runtime

The EJB cannot provide the enterprise server with user information such as username and password. This information can only be provided locally in the enterprise server. You can do this in the Magic.ini file. You can also use command line parameters, such as the following example:

```
mgrntw.exe -User= -Password=
```

Generic Messaging Layer (mgrqgnrc.dll)

The Settings and Behavior settings of the Generic Messaging Layer's enterprise server layer must be defined to ensure that the eDeveloper enterprise server can accept requests from EJBs. No changes are required in the engine modules.

Environment (Mgreq.ini)

The definitions for the Environment property relate to the use of keywords.

Gateway=5

When you set **Gateway** to **5** this tells the enterprise server that the gateway is J2EE. This has the following consequences:

- The Generic Messaging Layer is signalled that EJB is supported.
- No new gateway module will be loaded, as with MQ/DQ/Corba.
- Requests will not be accepted from other sources that are not EJBs.

Other Keywords

- The **Log** keyword remains functional when EJB is supported.
- Other keywords are not relevant when EJB is supported.

Enterprise Server Layer

The Enterprise Server layer has three settings which are important for determining how the EJB interacts with the enterprise server. These settings are Initialization, License Validation, and Status Messages to the Broker.

Initialization

When **Gateway=5**, the messaging server passed from the enterprise server is optional. If a broker exists at the specified address, it will be used for login or administration purposes. If a broker is not present, the enterprise server filters out all messages sent to the broker when **Gateway=1**.

The enterprise server will be loaded on a port, specified in [MAGIC_COMMS]TCP/IP.

License validation

The license can limit the number of hits or concurrent requests on an enterprise server. When **Gateway=1**, this task is performed by the broker. However, when EJB is supported, license validation is carried out in the enterprise server.

Where the number of hits or concurrent requests is exceeded, the enterprise server will self-terminate on the next request (error 136), and subsequent hits will be returned with error 109. If the enterprise server is unable to serve a request because the number of concurrent requests was reached, it will return error 104.

For more information, see the Errors section on page 1397.

Status Messages to the Broker

When **Gateway=5**, and the messaging server sent from the enterprise server is acknowledged as an active Broker, it will be used for the enterprise server's logging or administration purposes. Request completion messages will not be sent as they have no request ID and are therefore meaningless to the Broker.

Broker Configuration and Deployment

Query-Only Enterprise Servers

The broker must be aware that some of the enterprise servers are "Query-Only" and they will then be marked as such. Query-only enterprise servers can only send status information and can neither accept requests from the broker nor be assigned to requesters.

Termination

If the Broker is terminated when EJB is supported, it will send XML-based termination requests to Query-only enterprise servers.

Loading enterprise servers

You can load an enterprise server during initialization, from command line requesters or from RQ functions.

Command Line Requester

Termination

A broker is required for password validation. Without a broker, the enterprise server can be terminated by the OS as with any other process.

Queries

A Broker is required.

Remote calls from other requesters (not EJBs)

Remote calls from any other requester, such as a command line or Web, when **Gateway=5** will not be functional.

J2EE and eDeveloper Installation

When you install eDeveloper, eDeveloper searches for a J2EE server. When eDeveloper finds the J2EE server it copies a special .jar file to a directory on the J2EE server: **%JAVA_HOME%\jre\lib\ext**. The file, Mgejbgnrc.jar, enables EJBs to interact with eDeveloper enterprise servers.

If eDeveloper cannot find the J2EE server, the .jar file is copied into an **EJB** folder in eDeveloper's default directory. Later, you should manually copy the file over to the **%JAVA_HOME%\jre\lib\ext** directory on the computer where the J2EE server is located.

Connection Difficulties

Where an EJB has problems connecting to the enterprise server, the EJB attempts to connect with another enterprise server if possible. If the EJB is still unsuccessful in connecting, either to the original enterprise server or to the alternatives, the EJB will throw an "exception". The eDeveloper exception contains the numeric return code, as described in the requester documentation, and the name of the URL resource that was used, for example MAGICURL1.

Error Types

The following are cases when the eDeveloper enterprise server returns an error, and what the errors mean.

Code	Description	Occurrence	Action
104	Application in use	During License validation	The EJB continues to connect to the enterprise server for as long as allowed in the CommunicationTimeout setting. If the EJB fails to connect it will throw the "ApplicationBusy" exception to the client that activated it.
109	Connection to the enterprise server refused	During termination	Verify that the enterprise server you wish to terminate is at the requested address.
117	J2EE type servers can only accept remote calls from EJBs	Following a remote call by a different requester	Set Gateway=5 in the Mgreq.ini file, to ensure that users cannot send remote calls from a command line or web requester in this directory.

Code	Description	Occurrence	Action
128	Application rejected	When a request is accepted by an enterprise server that has another application open	The EJB continues to connect to the enterprise server for as long as allowed in the CommunicationTimeout setting. If the EJB fails to connect it will throw the "ApplicationBusy" exception to the client that activated it.
129	Mode rejection	When a request is accepted by an enterprise server which is Busy-Toolkit or Avail-Running on another application	Switch the enterprise server to Avail-Idle.
136	Maximum number of hits was reached	During license validation	Enable more license access than allowed for in the enterprise server
138	Program aborted during execution		The EJB includes error messages from the aborted program in an exception thrown to the EJB client.

Multi-User Considerations

23

The method and degree of data-sharing are major concerns for the development of multi-user applications. The developer must consider the implications of multiple users accessing the same data simultaneously. This chapter focuses on multi-user considerations, and the many issues that are inherent when users share the same data.

In this chapter:

• Definitions
• Concurrency
• Table Modes
• Setting the Multi-User Environment
• Context

Definitions

The following are terms necessary for the understanding of the multi-user issues explained in this chapter.

Isolation Level

The Isolation level determines what happens during the concurrent (simultaneous) use of the same transaction. The user can change the Isolation

level in the database only. Changes made in the DBMS Properties dialog box has an effect on the system.

The types of situations that can occur are:

- Dirty Reads return different results within a single transaction when an SQL operation accesses an uncommitted or modified record created by another transaction. For example, one user can view the changes made to the data by another user before the transaction is committed. If a rollback occurs, the data viewed will still reflect the change. Dirty Reads increase concurrency, but reduce consistency.
- Non-Repeatable Reads return different results within a single transaction when an SQL operation reads the same row in a table twice. Non-Repeatable Reads can occur when another transaction modifies and commits a change to the row between reads within the same transaction. Non-repeatable reads increase consistency, but reduce concurrency.
- Phantoms return different results within a single transaction when an SQL operation retrieves a range of data values twice. Phantoms can occur if another transaction inserted a new record and committed the insertion between executions of the range retrieval.

Each Isolation level behaves differently for the situations described above.

#	Isolation Level 1	Dirty Read	Non-Repeatable	Phantom
0	Read uncommitted	X	X	X
1	Read committed		X	X
2	Repeatable read			X
3	Serializable			

Locks

Locks are used to preserve data integrity. When a user updates data, some notification should appear, so other users do not update the same data. Other users of the same data must receive a notification that the data is currently being updated. This notification is called a lock.

When no lock is issued, updated information can be lost. If each user transaction is unaware of the other, the last update overwrites the other updates.

Process

A process is an instance of a program running in a computer. It is close in meaning to a task, a term used in some operating systems. Like a task, a process is a running program with which a particular set of data is associated so that the process can be kept track of. An application that is being shared by multiple users will generally have one process at some stage of execution for each user.

Transactions

A transaction is a sequence of Data Manipulation steps in the database that must be completed in its entirety before the database is actually updated. If something fails before the sequence is completed successfully, all changes will be undone (this is known as rollback). In other words, either the whole sequence is updated or none of it. This also preserves data integrity.

Concurrency

Concurrency is when at least two users are accessing the same application at the same time. This section focuses on how to allow concurrent users to access the same data without any loss of data. The following example illustrates the importance of data integrity in a concurrent work environment.

Two users are accessing a database table called Stock Level. Currently, there are 94 items in stock. The first user enters the Total in Stock column to update the item number to 93 (having sold an item). The second user enters the Total in Stock column to update the item number to 92 (having sold two items). Three items have been sold, but the number of items only reflect the sale of two items. Data was lost because of a lack of data integrity.

Locking

If a transaction cannot open a table or record at runtime, because of a potential conflict with another transaction, eDeveloper will continue trying to open the table or record that is currently locked. While waiting for access, eDeveloper will display a message informing the end-user that the table or record is currently locked. eDeveloper's attempt to enter the table or record ends when either the end-user presses Exit (**ESC**) to abort the task, or when the transaction of the other user ends the transaction. Note that if the transactions of both users were non-conflicting, neither would have to wait to gain access to the table or record.

eDeveloper implements two kinds of locking methods: Database locks and Developer locks.

- A database lock is a lock that uses the methods specified by the specified DBMS. For example, using SELECT...FOR UPDATE in Oracle and SELECT...UPDLOCK in MSSQL.
- An eDeveloper lock uses the mglock to perform its logical locking method.

By default, all SQL gateways except for ODBC and Cache use physical locks as their locking strategy.

Identify Modified Row

The Identify Modified Row column in the DB Table repository lets you determine how eDeveloper will identify that a row is being updated. The option selected can determine the access of other users to fields in the row as described in Position and Update Fields and Position and Selected Fields

options. For a complete description of the Identify Modified Row column, see the DB Table Repository section in Chapter 6, Programs.

The options for the Identify Modified Row column are:

- **Position** – This is the lowest level. eDeveloper updates the record in the database using a unique identifier only, such as RowID in Oracle. In this case the last update will be the successful one. This may cause problems when two users need to update the same field. In general, unless the identifier has been modified (which in Oracle is improbable) or else the row has been deleted, eDeveloper will not identify this row as being updated, which may result in the loss of updated data.
- **Position and Updated Fields** – eDeveloper updates the record in the database using a unique identifier plus those columns that have been updated. For example if a user accesses the record to update the Total in Stock column, then eDeveloper will fetch the record (for update) using the unique identifier plus the Total in Stock column. In this method, if another user tries to update the Total in Stock column, one of them will receive an error message indicating the loss of their updated data. But, if one user accesses the Total in Stock column and the other user accesses the Total on Order column, no loss of data occurs.
- **Position and Selected Fields** – eDeveloper updates the record in the database using a unique identifier plus all the columns selected by the developer for use in that task. In other words, if in the same task, the developer used both the Total in Stock column and the Total on Order column, eDeveloper fetches the record (for update) using the unique identifier plus the Total in Stock column as well as the Total on Order column. In this method, any changes made to either of those columns, will result in an error message indicating the loss of updated data.

For example, if in a database table where there is a record with the Total in Stock and Total in Order fields that are being updated by User A and User B

concurrently, depending on Identify Modified Row option, the result will be as described in the table below.

Identify Modified Row Options	Total in Stock Updated By	Total in Order Updated By	Loss of Data	Error Message
Position	User A and User B	User A and User B	Yes	No
Position	User A	User B	No	No
Position and Updated Fields	User A and User B	User A and User B	Yes	Yes
Position and Updated Fields	User A	User B	No	No
Position and Selected Fields	User A	User B	Yes	Yes
Position and Selected Fields	User B	User A	Yes	Yes

Transactions

Much thought should be given to the subject of transactions. On the one hand transactions increase data integrity due to the use of the locking mechanism, but also reduces concurrency. The longer the transaction, the more concurrency is reduced. You can select the Transaction mode from the Enhanced tab in the Task Properties dialog. The options for the Transaction Mode property are:

- **Deferred Transactions** - All Data Manipulation operations are logged by eDeveloper in the Cache memory. Other users will have no access to that data. Once this transaction ends, eDeveloper updates the database with the various operations that were logged during the Deferred transaction. The updating of the database is performed according to the Identify Modified Row option described above. If data for the selected records has been modified before you have committed your changes, an error message will be displayed and your updates will be lost. The only course of action is to redo your updates. There is always a risk that updated data can be lost when working in the Deferred transaction mode.

Note that Deferred transactions are the only mode that is valid in a Browser Client task.

- Physical Transactions - All Data Manipulation operations are logged by eDeveloper in the rollback segment of a physical database. Other users will be able to see the updated data depending on the defined Isolation Level explained under the Definitions section. As a result, there is less probability of losing updated data.

For more information about Transaction modes, see Chapter 11, Data Management.

Note that in ODBC and Cache DBMS systems, there are no physical locks and all transactions are implemented by eDeveloper.

Task Level Transaction Usage Considerations

Selecting the transaction to begin Before Task Prefix, causes all updates to be logged as a single transaction. This has various implications for multiple users as it reduces concurrency but it does increase performance. The following suggestions should be considered:

- For improved concurrency, it is better to begin transactions at the Task Prefix handler for batch tasks with a dataview that includes only a few records. Of course, this depends on your application, as there may be instances in which you are required to lock many records. Note that if the user is working in Deferred mode, it is not recommended to begin the transaction before the Task Prefix handler since these operations are stored in Cache memory, and memory is limited.
- In Online tasks, tables can be locked for long periods of time. Once again, there may be instances in which this is required.
- Be aware that if a task is called within this transaction, those locked rows will also be part of the transaction and thus unavailable to other users.
- When entering a task in which either a transaction is already open or else there is a task level transaction, Oracle locks the entire dataview of that task (that is, all records in the range). In some cases, this is the intended behavior. You should be aware that if you only want to lock some of the records, you should adjust the range accordingly. This behavior is

beneficial for data integrity and performance, as all records are locked while they are being fetched to create the dataview. In other databases, the data is only locked when parked on the record. This affects performance, because for each record the data is locked. But on the other hand, concurrency is maximized.

Before Record Prefix Transaction Usage Considerations

The transaction is opened before the dataview is fetched, that is many records are locked at a time. For online tasks where the human factor determines the duration of time that a dataview is locked, it is not recommended to select the Record Prefix Transaction option in a multi-user environment.

Before Record Update Transaction Usage Considerations

When you select the Record Update Transaction option, The transaction is opened just before updating the record. Only then does eDeveloper send a request to lock the record, according to the Identify Modified Rows column in the DB Table repository. The advantage is that the transaction is small, which maximizes concurrency. But remember that because the lock is performed at the last minute, another user may have updated the data. Therefore, there is a risk of losing updated data.

Locking Strategy

The Lock Strategy property, available for online and batch tasks, defines when eDeveloper issues a lock to prevent other users from accessing the record processed by the current task, and whether verification is used for the information contained in the memory. Verification is performed according to the selected Identify Modified Rows option, which determines the way eDeveloper re-reads the record to update it. For more information about Locking Strategy, see Chapter 11, Data Management.

The Locking Strategy property can have one of the following values:

- Immediate - eDeveloper locks the record immediately as it is fetched and displayed to the end-user. The record is kept locked until it is released. This strategy increases data integrity, but reduces concurrency. Using the

Immediate Locking Strategy in an online, multi-user environment may cause prolonged record locks, as the period of time that a user may remain in the same program is unpredictable. This value applies only to the Physical Transaction mode.

- On Modify - eDeveloper locks the record as soon as the user enters a character in the record for an online task. The On Modify option is the default lock for online tasks. Online tasks by default are created with Deferred Transaction and No Lock options for the Transaction Mode and Locking Strategy task properties.

For batch tasks, the lock is issued by the first Update operation that is executed in a Record Prefix handler or Record Suffix handler, or the first task or program called, which causes a lock (refer to the Call operation in Chapter 7, Operations).

For the On Modify locking strategy, modification verification is required, as described in the Identify Modified Row column in the DB Table repository. eDeveloper will issue a lock in the following cases:

- Modifying an input column.
 - Calling a task or program in which the Lock property is set to Yes.
 - The task executes a non-abortable Update operation.
 - The Import Form operation is executed.
 - Event programs will lock the dataview of the event.
- Before Update - eDeveloper locks the record only before the actual write to the database. This means that the record is not locked during the interaction stage (online) and the Record Suffix operations for online and batch tasks. If the data verification stage of the lock fails (the record was updated by another user since last read), the update will not be carried out, and a message to that effect will be issued to the user. As a result, there is the possibility in this mode of losing updated data. Subtasks called by the current task may have updated information to other tables, thus possibly causing inconsistency if the current record is not updated to the disk. Since the transaction duration is short, concurrency is increased, but

at the expense of data integrity. It is recommended that you should only use the Before Update locking strategy, in cases where locking the underlying database would be problematic, and where the task structure ensures that there would be no loss of update data should the verification fail. This locking strategy option applies only to the Physical Transaction mode.

- No Lock - The record is not locked. The No Lock option should be used with great caution, because in this case eDeveloper may update records that have been accessed and updated by other workstations causing loss of data integrity. No Lock is the default setting for Online and Browser tasks.

Task Nesting and Locking

When calling a task or a program, the processing of the current view record is suspended until the call returns. The called task or program has access (for a program, if columns are passed as arguments) to all the current view record's variables. Therefore, the called task or program may also update some of the displayed columns of the record. When using the On Modify locking strategy, the view record is locked immediately on the first change or after a complete column is accepted. If a called subtask or a program is about to change the current record, the current record should also be locked.

As subtasks and programs are complete units that can take some time to execute, other users should not be allowed to gain access to the current dataview record and modify it while the subtask or program is executing. This would prevent writing the current record to the disk when the called subtask or program terminates. Therefore, the current record should be locked before executing the subtask or program. However, to avoid blocking access to other users, the current view record should not be locked before executing the subtask or program if the called subtask or program is not about to modify the current view record.

The Call operation's Lock property lets the developer lock the record when a task or program is called. Options for the Lock property are:

No - eDeveloper will not lock the current dataview record when executing the task or program. It is the programmer's responsibility to verify that no updates

will occur from the called process to the calling record columns. If such updates do occur, they may be lost if the current view record cannot be written to disk due to the changes made by other users.

Yes - means that the current view record will be locked immediately on executing the called task or program. Use this option when the called task or program is going to update the current view record, or the updates performed by the called task or program must be done together with the current dataview record. Be aware that if the transaction is not yet opened, this mode of the switch is invalid. For instance, having a Call Task in the Record Prefix handler and opening the transaction before the Record Suffix handler.

Isolation Level

When the database has an Isolation level of 0, all locks will be granted to this session. No user will wait, Even if a table is locked exclusively. This Isolation Level setting allows for maximum concurrency, but low data integrity. Depending on the data structure of your application, this Isolation level maximizes concurrency but to the detriment of data integrity. Informix, DB2 and MSSQL support Isolation level = 0, but Oracle does not.

Oracle supports Isolation level = 1 for a user's request to read data that is locked by another user, Oracle will display the previously unmodified data to this user, data that has been committed.

Differential Update

For SQL databases, you can select Differential Update from the Update Style field property. The Differential option lets you minimize the problem of losing updated data.

The following example present a problem that can occur when the Absolute Update option is selected.

There are 94 units of the item in stock (fetched from Total in Stock column), and User A updates the Total in Stock column in order to 93 (having sold an item) and User B updates the Total in Stock column to 92 (having sold 2 items in the same sale). Loss of updated data occurs.

A differential update, however, updates the database as:

- Total in Stock: – (minus) 1 for one user
- Total in Stock: – (minus) 2 for the other user

The end value of Total in Stock is Total in Stock – (minus) 3, which in this example is 91. No loss of updated data occurs.

For more information about this option please refer to Chapter 11, Data Management.

Table Modes

The sharing of data among many workstations or instances demands a process of synchronization, so that access to a table by more than one process is restricted or controlled.

The operations a process can perform on a table are:

- Read (denoted as R) - The process does not intend to update the table.
- Write (denoted as W) - The process might update the table. Write also implies Read. In the technical literature, R is called *Read Only* and W is called *Read/Write*.

Access Mode

In eDeveloper, the *Access mode* is the intended operation a process requests to perform on a table. eDeveloper facilitates controlled access to tables by requiring each process to declare its intended Access mode on the table before it can access it. The built-in automated system manager then decides whether to grant that access. However, the system manager cannot know in advance whether the access modes of two parallel processes will conflict.

For example, Process A requests the read-only access to a table, while Process B requests read-write access to the same table.

The system manager cannot know whether these two Access modes are conflicting. If Process A is simply counting the number of records in the table it is irrelevant if those records are being updated at that time, so the two access modes are compatible. But if Process A is performing statistical calculations, the modes are incompatible, and access to one of the processes should not be granted. Therefore, the system manager needs additional information to determine whether the two modes are compatible. In eDeveloper, this is called the *Share mode*.

Share Mode

In addition to a process declaring an Access mode, every process must declare a Share mode. The Share mode specifies which Access modes are compatible.

A statistical program would access a table in Access mode=R and Share mode=R. Thus it would only read records from the table and would allow any other process that is also only reading records from the table to access it at the same time.

Once the system manager is provided with the Access and Share modes, it can determine whether to grant the request.

To open a table in an Exclusive mode, the Share mode is set to N for None. No other process may be granted access to the table.

Multi-User Considerations When Defining Table Modes

- Table locks are issued when either None or Read is selected for the Share mode of a table within a task. eDeveloper locks this table exclusively. The behavior for an exclusive lock depends on the particular DBMS. Some DBMS systems support exclusive locks, other DBMS systems do not. In Oracle, for example, although the table is locked exclusively, other users are able to read from the table. Therefore the use of Share mode=N is invalid. It is recommended that the developer be aware of support limitations for exclusive locks by the selected DBMS.
- A table lock is only valid within a transaction. Do not use record level transactions with a table lock. eDeveloper will behave as if the entire table

is locked, and will not let additional user's access other records in the table.

- Table locks should be avoided if possible, because they greatly diminish concurrency.
- The developer can issue an eDeveloper lock at the table level. Note that for exclusive locking the same limitations apply, as described above.
- eDeveloper issues a database lock when the Write option has been selected for both the Access mode and Share mode. If the transaction is in a Deferred mode, a database lock is not issued and an eDeveloper lock is issued, using the mglock file. Be aware that the eDeveloper lock is only released once the transaction ends.
- In browser tasks, database locks or table locks do not apply, because there is no physical transaction. Only eDeveloper locks apply to browser tasks.
- In MSSQL, the record locking default is physical locking. eDeveloper sends a `SELECT ...UPDLOCK` command to the database that is similar to the `SELECT ... FOR UPDATE` command in Oracle. This default differs from previous versions of Magic, which was a logical lock (that is, a regular `SELECT` statement was sent to the database using all its selected columns). To change the default, refer to the MSSQL gateway section in Chapter 25, SQL Considerations.

How to Define Table Modes

You define the Access mode and Share mode of a table in a DB Table repository entry.

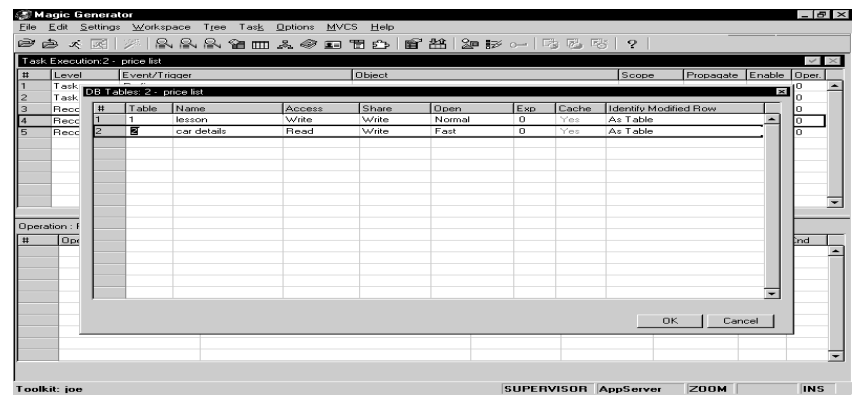


Figure 23-1 A Task DB Table Repository

The DB Table repository of a task includes the two columns labeled as:

- Access - The value can be either W for read/write (the default) or R for read only
- Share - The value can be W for read/write (the default), R for read, or N for none. None means the table is not shared in any mode.

Table Sharing Interaction

The following table describes the sharing interaction between two tasks opening the same table. A plus sign (+) indicates that modes are compatible while a minus sign (-) indicates that modes are incompatible

Process A		
Access Mode	Read	Write

Process B	Mod e	Share Mode		Rea d	Writ e	Non e	Rea d	Writ e	Non e
		Mod e							
	Rea d	Read		+	+	-	+	+	-
		Writ e		+	+	-	+	+	-
		Non e		-	-	-	-	-	-
	Writ e	Read		-	+	-	-	-	-
		Writ e		-	+	-	-	+	-
		Non e		-	-	-	-	-	-

Figure 23-2 Valid Access/Modes

For example, if Process A opens the a table in R/R mode (that is, Access=R and Share=R), Process B can open the same table with modes R/R and R/W.

If process A opens the table in R/W mode, process B can open the table in R/R, R/W, W/R, or W/W modes.

Setting the Multi-User Environment

The Environment dialog contains all of the globally configured eDeveloper properties. These properties reside in the [MAGIC_ENV] section of the Magic.ini file. You can use Environment settings to customize eDeveloper according to the specific needs of the installation. All changes made to settings in the Environment dialog are registered in the Magic.ini file. Some of the setting changes take effect immediately, while others will be effective from the next eDeveloper session.

eDeveloper Environment settings are not related to operating system environment variables. If eDeveloper was invoked with Command Line properties, the values that appear in the Environment dialog will reflect the Command Line properties for the respective components. Command Line options take precedence over the Magic.ini values. However, any modifications you make to the Environment dialog during a development session will also automatically update the Magic.ini file, and will override any corresponding Command Line values.

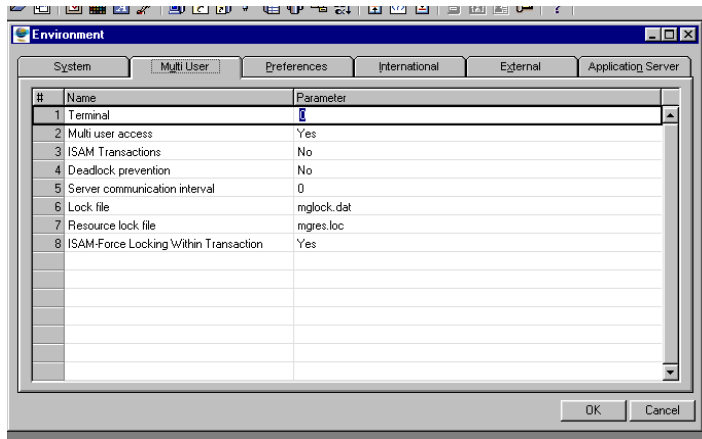


Figure 23-3 Enabling Multi-User Access

Environment dialog properties related to multi-user considerations are:

Multi-User Access

- No - The default value instructs eDeveloper to open all tables for exclusive use (overriding the task table open mode), thereby preventing access from any other workstation.
- Yes - Instructs eDeveloper to implement concurrence controls on all database table access, thereby allowing table-sharing while maintaining database integrity. The default is Yes. Change it to No for a single-user environment.

Terminal

Set Terminal to a unique numeric identifier for each end-user, or set it to 0 for an automatic terminal number resolution.

Lock File

eDeveloper implements record and table locking at two levels. The first level uses the locking mechanism of the underlying vendor-supplied DBMS. The second, optional, level is eDeveloper's own locking mechanism. The eDeveloper locking mechanism involves creating a lock file in every directory that contains at least one of its tables (data or system files).

The first user accessing a shared table in a directory causes the directory's eDeveloper lock file to be opened. Exiting from eDeveloper deletes the eDeveloper lock file.

Context

Note that each context has its own environment. The list below are of objects that are not shared and are only valid for the current context:

- Global variables
- Resident tables
- Memory tables
- Resident INI
- Cache (for Deferred transactions)

Workgroup Development 24

Magic's workgroup development capability maintains concurrence among developers working on the same application. This capability underlies the concept of Team Development.

In this chapter:

• Workgroup Options
• Team Development
• Active Management

Workgroup Options

Workgroup options are defined in the Workgroup tab of the Application Properties dialog. The following settings are available:

Activate Team Development

- Options: Yes and No.
- Default: No.

MVCS Snapshot File

- Zoom from this field to select the MVCS snapshot file for Team Development.

MVCS Lock File Path

- Zoom from this field to select the MVCS Lock File path for Team Development.

Workgroup options can be modified only if the developer has supervisor rights and nothing is currently checked out. For more information on supervisor rights see Chapter 13, Authorization System.

Objects

The basic unit for manipulating an application is an object. All Team Development operations are performed on an object basis.

Check Out/Check In Mechanism

The Workgroup Development system provides a check out/check in mechanism for every repository and program. A repository must be checked out as a whole unit. A Program, however, can be checked out individually.

The MVCS pulldown menu provides the following commands:

- Check Out Object
- Check In Object
- Checked Out Object List
- UnCheck Object
- Resync Objects
- Check Out Program Repository
- Check In Program Repository

Check Out Object

A developer can edit an object only after checking it out. Without checking out the object, the developer is limited to read-only access. Note that although the Form Editor will not appear to be read only, any modifications applied to the form will not be applied if the task is not checked out first.

All repositories, with the exception of the Program repository, are checked out as a whole unit. Programs are checked out individually. You are required to check out the Program repository, which is also considered an object, whenever you modify the Program list.

Checking out a repository automatically locks that object from any other developer. Other developers then have only read-only access to the last updated version of the repository.

When an object is checked out under Team Development, any modification is written into the developer's temporary snapshot file. The latter file can be accessed only by the developer who created it.

A program must be checked out before it can be deleted.

In an application under Team Development, checking in an object updates and synchronizes the MCF.

The Checked-Out Object list provides a list of objects that are checked out.

Figure 24-1 Checked Object List

- Object name - Name of one of the basic application objects.
- Locked by - Identification of the user who checked out the object.
- Locked on - Date and time when the object was locked.

Click one of the column headers to sort the Object List information by that column.

- Users - Click Users to display a list of the active users in the application workgroup.
- OK - Click OK to exit the Object List.
- Refresh - Click Refresh to update the Checked Object List.

UnCheck Object

The developer can uncheck every checked-out object. This cancels any modifications made to the object, and cancels its checked-out status.

The Workgroup (MVCS) Menu

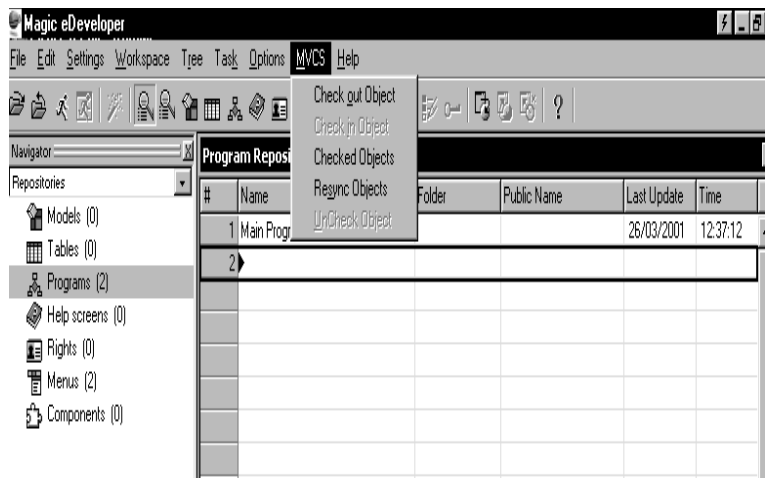


Figure 24-2 The MVCS Pulldown Menu

The MVCS options are described in the table below.

Use:	To:
Check Out Object	Check out the selected object. Enabled only when the object is not currently checked out.
Check In Object	Check in the selected object. Enabled only when the object is currently checked out by the same developer. All current modifications to the object will be saved and other developers will be able to view the modified objects.
Object List	Display a list of objects currently checked out.
Resync Objects	Synchronizes the objects.

Use:	To:
Uncheck Object	Uncheck the current object, canceling all modifications made to the object since it was checked out.
Check Out Prog Rep	Check out the Program repository to make changes in the Program list.
Check In Prog Rep	Check in the Program repository to make available to other developers the changes you made in the Program list.

Team Development

Team Development maintains concurrence among developers. Up to 128 developers can work on a single application file using the Team Development feature. By maintaining locks, the team development interface controls concurrence among developers and helps synchronize the application.

Requirements for Team Development

- Each developer in the workgroup who accesses the application must have a unique User ID.
- The system must be configured for multi-user access.
- Development cannot be performed on an application currently being accessed by users in Runtime using the MGRNTW executable.

Activation of Team Development

Set the Activate Team Development property on the Workgroup tab of the Application Properties dialog to Yes. When Team Development is active, the Navigator list bag displays the MVCS option. When you click MVCS, all check out objects appear and can be selected for the enabled MVCS options.

Snapshot File

An object must be *checked out* before it can be modified in an application.

When an object is checked out, any modification is written to a temporary file that holds all the changes until the object is checked back in. This temporary file mechanism allows the rest of the development team to continue work on the application.

The file that contains the temporary modifications is called the snapshot file. The snapshot file name can be defined in the Application Properties dialog. If a snapshot file name is not defined, a default filename is created in the format `xxSNP.MCF`, where `xx` is the application prefix.

Concurrency

Concurrency among developers is maintained by volatile and non-volatile locks on the application. The non-volatile locks are placed on the application file itself, indicating that an object has been checked out. Volatile locking is added to indicate activity on the application, and is implemented by the lock file.

Modifications to the Program Repository

The Program repository maintains the application's Program list order. Every Program list modification requires updating the Program repository.

The Program repository is not automatically checked-out and checked-in for modifications to the Program list. The developer is required to manually check out and check in the Program repository.

You can only modify the Program list by checking out the Program repository. The Program list can be modified by:

- Creating a new program
- Deleting a program
- Changing the sequence of programs
- Moving a program to another program folder

- Renaming a program folder
- Overwriting a program
- Copying programs
- Assigning rights to the Program repository
- Importing a program
- Generating a program from the Table repository

Check Out and Check In of the Program Repository

You can check out or check in the Program repository by clicking the Check Out or Check In options as displayed on the MVCS menu, or by clicking the Check Out or Check In toolbar buttons. You cannot uncheck the Program repository.

Importing Programs

When you import an application that has programs, you are required to check out the Program repository. When importing an eDeveloper V9.3 export file, eDeveloper prompts you to check out the Program repository. If you do not check out the Program repository or the Program repository is checked out by another developer, eDeveloper issues a warning that the import process cannot be performed.

When you import an application of a previous eDeveloper version, the import process cannot detect the programs included. You will only be prompted when eDeveloper, through the import process, finds a program. If you do not check out the Program repository or if the Program repository is checked out by another developer, eDeveloper issues a warning that the import process cannot be performed. The import process will fail and the application will be rolled back.

Lock File

The application Lock file is called xxMVCS.LOC and resides in a directory defined in the MVCS Lock File Path field in the Workgroup tab of the

Application Properties dialog. Locking is performed during development only if Activate Team Development has been set to Yes.

The following table defines the locks issued.

Object	Action	Lock
Models	Enter repository if not checked out	Share
		Exclusive
	Enter repository if checked out	Exclusive
	Check in	
Tables	Zoom into Table, Columns, or Indexes	Share
	Properties	Share
	APG on file	Exclusive
	Check in	
Program Repository	Adding, Moving, and Removing programs	Exclusive
Programs	Zoom	Share
	Execute	Share
	Check in	Exclusive
Helps	Zoom into Help	Share
	Check in	Exclusive
Rights	Enter repository if not checked out	Share
		Exclusive
	Enter repository if checked out	Exclusive
	Check in	
Menus	Enter repository if not checked out	Share
		Exclusive
	Enter repository if checked out	Exclusive
	Check in	
Application Properties	Check in	Exclusive

Note: When an object is locked in share mode, all actions that need exclusive access to the locked object are blocked.

The Synchronization Process

The most important aspect of Team Development is synchronizing work among the developers. Synchronization is divided into two parts:

- Detecting if a change to the application occurred, and whether the change affects anything.
- Synchronizing the changes into the development environment for each of the users.

Detection of Modifications

In order to detect modifications, each toolkit has counters that determine the current revision of the objects. Another copy of these counters is placed in the station lock file. Whenever an object is checked in, the counter of the object is incremented in the station lock file.

When a synchronization event occurs, the toolkit compares the internal counters with the counters in the station lock file.

Synchronizing Modifications

When the Team Development feature detects that modifications have been made, a synchronization process begins. This process involves reading the modified objects into the development environment.

After loading, all open windows that are related to any version control object are re-displayed with the updated object. If the cursor was positioned on an entry that was removed, the cursor parks on the top of the object's repository. A message indicating the modification is displayed.

Synchronization Timing

The following table defines the timing of the synchronization events.

Object	Action
Models	Access Model repository Display Model Selection repository

Object	Action
Tables	Access Table repository Display Table Selection repository Zoom into Table Columns Zoom into Table Indexes
Program repository	Access Program repository
Programs	Access Program repository Display Program Selection repository Execute program Edit or browse program
Helps	Access Help repository Display Help Selection repository Edit or browse help
Menus	Access Menu repository
Components	Access Component repository

Application Access and Share Modes

The following table defines the Access and Share modes for accessing the application file.

Development / Deployment	Team Development	Access	Share
Development	On	Write	Write
Development	Off	Write	None
Deployment	N/A	Read	Read

Station Lock File

The application station lock is called `xxSTTION.LOC`, where `xx` is replaced by the application prefix. The station lock file resides in a directory defined by the MVCS lock file path field on the Workgroup tab in the Application Properties dialog. If that field is left blank, the file will be opened in the current working directory.

When you open an application, your User ID is entered into the station lock file. No other user using the same User ID will be allowed to open the application. Note also that the same user cannot open two instances of the application while using a single User ID. If for any reason the user attempts to open the application but has not logged on, the message "MVCS is on - cannot open application without user logon" will be displayed.

The station lock file is created upon the first logon to the application and is removed after the last logoff.

To define the eDeveloper environment when working with SQL databases, you must know how to define the flags and settings in the Magic.ini file.

In this chapter:

- | |
|--|
| • Configure and Define the eDeveloper Environment |
| • Naming Conventions - eDeveloper Gateways |
| • eDeveloper and SQL Configuration and Performance |
| • eDeveloper Database Gateway for Oracle |
| • MSSQL Server Database Gateway |
| • Informix Database Gateway |
| • DB2 Database Gateway |
| • ODBC Database Gateway |
| • ODBC Check Driver Utility |

Configure and Define the eDeveloper Environment

The eDeveloper Database Gateway for any SQL database must be defined and installed under both the eDeveloper Client/Server architecture and the RDBMS Client/Server architecture. It is necessary to define for eDeveloper that a specific gateway must be loaded by pointing to a variable that contains a DB number. The DB number points to a specific executable that is the relevant gateway.

Windows Operation Systems

In Windows platforms, the MGDBnn is set to point to the relevant DLL in the [MAGIC_ GATEWAYS] section of the MAGIC. INI file. For example, the Oracle id in the DBMS section is 13:

```
MGDB13= mgora8.dll
```

If you used the eDeveloper client server, MGDB13 should refer to .dll. An example of the gateway section in the MAGIC. INI file under Windows is:

```
[MAGIC_ GATEWAYS]
;MGCOMM01=mgwsock.dll
MGDB00=MGBTRV.dll
;MGDB03=mdcisam.dll
MGDB13=mgora8.dll
MGDB14=mginf73.dll
;MGDB16=gateways/mgeac32.dll
;MGDB18=mddb2.dll
;MGDB19=mgodbc.dll
MGDB20=mgms7.dll
MGDB21=mgmemory.dll
```

Unix Operating Systems

In Unix operating systems an environment variable points to the executable, which should be used for a specific gateway.

For example, in UNIX:

```
MAGIC_ DB_ 14_ DRIVER=$ MAGIC_ HOME/ bin/ mgoracle8
```

where the number 14 refers to the DB number +1.

Naming Conventions - eDeveloper Gateways

The actual gateway image will vary from one operating system to another. Therefore the names of the images are structured to distinguish among them.

In the Windows operating systems, the structure is **MGyyyyyy** where **yyyyyy** stands for the specific RDBMS, such as INF or ODBC.

In the UNIX operating system, the structure is **MGyyyyyy** where **yyyyyy** stands for the specific RDBMS, such as Informix or Oracle.

Gateway Name Structure

The following is the structure of the eDeveloper Gateway version shown in the Help/About information:

- The first string will contain the RDBMS name.
- The next string will be the word `Version`.
- The third string will be the eDeveloper version number followed by a "-" character.
- The fourth string will be the source id which will be in the format of X.Y, with X representing the major source version, such as 9 or 8, and Y representing the minor source version, or its running number, which is changed every time the source is changed.
- The last string will be the date on which the gateway was created.

For example:

```
Oracle 8 Version 9.00 - 9.1 08-Jan-2001
```

Oracle 8 is the RDBMS name and version, followed by the word Version, followed by the eDeveloper version number 9.00, then the source id 9.1 and finally the creation date of the gateway 08-Jan-2001.

eDeveloper's API Implementation and Versions

eDeveloper is a dynamic tool with no compilation or link stages. eDeveloper programs can be executed as soon as an operation is inserted into the proper eDeveloper table. The databases defined in eDeveloper are also dynamic, and can change from one program to another and from one application to another. eDeveloper does not limit the variety of dataview.

To provide this versatility with a single rule-based engine, eDeveloper uses dynamic SQL to construct the SQL dataview that eDeveloper programs require. Each eDeveloper task that uses SQL tables issues the proper SQL statement to specify the dataview needed. The SQL statement is then prepared by the eDeveloper Database Gateway. During this preparation stage the DBMS optimizer analyzes the statement and prepares the statement for processing.

Unlike 3GLs and 4GLs that use embedded SQL to eliminate runtime SQL statement parsing and optimization by pre-compilation, eDeveloper does not use compilation. Therefore, the preparation stage may produce additional overhead by executing a task that is repeatedly called. However, if the task is declared resident, the SQL statement is prepared just once and is not released. Repeated calls to the task will use the existing statement instead of preparing a new one. In this case, the performance of eDeveloper's dynamic SQL implementation is the same as the performance of an embedded SQL program.

Oracle

The Oracle Database Gateway is written with the Oracle Call Interface (OCI) procedures to achieve maximum capability and performance. It is designed to work with Oracle version 7.3 and above. Oracle subtitles need to be aligned the same way as the subtitles in Informix and DB2.

MS-SQL

The MS-SQL Database Gateway is written with the Object Link Embedding DB interface (OLE-DB). For maximum performance it uses client cursors and commands. It is designed to work with the Microsoft SQL server Versions 7 and above. MS-SQL subtitles need to be aligned the same way as the subtitles in Informix and DB2.

ODBC

The Database Gateway is written using the ODBC 2.00 API. It is designed to work with ODBC drivers version 2 and above. ODBC subtitles need to be aligned the same way as the subtitles in Informix and DB2.

Informix

The Informix Database Gateway is written using /C API. It is designed to work with Informix version 7.3 and above.

DB2

The DB2 Database Gateway is written using the DB2 Call Level Interface () procedures. It is designed to work with DB2 version 5 and above.

Pervasive SQL.2000

eDeveloper supports both the Pervasive SQL.2000 ISAM engine through eDeveloper's Btrieve gateway, and the SQL engine through the eDeveloper Pervasive gateway.

Data Definition Rules

Characteristic	MSSQL	Oracle	Informix	DB2
Longest object name (user, column, table, view, index)	128	30	18	18
Maximum length of char/ binary	8000	2000	255	254
Maximum length of	8000	4000	32767	4000
Most columns in a table, view	1024	1000		255
Maximum length of row (row length)	8060	32511	32700	4005
Maximum index length	900	255	255	255
Maximum size of each BLOB field	2GB	2GB		
Maximum number of segments per index	16	32	16	16
Maximum number of memo fields supported		1 ¹		

1 eDeveloper's Table Checker cannot distinguish between Oracle LONG and LONG RAW and eDeveloper Memo fields. If eDeveloper finds more than one field of any of these types, it will return the following error message: Database supports one memo field in record.

Configuration and Performance

Now that you understand how eDeveloper & SQL work, it is important to learn how to work with eDeveloper & SQL efficiently. Various concepts and techniques for achieving optimal eDeveloper & SQL performance are discussed below.

Transactions

A transaction is a sequence of one or more SQL statements that are usually closely related but perform interdependent actions and which form a logical unit of work. Each statement in the transaction performs some part of a task, but all of the statements are required to complete the task. The DBMS executes the sequence as one operation because the statements are grouped as a single unit. All the statements must be completed for the database to remain consistent.

When applications update multiple tables in a database, transactions ensure database integrity.

The Transaction Mechanism

If the transaction does not complete successfully, a Rollback statement returns the data to the state it was in prior to the beginning of the transaction. If the transaction completes successfully, a COMMIT statement permanently stores the data in the database.

Due to the nature of SQL and relational database architecture, where each update can act on only one table at a time and where there are no trailers in the usual sense, two update statements must be executed simultaneously. Both update statements must either succeed and COMMIT or fail and Rollback.

In a transaction a group of updates must either COMMIT or Rollback together. First a transaction is declared, and then either the COMMIT or the Rollback SQL statement is issued for all the statements that have been issued since the transaction declaration.

Locking

eDeveloper V9 introduces the concept of Physical and Deferred transaction modes. The following is the default behavior for the eDeveloper SQL gateways in each of the transaction modes:

- Physical transaction mode

All SQL gateways (except for ODBC) use physical locking as their default locking strategy.

ODBC always use logical locking, regardless of the transaction mode.

- Deferred transaction mode

All SQL gateways use logical locking as their locking strategy.

Records are locked to preserve data integrity and to give each user a consistent view, while providing maximum concurrency. Locking prevents a record or group of records from being changed while a user is viewing or modifying them. Locks can either be exclusive, not allowing other users to even read the records, or shared, letting other users read the records without modifying them in any way.

Transactions and locking are tightly bound in RDBMSs. Because SQL databases run in multi-user environments, indiscriminate locking can cause an application to severely limit user access. Different levels of locking are available in all the RDBMSs.

A record is locked for updating and then immediately released when we leave it. In an RDBMS, the lock is enforced at the beginning of the transaction and released only by a COMMIT or Rollback operation.

Locking Levels

During a normal operation, the RDBMS locks the view structures. Implicit locking is performed automatically and protects the data without any user intervention. Overriding default locking is known as explicit locking.

Implicit locking occurs automatically when SQL statements are executed. For example, the statements INSERT, UPDATE, and DELETE cause implicit locking so that data consistency and integrity are maintained during transactions.

Some , such as Oracle and Informix, acquire locks at a record level. Other RDBMSs acquire only page-level locks, which cause other records belonging to the same page to also be locked.

Escalating a Lock to a Table Lock

In some database systems lock escalation occurs because when many locks are held at one level, the RDBMS automatically changes these locks to a different lock at a higher level (such as a table lock). The number of locks is therefore reduced, but the granularity of what is locked is increased.

Enforcing Locks

Explicit locking can be acquired at the same levels as implicit locking. When updating a record, an implicit exclusive lock is acquired at record level.

When initiating a

```
select * from table where ... for update
```

statement under Oracle, or a

```
select * from table (UPDLOCK NOWAIT) where ...
```

statement under MSSQL, an explicit shared lock is acquired at record or page level, and the record cannot be updated until the lock is released.

Note that while using Informix, one cannot explicitly lock records when using the ORDER BY clause.

Lock Duration

A data lock is acquired at some time during the lifetime of a transaction and is withheld until a COMMIT or Rollback command is initiated. Locks are not released during a transaction.

Example:

```
begin transaction;  
update table set fld1= 1 where fld2= 2  
update table set fld1= 3 where fld2= 4
```

both records are locked with an exclusive lock

```
commit;
```

both records are released.

Locking and Transaction Processing

Locking and transaction processing are essential in multi-user environments. To achieve maximum concurrency, online transactions must be short and must not interfere with user interaction.

Transactions are opened according to the eDeveloper's Task properties settings. The following affect the way eDeveloper opens transactions and enforces locking include:

- Environment settings for multi-user and ISAM Transaction. In the system's Environment settings, both the Multi-user access setting and the ISAM Transactions setting must be set to Yes. Otherwise, eDeveloper will not issue any transaction that handles requests.
- Database settings for table locking.
- Access and share mode in the DB Table repository.
- Task type specified in the Task Properties dialog: Online, Batch or Browser.
- Locking strategy options specified in the Task Properties dialog.
- Transaction mode option specified in the Task Properties dialog.
- Transaction Begin option specified in the Task Properties dialog.

- The Lock property specified in the Call Properties dialog, for Call Task and Call Program operations.

The Multi-User Access Setting

The Multi-user access setting, located in the Multi User tab of the Environment dialog, specifies that when an eDeveloper lock is performed, the lock is also performed inside the underlying database.

When the Multi-user access setting is set to Yes, eDeveloper also sends the necessary SET TRANSACTION commands to the DBMS to ensure data integrity with other running applications.

When the Multi-user access setting is set to No, the application cannot be used simultaneously by more than one eDeveloper user. Therefore, no locks are performed by eDeveloper, and no physical or logical locking is performed by the gateway (that is, no SELECT ... FOR UPDATE). However, other DB users can change the data directly through the DBMS tool.

Table Access and Share Mode

In eDeveloper, you open a table with both an access mode and a share mode as specified in the task's DB Table repository (CTRL+D). When opening a table, eDeveloper performs some internal tasks, such as getting the structure of the table from the database, checking for the table's existence, and more. If eDeveloper locks are used, relevant information is written in the mglock file.

When working with RDBMSs, there is no Open File or Open Table command. Therefore, the access and share modes have almost no meaning except for the following cases:

- Share is None - The SQL gateway sends a Lock Table command to the SQL database. This is done only Oracle and Informix.
- If when a lock is requested the access modes is READ, the share mode is WRITE, and the table is a linked table, the records of the linked tables will not be locked. If the access is WRITE, eDeveloper assumes that the linked table may also be updated, and a lock is issued for it as well.

Physical and Logical Locks

There are two major levels of locks in SQL:

- An exclusive lock, which is automatically generated when an UPDATE statement is issued.
- A shared lock, which can be issued by a SELECT statement or by adding the FOR UPDATE clause at the end of a SELECT statement. A shared lock tells the SQL server that you plan to update the record, and that no updates will be allowed until this lock is released. If another user tries to update the record or to send a SELECT ... FOR UPDATE statement, an error message that the record has been locked by another user is sent to that user.

Physical Locks

The SELECT ... FOR UPDATE statement is available in Oracle, MSSQL 7, Informix, and DB2. These RDBMSs support row-level locking. Therefore, when a lock is requested according to the selected locking strategy, eDeveloper tells the gateway that the record should be read again with a lock. If a transaction has not been started, the gateway starts a transaction. Then the gateway reads the current record with the FOR UPDATE clause (except in MSSQL 7 where it uses the UPDLOCK hint instead).

The procedure described above ensures that no application, including non-eDeveloper applications, will be able to update the record until the end of the transaction, which usually occurs after the update is done.

Example:

The current record is retrieved:

```
SELECT empnum, ename, deptno, rowid
FROM emp
WHERE rowid= 1111
```

Returned values: 1 , John, 30

The lock is requested:

```
SELECT empnum, ename, deptno, rowid
FROM emp
WHERE empnum= 1111
FOR UPDATE NO WAIT
```

Returned values: 1 , John, 30

Assume that the deptno was changed to 40:

```
UPDATE emp SET deptno= 40
WHERE rowid= 1111
```

In batch tasks, when an Immediate locking strategy is used the gateway may be able to lock the whole dataview, ahead of time.

In Oracle, a SELECT statement can be issued with a FOR UPDATE clause and an ORDER BY clause. Therefore, when the cursor is defined at the beginning of the task, it is declared with a FOR UPDATE clause. Then all the records are fetched from this cursor.

Informix and DB2 do not allow the procedure used by Oracle. Instead, the cursor is opened, and another cursor is opened with a FOR UPDATE clause for every record.

For additional information see the reference material on Operations, in the Link Join section.

Logical Locks

MSSQL 7 supports both logical and physical locks. To work with logical locks, the flag SQL_PHYSICAL_LOCKING=N should be specified in the Database Information field in the Database Properties dialog.

ODBC supports only page level locking, which may result in locking problems. Therefore a logical lock strategy is used in which the record is not actually locked. Instead, eDeveloper verifies, for integrity reasons, that no one changes the record from the moment the record was logically locked until the update.

When a lock is requested and eDeveloper asks the gateway to read the record with a lock, the gateway reads the record and keeps the values of the read record.

When the UPDATE statement is then issued in the record suffix, all of the columns are added to the WHERE clause.

If in that period of time the record, which was not locked, has been changed by another user, the gateway sends a message that the record has been changed by another user, and the UPDATE fails.

Example:

The current record is retrieved:

```
SELECT empnum, ename, deptno
FROM emp
WHERE empnum= 1
values: 1 , John, 30
```


Lock is requested:

```
SELECT empnum, ename, deptno
FROM emp
WHERE empnum= 1
values: 1 , John, 30
```

Assume that the deptno was changed to 40:

```
UPDATE emp SET deptno= 40
WHERE
empnum= 1 AND ename= John' AND deptno= 30
```

The ODBC gateway also uses logical locking behavior because it cannot assume that a record lock or a FOR UPDATE statement is available in the accessed database.

Null Value

Nulls represent missing and unknown data. All SQL databases support null values. A field that has a null value is different from a blank field in an ISAM file. Null means that the value is not known. Null values require special handling. If you attempt to do arithmetical operations on a numeric column and one or more of the values are null, then the result will be null. If an alpha field allows null values, and you select all records in which the alpha field is blank, records with the null value in the alpha field will NOT be selected.

Null values do not participate in index searches. It is highly recommended not to define indexes on columns that are null-allowed.

For example, this SELECT statement

```
SELECT *
FROM Table1
WHERE Fld1>=4 or Fld1<4
```

will return all the records in Table1 except for the records where Fld1 is null.

Nulls are represented in a different sort value in each database. For example, if we perform this SELECT statement in Oracle and in MSSQL, we'll receive a different order of records in each database.

```
SELECT *  
FROM Table1  
Order by Fld1 ASC
```

In Oracle nulls are saved as the highest value in the database, so records with nulls in column Fld1 will appear as the last records of this SELECT statement.

In MSSQL nulls they are saved as the lowest value in the database, so records with nulls in column Fld1 will appear as the first records of this SELECT statement.

Index Definition and Usage

For best response time RDBMS indexes should be used for most data retrieval. Usually the RDBMS uses one of the indexes when the SQL statement has a WHERE clause on one or more first segments of that index and the requested record order is consistent with that index.

Each of the following examples illustrates which SQL statements use the indexes and when. For these examples, assume there is an index IN1 on fields F1, F3, F5 of table TBL1, and another index IN2 on F3, F4, F5.

eDeveloper issues this next statement when using the first key and ranging on F1 with the same expression for FROM and TO, and on F3 with two different expressions.

Example:

```
SELECT F1, F2, F3, F4, F5  
FROM TBL1  
WHERE F1= 1  
AND F3=> 100  
AND F3=< 200  
ORDER BY F1, F3, F5
```

Index IN1 will be used for the range on F1 and F3. The order will be achieved automatically by using the index.

eDeveloper issues the next statement when using the first key and ranging on F1 with the same expression for FROM and TO, and on field F2 with two different expressions.

Example:

```
SELECT F1, F2, F3, F4, F5
FROM TBL1
WHERE F1= 1
AND F2<= 'x'
AND F2>= 'c'
ORDER BY F1, F3, F5
```

Index IN1 will be used for the range on F1. The RDBMS searches all records with F1= 1. Only those with F2 between c and x will be in the result table. The order will be achieved automatically by using the index.

eDeveloper issues this next statement when using the first key and ranging on F1 with a different expression for FROM and TO, and on field F3 with two different expressions.

Example:

```
SELECT F1, F2, F3, F4, F5
FROM tbl1
WHERE F1>= 1
AND F1 <= 10
AND F3>= 100
AND F3<= 200
ORDER BY F1, F3, F5
```

Index IN1 will be used for the range on F1. The RDBMS searches all records with F1 between 1 and 10 and compares them to the range of F3 values. The range on F3 is not done by using the index because the range of the previous segment was not on a single value. The order will be achieved automatically by using the index.

eDeveloper issues this next statement when ranging on F1 with a single expression and on F3 with two different expressions and using the second key.

Example:

```
SELECT F1, F2, F3, F4, F5
FROM TBL1
WHERE F1= 1
AND F3>= 100
AND F3<= 200
ORDER BY F3, F4, F5
```

Index IN1 will be used for the range on F1 and F3. The RDBMS orders query results by sorting the result table. The second index cannot be used to supply the requested order because the first index is used for range. Using sort is relatively fast, as only the result table will be sorted and in most cases it will be relatively small.

Note: Indexes take up space in the database and are time-consuming when inserting, updating, and deleting records from the database. In some extreme cases indexes can cause poor performance for SELECT statements. For example, when a table's data consists of less than a block of disk space, which can be several thousand records for a normal table, a SELECT statement does not perform well. When accessing the table through an index the RDBMS actually executes two I/Os; one for the index and one for the data. Executing a full table scan on a single block that was read into memory in a single I/O is faster because memory access is always faster than disk I/O. When all of the columns selected in the query are in the index, the RDBMS will then access only the Index and not the data - this is called a cover index query.

However, in most cases, indexes will enhance the speed of your application.

Range Definition

When browsing large tables in relational databases it is important to use ranges to reduce the number of records in the view. When a table is accessed without ranges, such as in the APG, some of the operations can take a long time, especially operations such as locate and page up. To improve performance the ranges should be on segments of an index.

eDeveloper lets you specify ranges from four places:

- eDeveloper SELECT statement - All ranges mentioned in the eDeveloper SELECT statement become part of the SELECT statement. The range will then be handled by the RDBMS with a WHERE clause, even during a sequential search. eDeveloper receives only the records that answer the query.
- Range expression at the task level - When using a range expression at task level, eDeveloper checks each record returned from the database against the expression, and decides if the record is part of the view. eDeveloper receives all the records and performs the filtering on its own.
- DB SQL Range – free text that will be concatenated to the WHERE clause sent to the database as is. This is done for Physical transaction mode tasks only. There is a need to know the specific database syntax.
- eDeveloper SQL Range – an eDeveloper expression that eDeveloper translates to the appropriate syntax per database. This is added to the WHERE clause sent to the database. There is no need for the specific database syntax.

These four options perform very differently. It is better to put as much of the range information as possible in the range that can be sent to the database. Only enter ranges that cannot be expressed otherwise at the task Range level.

Supported eDeveloper functions for the SQL Where expression

Character functions	Len, Lower, Upper, LTrim, RTrim, Trim, MID, InStr, &, LIKE, IN
Date functions	Date, Time, AddDate, AddTime, Year, Month, Day, Hour, Minute, Second, Str, Val
Arithmetic operators	+, -, *, /
Logical functions	OR, AND, NOT
Comparison operators	=, <, <=, >, >=, <>
Null functions	NULL, ISNULL
Arithmetic functions	ABS, MOD, LOG, EXP, Round
Trigonometric functions	SIN, ASIN, COS, ACOS, TAN, ATAN
Convert functions	ASC, CHR, DStr, TStr

For more information about ranges, refer to the Range/Locate section in Chapter 6, Programs.

Sorting

Internal sorting in eDeveloper is very costly. It is always best to ask the RDBMS to sort the data and to supply the records in the required order to eDeveloper. This is done by using the Sort Using RDBMS feature. For more information, refer to the Sort Repository section in Chapter 6, Programs.

Stored Procedures

Stored procedures provide a very powerful way to move parts of the application logic to the server. They can be called from within the eDeveloper environment by using the Direct SQL feature.

The syntax of stored procedures differs by database, so using this feature may conflict with the application's portability requirements. For more information see the Direct SQL Command section in Chapter 6, Programs.

Reducing Network Traffic

Reducing network traffic is the best way to improve performance. Follow these guidelines to reduce traffic on the network:

- Select only the fields you really need from the record. Only those fields will be received from the database.
- Use ranges that can be sent to the database instead of a Task Range.
- Use Link Inner Join/Left Outer Join instead of Link Validate/Query.
- Use Database Views instead of links, when possible.

Using RDBMS views instead of simple tables can improve performance when joining tables. eDeveloper's link operation is implemented by issuing a separate SELECT statement to the database for every linked table. Batch tasks can be performed more efficiently by using a view that joins the main file and the linked files or by using eDeveloper's Link Inner Join/Left Outer Join operation instead of Link Validate/Query. Let the database do the join.

Note the differences in how eDeveloper and RDBMSs use the term *view*:

- eDeveloper uses the term *view* to describe the fields selected from the main file together with fields selected from the linked files and virtual fields calculated by the real fields.
- SQL databases use the term *view* to describe a predefined SELECT statement saved in the internal Data Dictionary and used as a table. The view definition can join several tables, use statistical functions, and use a WHERE clause to select part of the records in the table.

Example:

There is a task with a main file containing 10,000 records and three linked files. If performed in the usual way (using Link Query/Validate), the number of SELECT cursors opened by the task would be: $1 + 3 * 10,000 = 30,001$.

If you use eDeveloper's Link Inner Join/Left Outer Join operations instead of Link Validate/Query, the number of SELECT statements executed by the task would be: 1.

If you define a DB view in the database, the number of SELECT statements executed by the task would be: 1.

Note that updates on views is sometimes restricted by database limitations.

Incremental Locate

When you use Incremental Locate, every time you type a character eDeveloper re-issues all the SELECT statements of the task.

It is advisable to avoid the use of the Incremental Locate operation in your applications.

For more information see the Update operations in Chapter 7, Operations.

Direct SQL

Direct SQL can provide better performance by using RDBMS features that are not used in normal eDeveloper programming.

Global update to table rows and global delete of table rows are good examples. It is especially important to use global statements when working in a Client/Server environment to avoid excessive network traffic.

Another example is using statistical functions such SUM, MAX, or AVG. If you want a record count, it is more efficient to use SELECT COUNT(*) in a Direct SQL task than to read records just to count them.

String Time Attribute Mapping

eDeveloper maps an eDeveloper String Time column to CHAR in the database. If you want to map eDeveloper's String Time to a Date data type in the database (Date in Oracle, DateTime in MSSQL, etc.), specify MTime as the Type property under the column properties/SQL.

Properties Supported by Various Gateways

Level	Property	Type	Default	Applicable Gateways
DBMS	Log Level	None, Developer, Support, Customer	None	All
	Log Name	Alpha 255	N/A	All
	Log Sync	Yes/No	No	All
	Show Plan	Yes/No	No	MSSQL DB2
	Maximum Connections	Number	3	MSSQL
	Isolation Level	Number	MSSQL= 0 DB2 = 1 Informix = 1	MSSQL DB2 Informix
	Check Existence	Yes/No	No	All
	Check Existence	Yes/No	No	All
	Hint	Alpha 255	N/A	Oracle MSSQL
	Array size	Number	0	Oracle MSSQL DB2
Database				

Level	Property	Type	Default	Applicable Gateways
Table	Connect string	Alpha 255	N/A	Oracle
	Owner	Alpha	N/A	All
	Position	Default/ Unique Index/ RowID	Default	All
	Index	Unique Index Number	None	All
	Check Existence	Yes/No/As Database	As Database	All
	Table Type	Table/View	Table	All
	Hint flag	Yes/No	Yes	Oracle MSSQL
	Hint	Alpha 255	N/A	Oracle MSSQL
	Cursor	Default/Yes/No	Default	MSSQL
	Array size	Numeric 4	N/A	Oracle MSSQL
Index	Clustered	Yes/No	No	DB2
				MSSQL
				Informix
	Hint	Alpha 255	N/A	Oracle
				MSSQL
				Informix
	Drop during re-index	Yes/No	No	All
	Owner	Alpha 255	N/A	Informix

Level	Property	Type	Default	Applicable Gateways
Column	Type	Alpha 255	N/A	All
	User Type	Alpha 255	N/A	MSSQL
				DB2

The eDeveloper Database Gateway for Oracle

eDeveloper Data Types

The following table lists eDeveloper attributes with the supported storage types and the valid data types in Oracle. Each entry in the table has a default data type that can be forced by specifying that type in the Type property on the SQL tab in the Column properties dialog.

Note: n is the specified picture number.

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	Oracle Data Type
Alpha	String	n,1-4000	n,1-4000	VARCHAR2
		n, 4001-32700	n, 4001-32700	LONG
	Lstring	n, 1-255	n+1, 2-256	VARCHAR2
	Zstring	n, 1-4000	n+1, 2-4001	VARCHAR2
		n, 4001-32700	n+1, 4002-32701	LONG
Numeric	Signed Integer	n,1-4	2	NUMBER
		n, 5-9	4	NUMBER
	Unsigned Integer	n, 1-2	1	NUMBER

eDeveloper per Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	Oracle Data Type
		n, 3-4	2	NUMBER
		n, 5-9	4	NUMBER
	Float	n, 1-7	4	NUMBER
		n, 8-16	8	NUMBER
	Float MS- Basic	n, 1-7	4	RAW
		n, 8-16	8	RAW
	Float Decimal	n, 1-7	4	RAW
		n, 8-16	8	RAW
	Packed Decimal	n, 1-10	2*n-1	RAW
	Numeric	n, 1-18	n	RAW
	Character Number	n, 1-18	n-1	RAW
	String Number	n, 1-19 ¹	n+1	NUMBER
	eDeveloper Number	n, 1-18	n/2+1 ¹	RAW
	C-Isam Decimal	n, 1-32	n/2+1 ¹	RAW
Logical	Integer Logical	5	1	NUMBER
		5	2	RAW
	String Logical	5	1	RAW
Date	String Date	###/###/ ####	8	Date ²

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	Oracle Data Type
Time	Integer Date	##/##/ ####	4	NUMBER
	Integer Date 1901	##/##/ ####	4	NUMBER
	YYMMDD Date	##/##/ ####	4	RAW
	eDeveloper Date	##/##/ ####	4	RAW
	eDeveloper Date 1901	##/##/ ####	4	RAW
	String Time	HH:MM:SS	6	CHAR ³
	Integer Time	HH:MM:SS	4	NUMBER
	HMSH Time	HH:MM:SS	4	NUMBER
	eDeveloper Time	HH:MM:SS	4	RAW
	Memo			
	String Memo	n, 1-1998	n+2	RAW
	eDeveloper Memo	n, 1999 and above	n+2	LONG RAW
BLOB	Binary Large Object		12 (default)	LONG RAW

¹ - Note that eDeveloper's number support is up to 18 digits.

² - If you want to map eDeveloper String Date to Oracle character, specify 'CHAR(8)' as the Type under the column properties/SQL.

³ - If you want to map eDeveloper's String Time to Oracle Date data type, specify 'MGTime' as the Type property under the column properties/SQL.

Blob Mapping Flag

The LONGRAW data type is the eDeveloper BLOB field default for an Oracle database. There are restrictions from Oracle when using LONGRAW. For example, only one LONGRAW field can be specified in a record and retrieval is limited to only this kind of data type value.

You can overcome these restrictions by activating the **Default_Blob_to_Blob** flag to change the BLOB field default value to instruct the gateway to use different mapping without specifying an SQLTYPE. Default_Blob_to_Blob can be specified for Oracle database entries in the Database Information field under the SQL tab.

When the flag is set to Yes, the default mapping in the Oracle gateway for an eDeveloper BLOB will be an Oracle BLOB data type instead of LONG RAW. When the flag is set to No, the default mapping will be LONG RAW.

Oracle Data Types

The following table shows the results of an eDeveloper Get Definition operation from an Oracle Table, and shows eDeveloper equivalents for Oracle database data types.

Oracle Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
CHAR(n) VARCHAR2(n)	Alpha	Zstring	n+1,2-4001	n,1-4000
LONG ¹	Alpha	Zstring	Default (1)	Default (0)
LONG RAW ¹	BLOB	Binary Large Object	Default (12)	
RAW (n)	Alpha	String	n, 1-2000	n, 1-2000

Oracle Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
NUMBER	Numeric	Float	8	4.3
NUMBER (p,s)	Numeric	Float	8	p-s,s
NUMBER (p)	Numeric	Float	8	p
Date ²	Date	String Date	8	DD/MM/YY YY YY
ROWID	Alpha	Zstring	19	18

¹ -You must set the Picture for LONG and LONG RAW data types columns to the appropriate size for your application after getting the table's definition.

² - By default, Oracle Date data type is mapped to eDeveloper date storage type. If you want to be able to see all parts of a Date column in the format 'YYYY/MM/DD HH:MM:SS.mmm', you should map the Oracle Date data type to eDeveloper's Alpha attribute. This can be done by specifying 'SQL_DateTOALPHA= Y' in the Database Information field in the Database Properties dialog (see the Database Information section in this chapter for more information).

Long and Long RAW Data Types

After performing a Get Definition on an Oracle table, the user must enter the appropriate picture that contains a column of type LONG or LONG RAW. By default, the field that receives the column will have a picture of 0. This picture is invalid, and the user must enter a valid picture between 1 and 32000.

Hints

Using a Hint String, the developer can specify a hard-coded string that can be added to the Select statement as is, without being checked. The syntax of a Hint is :

```
/*+ Oracle Hint */.
```

It is recommended that you use the Optimizer Hints in special cases only. A full list of the Optimizer Hints that can be sent to Oracle can be found in the Oracle documentation.

Database Information

The Database Information parameter (in the Database properties, Table properties, Column properties, and Index properties) lets you supply database-dependent information that eDeveloper can pass to the underlying DBMS. The use of this parameter is optional. The values that can be sent in the Database Information parameter are:

Database Properties

- SQL_ DateTOALPHA

The SQL_ DateTOALPHA setting automatically converts the RDBMS date field to an eDeveloper alpha Zstring field with a length of 19 characters and with the format YYYY- MM- DD HH: MM: SS.

Note: Neither eDeveloper nor the RDBMS can perform data validation on an Alpha Date field due to the use of the internal date format in the RDBMS. If you use the SQL_ DateTOALPHA parameter you should implement your own validity checks for Insert and Update operations. Otherwise, invalid dates can be inserted in the database.

- NLSSORT= Y

The NLSSORT setting lets the application match character strings that follow alphabetic conventions. For more information, refer to the NLSSORT Support section in this chapter.

Table Properties

- TABLESPACE=...
- INITTRANS=...
- MAXTRANS=...

- PCTFREE=...
- PCTUSED=...
- CLUSTER=...
- STORAGE=...

It is possible to specify any of the above parameters in the Table Properties/ Database Information field when you are creating a table in Oracle.

Table Locking

Table locking is available only within an eDeveloper transaction. eDeveloper will ignore the request if it is issued outside of a transaction. Therefore, Share None refers to an Exclusive table lock and Share Read refers to a Share table lock.

Physical Locking

A physical lock is a method that ensures that no one else can modify a record that is currently locked. Specifically, from the moment a user locks the record until that user releases the record, the record cannot be modified by another user. The physical lock is implemented as follows: When eDeveloper locks a record according to its locking strategy, eDeveloper issues a Select statement with a FOR UPDATE clause.

The FOR UPDATE clause prevents other applications from making changes to the record until the end of the transaction.

Views

A View must have a virtual unique index defined. Insert, Update, and Delete operations are allowed on Views, but only if that View has a ROWID. A View that is defined on more than one table does not have a ROWID. Therefore, the Position parameter on the SQL tab in the Table Properties dialog should be Unique Index, and not Default or ROWID.

Note: eDeveloper relates to View as a regular table. It is recommended that you not use eDeveloper to perform any type of rename or convert operations. If you do execute one of these operations, eDeveloper will display an error message and will not convert or rename the View in the database.

Unique Identifier

eDeveloper must have a unique row identifier for each table that it opens. Whenever possible, eDeveloper will use Oracle's ROWID as the unique identifier. If the dataview is built from fields from more than one Oracle database table, there will be no Oracle ROWID, and a unique index must be entered. This may be a virtual index.

NLSSORT Support

The NLSSORT Support feature lets the application match character strings that follow alphabetic conventions. Normally, character strings in a WHERE clause are compared by using the character's binary values.

Using the NLSSORT Support feature in the WHERE clause allows users to work on a foreign-language Oracle client while sorting data alphabetically in their own language.

To use this feature, add the following flag in the DatabaseProperties\SQL\ Database Information field in the Database repository:

```
NLSSORT= Y
```

This flag adds the NLSSORT function to all the WHERE clauses that eDeveloper sends to the database, as follows:

```
NLSSORT (value) comparison_ operator NLSSORT(column)
```

For example:

```
NLSSORT ( ' A ' ) =NLSSORT (FLD1)
```

Note that use of this flag may cause performance problems. Use of this flag also causes the eDeveloper Database Gateway for Oracle to be case insensitive.

Stored Procedures

Procedures without INOUT or OUT parameters must be called as a PL/ SQL block:

```
begin procname (par1 int, par2 int); end;
```

If the procedure accepts more than one parameter, separate the parameters with commas. If the Oracle parameters are inout or out, the APG button must be selected before executing the procedure. Do not enter an out parameter in a stored procedure. Instead, use a comma as a place-holder. Do not add any punctuation at the end of the procedure.

MSSQL Server Database Gateway

eDeveloper Data Types

The following table lists eDeveloper attributes with the supported storage types, and the valid data types in MSSQL. Each entry in the table has a default data type that can be forced by specifying that type in the Type property on the SQL tab in the Column properties dialog.

Note: n is the specified picture number.

eDevelop er Attribute	eDeveloper Storage Type	eDevelope r Picture	eDeveloper Storage Size	MSSQL Data Type
Alpha	String	n,1-8000	n,1-8000	CHAR
		n, 8001-32700	n, 8001-32700	Text
	Lstring	n, 1-255	n+1, 2-256	BINARY
	Zstring	n, 1-8000	n+1, 2-8001	CHAR
		n, 8001-32700	n+1, 8002-32701	Text

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	MSSQL Data Type
Numeric	Signed Integer	n, 1-4	2	SMALLINT
		n, 5-9	4	INTEGER
	Unsigned Integer	n, 1-2	1	BINARY
		n, 3-4	2	BINARY
		n, 5-9	4	BINARY
	Float	n, 1-7	4	REAL
		n, 8-16	8	DOUBLE PRECISION
	Float MS- Basic	n, 1-7	4	BINARY
		n, 8-16	8	BINARY
	Float Decimal	n, 1-7	4	BINARY
		n, 8-16	8	BINARY
	Packed Decimal	n, 1-10	2*n-1	BINARY
	Numeric	n, 1-18	n	BINARY
	Character Number	n, 1-18	n-1	BINARY
	String Number	n, 1-19 ¹	n+1	BINARY
	eDeveloper Number	n, 1-18	n/2+1 ¹	BINARY
	C-Isam Decimal	n, 1-32	n/2+1 ¹	BINARY

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	MSSQL Data Type
Logical	Integer Logical	5	1	BIT
		5	2	SMALLINT
	String Logical	5	1	BINARY
Date	String Date	##/##/ ####	8	DateTime 2
	Integer Date	##/##/ ####	4	INTEGER
	Integer Date 1901	##/##/ ####	4	INTEGER
	YYMMDD Date	##/##/ ####	4	BINARY
	eDeveloper Date	##/##/ ####	4	BINARY
	eDeveloper Date 1901	##/##/ ####	4	BINARY
Time	String Time	HH:MM:SS	6	CHAR 3
	Integer Time	HH:MM:SS	4	INTEGER
	HMSH Time	HH:MM:SS	4	BINARY
	eDeveloper Time	HH:MM:SS	4	BINARY
Memo	String Memo	n, 1-7998	n+2	VARBINAR Y
	eDeveloper Memo	n, 7999 and above	n+2	IMAGE
BLOB	Binary Large Object		12 (default)	IMAGE

¹ - Note that eDeveloper's number support is up to 18 digits.

² - If you want to map eDeveloper String Date to MSSQL character, specify 'CHAR(8)' as the Type under the column properties/SQL.

³ - If you want to map eDeveloper's String Time to MSSQL DateTime data type, specify 'MGTime' as the Type property under the column properties/SQL.

MS-SQL Data Types

The following table shows the results of an eDeveloper Get Definition operation from an MS-SQL Table. eDeveloper equivalents for Microsoft SQL server data types are shown.

SQL Server Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
CHAR(n), VARCHAR(n)	Alpha	Zstring	n+1, 2-8001	n, 1-8000
Text ¹	Alpha	Zstring	Default (11)	Default (10) 1
INTEGER	Numeric	Signed Integer	4	10
SMALLINT	Numeric	Signed Integer	2	5
TINYINT	Numeric	Unsigned Integer	1	3
NUMERIC(p, s)	Numeric	Float	8	p-s, s
DECIMAL(p, s)	Numeric	Float	8	p-s, s
DOUBLE PRECISION	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
FLOAT, REAL	Numeric	Float	4	5.2

SQL Server Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
MONEY	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
SMALLMONEY	Numeric	Float	4	5.2
DateTime ²	Date	String Date	8	##/##/####
SMALLDateTime ²	Date	String Date	8	##/##/####
BINARY(n), VARBINARY(n)	Alpha	String	n, 1- 8000	n, 1-8000
IMAGE	BLOB	Binary Large Object	Default (12)	
BIT	Logical	Integer Logical	1	5
TimeSTAMP	Alpha	String	8	8

¹ -You must set the Picture for Text data type columns to the appropriate size for your application after getting the table's definition.

² - By default, MSSQL Date data types (DateTime, SMALLDateTime) are mapped to eDeveloper date storage type. If you want to be able to see all parts of a DateTime/ SMALLDateTime column in the format 'YYYY/MM/DD HH:MM:SS.mmm' for DateTime, and 'YYYY/MM/DD HH:MM' for SMALLDateTime, you should map the MSSQL DateTime/ SMALLDateTime to eDeveloper's Alpha attribute. This can be done by specifying 'SQL_DateTOALPHA= Y' in the Database Information field in the Database Properties dialog (see the Database Information section in this chapter).

Text Data Type

The user must enter the appropriate picture, after performing a Get Definition on a MSSQL table, which contains a column of type Text. By default, the field that receives the column will have a picture of 10, and the user must enter a valid picture between 1 and 32000.

Physical Locking

A physical lock is a method that ensures that no one else can modify a record that is currently locked.

Specifically, from the moment a user locks the record until that user releases the record, the record cannot be modified by another user. The physical lock is implemented as follows: When eDeveloper locks a record according to its locking strategy, eDeveloper issues a Select statement with a UPDLOCK hint.

The UPDLOCK hint prevents other applications from making changes to the record until the end of the transaction.

Logical locking is also available by using the flag `SQL_PHYSICAL_LOCKING=N`. For more information, refer to the Database Information section in this chapter.

Hints

Using a Hint String, the developer can specify a hard-coded string that will be added to the Select statement as is, without being checked.

Specifying `FORCE_ INDEX` Hint in the Index Properties dialog applies to this index only. Specifying `FORCE_ INDEX` Hint in the Table Properties dialog applies to all the indexes in that table, and specifying `FORCE_ INDEX` Hint in the Database Properties dialog applies to all the indexes in all the tables in the database.

If you specify a `FORCE_ INDEX` Hint in the Table or Index Properties dialog, and you do not want the gateway to force the optimizer to use an index for a specific index, or for all indexes in a particular table, then you can set the Hint

to NO in the Table or Index Properties dialog, and the optimizer hint will not be issued.

FORCE_ INDEX Hint can cause the gateway to force the index in the following manner:

eDeveloper will add the following to the Select statement generated by the gateway:

```
" ( INDEX indname) "
```

where indame will be the name of the index for that particular table in eDeveloper.

It is recommended that you use the Optimizer Hints in special cases only. A full list of the optimizer hints that can be sent to MSSQL can be found in the MSSQL documentation.

Identity Column

Columns that have the IDENTITY property contain system-generated values that uniquely identify each row within a table. This is used to generate sequential numbers (for example, employee identification numbers). When inserting values into a table with an identity column, MSSQL automatically generates the next identifier based on the last used identity value (incremented by adding rows) and the increment value specified during column creation.

By default, data cannot be inserted directly into an identity column.

IDENTITY column is usually defined to be used as position for a table in eDeveloper; therefore it has to be selected in the eDeveloper task. This means that the INSERT command that will be issued by the task will contain this column. Since MSSQL does not allow updating this column's value manually, an error message should appear. The solution is to identify to eDeveloper that a column is an IDENTITY column, and to disregard this column in eDeveloper's Insert commands.

This was implemented in the eDeveloper MSSQL gateway, in the following way:

A column will be identified as an IDENTITY column by adding the keyword "IDENTITY" to the Type Property in the SQL Tab of the Column Properties, following the SQLType of the column. This will either be done by the eDeveloper Programmer (if the table is created from eDeveloper) or by the gateway when the programmer issues Get Definition. This will indicate to the gateway to handle this column as IDENTITY.

Example:

An INT column that has an IDENTITY property should be defined as:

```
INT IDENTITY
```

in the Type field in the Column Properties/SQL section.

An IDENTITY column will be skipped when the gateway issues an INSERT command.

In Get Definition, the default for IDENTITY columns will be "Non Modifiable".

Note: If the programmer changes the Modifiability of an IDENTITY column to Yes and updates it in a task, an error message will result.

Views

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted on Views. MS-SQL allows updates on multiple table views. If one of the segments of the Position Index is updated, records shown on the screen may become inconsistent. It is important to note that eDeveloper considers a View as a regular table. eDeveloper should not be used to perform any rename or convert operations. If eDeveloper is used to execute any rename or convert operations, eDeveloper will display an error message and will not convert/ rename the View in the database.

Temporary Tables

Temporary tables are tables that are dropped automatically by MSSQL when MSSQL has finished working with them. There are two kinds of temporary tables: Local and Global.

Local Temporary Tables

A local temporary table is visible only in the current session and is dropped automatically at the end of the current session. Its name is prefixed with a single number sign (#table_name).

Notes:

- Local temporary tables cannot be created in a sub-task. In order to create a local temporary table on the fly, you must define it in the DB table of the parent task. Another option is to work with global temporary tables.
- Local temporary tables cannot be viewed using Direct SQL. Use global temporary tables instead.
- Local temporary tables cannot be created when the MCF file is also stored in MSSQL. In order to do this, define two separate DB entries in eDeveloper - one for the MCF file and another for the data. Another option is to work with global temporary tables.

Global Temporary Tables

A global temporary table is visible to all sessions. Global temporary tables are automatically dropped when the session that created the table ends and all other tasks have stopped referencing them.

The association between a session and a table is maintained only for the life of a single Transact-SQL statement. This means that a global temporary table is dropped at the completion of the last Transact-SQL statement that was actively referencing the table when the creating session ended.

Global temporary table names are prefixed with a double number sign (##table_name).

Global and local temporary tables are automatically created in TempDB by MSSQL, regardless of from which database the CREATE command was issued.

Temporary tables are automatically dropped when they go out of scope, unless they have already been explicitly dropped using DROP TABLE.

The MSSQL gateway supports storing temporary tables in the TempDB database simply by adding the number sign ('#' or '###') as the prefix of the table name in the Table Repository.

Cursors and DB Commands

There are two ways to work with MSSQL – using Cursors or using DB Commands.

DB Commands

The gateway uses cursors by default. When processing a large number of records in an eDeveloper task, such as in a batch task that scans a file, it is possible to change the default setting by changing the Cursor parameter in the Table Properties dialog to No. This will cause the gateway to use DB commands instead of cursors, requiring separate connections for each result set. The maximum number of connections is user-defined, and the default setting for the maximum number of connections is 3.

The gateway can send only one DB command in the current connection (in addition to numerous cursors). If another DB command should be issued, a new connection will be opened. DB commands are used for commands that execute a single record or command, such as fetching a linked record, CREATE TABLE command, or a executing a Direct SQL task. Generally speaking, the greater the number of connections the gateway uses, the better the client performance. However, at the same time memory requirements increase while server performance decreases. If all of the existing connections are already used by a pending command and a new connection is needed, an existing connection will be freed for your use according to the LRU algorithm. This can affect performance adversely, because the released command will eventually be reissued.

If a Direct SQL batch task is used (Stored Procedure or Select statement), the gateway is unable to reuse this connection until all the results have been fetched. If nested direct SQL batch tasks are used, and the maximum number of connections the gateway can use is not enough, the following error message will appear:

MS-SQL Gateway: No more connections available.

If this happens, try increasing Max Connections in the DBMS properties. If this error message still appears, either increase the value of the parameter noted in the error message or modify your application, so that it does not use the nested Direct SQL tasks.

Cursors

The MSSQL gateway uses Dynamic cursors for Online tasks and KeySet cursors for Batch tasks.

- Relevant Parameters
 - Setting the Table Properties/SQL/Cursor parameter to No disables the use of cursors on the specific table and uses DB command instead.
 - Setting the Max Connections parameter in the DBMS Properties dialog controls the maximum number of connections the gateway can use. The default value is 3 (plus 1 extra) connections. The extra connection is used by the gateway for commands that fetch a single record, or for a command without fetchable results, such as Update. This number is for the Server/User-name pairs. If you have defined several databases in *Settings/Databases*, and some of them use different servers or different user names, this number applies to each database that has a different server or user name.

Database Information

The Database Information parameter in the Database Properties, Table Properties, Column Properties and Index Properties dialogs lets you supply database-dependent information that eDeveloper can pass to the underlying DBMS. The use of this parameter is optional. The values that can be sent in the Database Information are:

Database Properties

- SQL_ DateTOALPHA

The SQL_ DateTOALPHA setting automatically converts the RDBMS date field to an eDeveloper alpha Zstring field with a length of 19 characters and with the format YYYY- MM- DD HH: MM: SS.

Note: Neither eDeveloper nor the RDBMS can perform data validation on an Alpha_ Date field due to the use of the internal date format in the RDBMS. If you use the SQL_ DateTOALPHA parameter you should implement your own validity checks for Insert and Update operations. Otherwise, invalid dates can be inserted in the database.

- **SQL_PHYSICAL_LOCKING**

MSSQL supports physical locking at the row level as a default in eDeveloper Version 9. In order to use logical locking (as in previous versions), add the flag:

```
SQL_PHYSICAL_LOCKING=N
```

in the Database Information field in the Database properties. When this flag is set to Y or the flag is not set at all, eDeveloper sends the UPDLOCK table hint in the FROM clause as described below:

```
SELECT  a,b
FROM    tab1  (UPDLOCK NOWAIT)
ORDER BY a
```

When this flag is set to N, the locking behavior will be logical locking as it was in previous versions.

- **SQLOwner**

The MSSQL gateway supports opening an MCF file whose owner is different from the connection owner and is identified by NT authentication.

A user wanting to use NT authentication instead of SQL user, only needs to leave the User and Password fields empty in the Database entry of the Database repository. In this case, the owner is determined according to the login information specified in the MSSQL server. A problem is that you cannot use the NT authentication when your MCF belongs to another owner (other than the one specified in the login information of the MSSQL server).

To solve the problem, simply specify the owner of the MCF in the Database Information field of the Database entry in the format:

```
SQLOwner="xxxx"
```

where 'xxxx' is the owner name.

Table Properties

- SQLBLOB

The parameter `SQLBLOB= n` allows you to specify `n`, the size in bytes of the largest BLOB in a table. The default value is 65534. The maximum value is 2147483647. The value of SQLBLOB is relevant only in Direct SQL and Convert in the following two cases:

- when executing a stored procedure with BLOBs from Direct SQL

or

- when using a direct SQL statement with a Result Database in Btrieve, or when converting a file to Btrieve, Btrieve has a limitation of 65534 for each BLOB. You can reduce the value of SQLBLOB to below 65534, but any attempt to increase it above 65534 will have no effect.

Direct SQL

BLOB variables are not supported as input parameters for Direct SQL statements used in an MSSQL database.

Informix Database Gateway

eDeveloper Data Types

The following table lists eDeveloper attributes with the supported storage types and the valid data types in Informix. Each entry in the table has a default data type that can be forced by specifying that type in the Type property on the SQL tab in the Column Properties dialog.

Note: n is the specified picture number.

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	Informix Data Type
Alpha	String	n,1-32700	n,1-32700	CHAR
	Lstring	n, 1-32700	n+1, 2-32701	CHAR
	Zstring	n, 1-32700	n+1, 2-32701	CHAR
Numeric	Signed Integer	n,1-4	2	SMALLINT
		n, 5-9	4	INTEGER
	Unsigned Integer	n, 1-2	2	CHAR
		n, 3-4	3	CHAR
		n, 5-9	5	CHAR
	Float	n, 1-7	4	FLOAT
		n, 8-16	8	DOUBLE
	Float MS-Basic	n, 1-7	4	CHAR
		n, 8-16	8	CHAR
	Float Decimal	n, 1-7	4	CHAR
		n, 8-16	8	CHAR
	Packed Decimal	n, 1-10	2*n-1	CHAR
	Numeric	n ,1-18	n	CHAR
	Character Number	n, 1-18	n-1	CHAR
	String Number	n, 1-19 ¹	n+1	CHAR
	eDeveloper Number	n, 1-18	n/2+1 ¹	CHAR
	C-Isam Decimal	n, 1-32	n/2+1 ¹	CHAR

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	Informix Data Type
Logical	Integer Logical	5	n,1-2	SMALLINT
	String Logical	5	1	CHAR
Date	String Date	##/##/####	8	Date
	Integer Date	##/##/####	4	INTEGER
	Integer Date 1901	##/##/####	4	INTEGER
	YYMMDD Date	##/##/####	4	CHAR
	eDeveloper Date	##/##/####	4	CHAR
	eDeveloper Date 1901	##/##/####	4	CHAR
Time	String Time	HH:MM:SS	6	CHAR ²
	Integer Time	HH:MM:SS	4	INTEGER
	HMSH Time	HH:MM:SS	4	NUMBER
	eDeveloper Time	HH:MM:SS	4	CHAR
Memo	String Memo	n, 1-1998	n+2	CHAR
	eDeveloper Memo	n, 1999 and above	n+2	CHAR
BLOB	Binary Large Object		12 (default)	BYTE

¹ - Note that eDeveloper's number support is up to 18 digits.

² - If you want to map eDeveloper's String Time to Informix Date data type, specify 'MGTime' as the Type property under the column properties/SQL.

Informix Data Types

The following table shows the results of an eDeveloper Get Definition operation from an Informix Table, and shows eDeveloper equivalents for the Informix database data types.

Informix Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
CHAR(n)	Alpha	Zstring	n+1, 2-32701	n, 1-32700
VARCHAR(n)	Alpha	Zstring	n+1, 2- 256	n, 1-255
INTEGER	Numeric	Signed Integer	4	4
SMALLINT	Numeric	Signed Integer	2	2
FLOAT	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
SMALLFLOAT	Numeric	Float	4	4
DOUBLE	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
DECIMAL(p,s)	Numeric	Float	8	8
MONEY(p,s)	Numeric	Float	8	8
Date	Date	String Date	8	DD/MM/YYYY

Informix Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
DateTime	Alpha	Zstring	1-25	
INTERVAL	Alpha	Zstring	1-24	
BYTE	BLOB	Binary large Object		
SERIAL	Numeric	Signed Integer	4	4

Views and Fragmented Tables

Tables consisting of Views, or fragmented tables without a ROWID, must have a unique index that can be either virtual or real. Certain views defined by Informix as non-modifiable views are Read-Only, and Insert, Update, and Delete operations are not permitted.

It is important to note that eDeveloper considers Views and fragmented tables as regular tables. eDeveloper should not be used to perform any rename or convert operations. An attempt to use these operations will cause eDeveloper to display an

error message and will not do the Convert or Rename of the View in the database.

A View that is defined on more than one table does not have a ROWID, and that is the reason that the Position parameter on the SQL tab in the Table Properties dialog should be set to Unique Index and not to the default.

Table Locking

Table Locking is available only within an eDeveloper transaction. eDeveloper will ignore the request if it is issued outside of a transaction. Therefore, Share None refers to an Exclusive table lock and Share Read refers to a Share table Lock.

Physical Locking

A physical lock is a method that ensures that no one else can modify a record that is currently locked.

Specifically, from the moment a user locks the record until that user releases the record, the record cannot be modified by another user. The physical lock is implemented as follows: When eDeveloper locks a record according to its locking strategy, eDeveloper issues a Select statement with a FOR UPDATE clause.

The FOR UPDATE clause prevents other applications from making changes to the record until the end of the transaction.

Text and Byte Data Types

The user must enter the appropriate picture, after performing a Get Definition on an Informix table, which contains a column of type Text or BYTE. By default, the field that receives the column will have a picture of 0. This picture is invalid, and the user must enter a valid picture between 1 and 32000.

DB2 Database Gateway

eDeveloper Data Types

The following table lists eDeveloper attributes with the supported storage types and the valid data types in DB2. Each entry in the table has a default data type that can be forced by specifying that type in the Type property on the SQL tab in the Column Properties dialog.

Note: n is the specified picture number.

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	DB2 Data Type
Alpha	String	n,1-254	n,1-255	BINARY
		n, 255-32700	n, 256-32700	LONGVARBINARY
	Lstring	n, 1-254	n+1, 2-255	BINARY
	Zstring	n, 1-254	n+1, 2-255	CHAR
Numeric	Signed Integer	n, 255-32700	n+1, 256-32701	LONGVARCHAR
	Unsigned Integer	n,1-4	2	SMALLINT
		n, 5-9	4	INTEGER
		n, 1-2	1	SMALLINT
		n, 3-4	2	SMALLINT
	Float	n, 5-9	4	INTEGER
		n, 1-7	4	FLOAT
		n, 8-16	8	DOUBLE
		n, 1-7	4	BINARY
	Float MS-Basic	n, 8-16	8	BINARY
		n, 1-7	4	BINARY
		n, 8-16	8	BINARY
		n, 1-7	4	BINARY
	Float Decimal	n, 8-16	8	BINARY
		n, 1-10	2*n-1	BINARY
	Packed Decimal	n, 1-18	n	BINARY
	Numeric	n, 1-18	n-1	BINARY
	Character Number			

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	DB2 Data Type
	String Number	n, 1-19 ¹	n+1	BINARY
	eDeveloper Number	n, 1-18	n/2+1 ¹	BINARY
	C-Isam Decimal	n, 1-32	n/2+1 ¹	BINARY
Logical	Integer Logical	5	n,1-2	BINARY
	String Logical	5	1	BINARY
Date	String Date	##/##/ ####	8	Date ²
	Integer Date	##/##/ ####	4	BINARY
	Integer Date 1901	##/##/ ####	4	BINARY
	YYMMDD Date	##/##/ ####	4	BINARY
	eDeveloper Date	##/##/ ####	4	BINARY
	eDeveloper Date 1901	##/##/ ####	4	BINARY
Time	String Time	HH:MM:SS	6	Time ³
	Integer Time	HH:MM:SS	4	BINARY
	HMSH Time	HH:MM:SS	4	BINARY
	eDeveloper Time	HH:MM:SS	4	BINARY
Memo	String Memo	n, 1-255	n+2	BINARY
	eDeveloper Memo	n, 255 and above	n+2	LONGVARBINARY

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	DB2 Data Type
BLOB	Binary Large Object		12 (default)	BLOB

¹ - Note that eDeveloper's number support is up to 18 digits.

² - If you want to map eDeveloper String Date to DB2 character, specify 'CHAR(8)' as the Type under the column properties/SQL.

³ - If you want to map eDeveloper's String Time to DB2 Time data type, you can also specify 'MGTime' as the Type property under the column properties/SQL.

DB2 Data Types

The following table shows the results of an eDeveloper Get Definition operation from a DB2 Table, with eDeveloper equivalents for DB2 data types.

DB2 Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
CHAR(n)	Alpha	Zstring	n+1, 2- 255	n, 1-254
VARCHAR(n)	Alpha	Zstring	n+1, 2- 4001	n, 1-4000
LONGVARCHAR(n)	Alpha	Zstring	n+1, 2- 32701	n, 1-32700
INTEGER	Numeric	Signed Integer	4	10
SMALLINT	Numeric	Signed Integer	2	5
DECIMAL(p, s)	Numeric	Float	8	p-s, s

DB2 Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
DOUBLE	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
FLOAT	Numeric	Float	4	5.2
Date	Date	String Date	8	##/##/####
Time	Time	String Time	6	##:##:##
BLOB	BLOB	Binary Large Object	Default (12)	
TimeSTAMP	Date	String Date	8	8

Views

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted.

It is important to note that eDeveloper considers a View as a regular table. eDeveloper should not be used to perform any rename or convert operations. If eDeveloper is used to execute any rename or convert operations, eDeveloper will display an error message and will not convert or rename the View in the database.

Physical Locking

A physical lock is a method that ensures that no one else can modify a record that is currently locked. Specifically, from the moment a user locks the record until that user releases the record, the record cannot be modified by another user. The physical lock is implemented as follows: When eDeveloper locks a record according to its locking strategy, eDeveloper issues a Select statement with a FOR UPDATE clause.

The FOR UPDATE clause prevents other applications from making changes to the record until the end of the transaction.

Using DB2 Handles

The DB2 Gateway is written in CLI, which requires the use of handles.

Handles are data objects that contain information about a single SQL statement. The handles are allocated at the beginning and released when the SQL statement is no longer used. Each eDeveloper cursor is a SQL statement that correlates to a DB2 handle. The statement handle is allocated the first time the cursor is opened; specifically the first time eDeveloper tries to fetch records from the table or file.

The statement handle is released only when the cursor is released or dropped. This is done because the cursor can be opened more than once. Link cursors are reopened for each record fetched, while main cursors are not reopened every time you go to the next page. Reallocating the statement each time will degrade performance.

As a result, for non-resident tasks, the handle is released when the task is closed, or for resident tasks, the handle is released when you exit eDeveloper runtime.

In DB2, the open handles limit is based on the isolation level: for each isolation level there is a limit of open handles. In eDeveloper, the isolation level can be set in the database level, and tables can be mapped to a different database in eDeveloper. This will result in different handles remaining open in different isolation levels, and the number of open handles remaining higher.

ODBC Database Gateway

This section describes the options available for the eDeveloper Database Gateway for ODBC.

eDeveloper Data Types

The following table lists eDeveloper attributes with the supported storage types and the valid data types in ODBC. Each entry in the table has a default data type that can be forced by specifying that type in the Type property on the SQL tab in the Column properties dialog.

Note: n is the specified picture number.

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	ODBC Data Type
Alpha	String	n,1-8000	n,1-8000	CHAR
		n, 8001-32700	n, 8001-32700	Text
	Lstring	n, 1-255	n+1, 2-256	BINARY
	Zstring	n, 1-8000	n+1, 2-8001	CHAR
		n, 8001-32700	n+1, 8002-32701	Text
Numeric	Signed Integer	n,1-4	2	SMALLINT
		n, 5-9	4	INTEGER
	Unsigned Integer	n, 1-2	1	BINARY
		n, 3-4	2	BINARY
		n, 5-9	4	BINARY
	Float	n, 1-7	4	REAL

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	ODBC Data Type
		n, 8-16	8	DOUBLE PRECISION
	Float MS-Basic	n, 1-7	4	BINARY
		n, 8-16	8	BINARY
	Float Decimal	n, 1-7	4	BINARY
		n, 8-16	8	BINARY
	Packed Decimal	n, 1-10	2*n-1	BINARY
	Numeric	n, 1-18	n	BINARY
	Character Number	n, 1-18	n-1	BINARY
	String Number	n, 1-19 ¹	n+1	BINARY
	eDeveloper Number	n, 1-18	n/2+1 ¹	BINARY
	C-Isam Decimal	n, 1-32	n/2+1 ¹	BINARY
Logical	Integer Logical	5	1	BIT
		5	2	SMALLINT
	String Logical	5	1	BINARY
Date	String Date	##/##/####	8	DateTime ²
	Integer Date	##/##/####	4	INTEGER

eDeveloper Attribute	eDeveloper Storage Type	eDeveloper Picture	eDeveloper Storage Size	ODBC Data Type
Time	Integer Date 1901	##/##/####	4	INTEGER
	YYMMDD Date	##/##/####	4	BINARY
	eDeveloper Date	##/##/####	4	BINARY
	eDeveloper Date 1901	##/##/####	4	BINARY
	String Time	HH:MM:SS	6	CHAR ³
	Integer Time	HH:MM:SS	4	INTEGER
	HMSH Time	HH:MM:SS	4	BINARY
	eDeveloper Time	HH:MM:SS	4	BINARY
Memo	String Memo	n, 1-7998	n+2	VARBINARY
	eDeveloper Memo	n, 7999 and above	n+2	IMAGE
BLOB	Binary Large Object		12 (default)	IMAGE

¹ - Note that eDeveloper's number support is up to 18 digits.

² - If you want to map eDeveloper String Date to ODBC character, specify 'SQL_CHAR(8)' as the Type under the column properties/SQL.

³ - If you want to map eDeveloper's String Time to ODBC DateTime data type, you can also specify 'MGTime' as the Type property under the column properties/SQL.

ODBC Data Types

The following table shows the results of an eDeveloper Get Definition operation from an ODBC table, and shows eDeveloper equivalents for Microsoft ODBC data types.

ODBC Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
SQL_CHAR	Alpha	Zstring	n+1, 2- 256	n, 1-255
SQL_VARCHAR	Alpha	Zstring	n+1, 2- 256	n, 1-255
SQL_LONGVARCHAR ¹	Alpha	Zstring	Default (0)	Default (0) 1
SQL_BIGINT	Numeric	Signed Integer	4	10
SQL_INTEGER	Numeric	Signed Integer	4	10
SQL_SMALLINT	Numeric	Signed Integer	2	5
SQL_TINYINT	Numeric	Unsigned Integer	1	3
SQL_DOUBLE	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
SQL_FLOAT	Numeric	Float	4	5.2
SQL_REAL	Numeric	Float	4	5.2

ODBC Data Type	Attribute	eDeveloper Storage Type	Storage Size	Picture
SQL_NUMERIC	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
SQL_DECIMAL	Numeric	Float	8	According to the Float property in the DBMS Properties (default – 10.3)
SQL_Date	Date	String Date	8	##/##/####
SQL_Time	Time	String Time	6	##:##:##
SQL_BINARY	Alpha	Zstring	N+1, 2- 256	n, 1-255
SQL_VARBINARY	Alpha	Zstring	N+1, 2- 256	n, 1-255
SQL_LONGVARBINARY	BLOB	Binary Large Object	Default (12)	
SQL_BIT	Logical	Integer Logical	1	5
TimeSTAMP	Date	String Date	8	##/##/####

[†] -You must set the Picture for SQL_LONGVARCHAR data type columns to the appropriate size for your application after getting the table's definition.

Locking

No explicit table locks exist in ODBC. Therefore, Share-None and Share-Read have no effect. The underlying database engine itself executes the locking.

Troubleshooting

Use the following layered approach to solve problems you may encounter when using the eDeveloper Database Gateway for ODBC. Test the ODBC driver. If you can access the data successfully but the gateway is still not functioning correctly, test the ODBC driver itself using a tool such as Microsoft Query or ODBC Test. If the data can be accessed through those tools, then the ODBC driver is correctly installed. You can also use the ODBC Check Driver utility provided by eDeveloper. For more information, refer to the ODBC Check Driver Utility section in this chapter.

Views

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted on Views. It is important to note that eDeveloper considers a View as a regular table. eDeveloper should not be used to perform any rename or convert operations. If eDeveloper is used to execute any rename or convert operations, eDeveloper will display an error message and will not convert or rename the View in the database.

Database Default Values

ODBC does not support default values.

Sort/Temporary Database

ODBC cannot be the database for Sort or Temporary tables.

Direct SQL

ODBC cannot be the Result Database in Direct SQL tasks.

ODBC Check Driver Utility

The ODBC Check Driver utility is included with the ODBC driver. Running this program tests the ODBC data source and prints a list of functions that are produced by the driver. The test will tell you whether or not the driver can be used with the eDeveloperGate Database Gateway for ODBC.

The MGCHKDRV utility was built to check any driver of ODBC for information.

The log file used in this example was produced by running the utility against an MS Access data source, but the structure of the log is the same for all data sources and differences appear only in the retrieved text. Text appearing in `courier 10` font comes from the log file.

Follow these steps to execute an SQL statement using the ODBC utility. The first part of each step, shown in *italics*, indicates the option that you can activate from the ODBC Test utility menu.

1. *Connect\Full Connect* – connect to the database
2. *Statement\AllocStmt* – allocate a statement handle
3. *Statement\SQLExecDirect* - execute the SQL statement you want
4. *Results\SQLGetData All* – get all the retrieved data
5. *Statement\FreeStmt* – free the statement
6. *Connect\Full Disconnect* – disconnect from the database

The Check Driver utility retrieves information about the data source in eight sections:

Section No.	Information in Section
1	Driver and DBMS Product Information
2	Data Source Information

Section No.	Information in Section
3	SQL statements supported by the datasource
4	SQL Limits
5	DBMS Type support
6	SQL_KEYWORDS
7	Functions
8	Example for the syntax of DML commands

ODBC Gateway - Data Source Information

The following information appears in the mgchckdrv.log file, or in any other log file you specify.

Section 1: Driver and DBMS Product Information

The utility connects to the data source selected and sends the SQLGetInfo function, which returns general information about the driver associated with the connection's allocated handle.

In order to retrieve the same information, simply connect to the data source using *Connect\Full Connect*, and select *Connect\SQLGetInfo* with the specific flInfoType you want. The result is in the rgbInfoValue field in the result window.

```
SQL_DBMS_NAME = ACCESS
```

A character string with the name of the DBMS product accessed by the driver.

```
SQL_ODBC_VER = 03.51
```

A character string with the version of ODBC to which the Driver Manager conforms. The version is of the form *##.##*, where the first two digits are the major version and the next two digits are the minor version. This is implemented solely in the Driver Manager.

```
SQL_DRIVER_NAME = odbcjt32.dll
```

A character string with the filename of the driver used to access the data source.

```
SQL_DRIVER_ODBC_VER = 03.51, major = 3, minor = 51
```

A character string with the version of ODBC that the driver supports. The version is of the form `##.##`, where the first two digits are the major version and the next two digits are the minor version. `SQL_SPEC_MAJOR` and `SQL_SPEC_MINOR` define the major and minor version numbers. For the version of ODBC described in this manual, these are 2 and 0, and the driver should return "02.00". If a driver supports `SQLGetInfo` but does not support this value of the `flInfoType` argument, the Driver Manager returns "01.00".

```
SQL_DRIVER_VER = 04.00.3513
```

A character string with the version of the driver and, optionally a description of the driver. At a minimum, the version is of the form `##.##.####`, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version.

```
SQL_DBMS_VER = 01.00.0000
```

A character string indicating the version of the DBMS product accessed by the driver. The version is of the form `##.##.####`, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The driver must render the DBMS product version in this form, but can also append the DBMS product-specific version as well. For example, "04.01.0000 Rdb 4.1".

```
SQL_DATABASE_NAME = C:\gateways\ODBCSDK\SMPLDATA\ACCESS\SAMPLE
```

A character string with the name of the current database in use, if the data source defines a named object called "database."

Note: In ODBC 2.0, this value of `flInfoType` has been replaced by the `SQL_CURRENT_QUALIFIER` connection option. ODBC 2.0 drivers should continue to support the `SQL_DATABASE_NAME` information type, and ODBC 2.0 applications should only use it with ODBC 1.0 drivers.

```
SQL_ACTIVE_CONNECTIONS = 64
```

A 16-bit integer value specifying the maximum number of active connection handles that the driver can support. This value can reflect a limitation imposed

by either the driver or the data source. If there is no specified limit or the limit is unknown, this value is set to zero.

`SQL_ACTIVE_Statements = 0`

A 16-bit integer value specifying the maximum number of active statement handles that the driver can support for a connection handle. This value can reflect a limitation imposed by either the driver or the data source. If there is no specified limit or the limit is unknown, this value is set to zero.

`SQL_ODBC_API_CONFORMANCE = Level 1 Supported`

A 16-bit integer value indicating the level of ODBC conformance:

`SQL_OAC_NONE = None`

`SQL_OAC_Level1 = Level 1 supported`

`SQL_OAC_Level2 = Level 2 supported`

`SQL_SEARCH_PATTERN_ESCAPE = \`

A character string specifying what the driver supports as an escape character that permits the use of the pattern match meta-characters underscore (_) and percent (%) as valid characters in search patterns. This escape character applies only for those catalog function arguments that support search strings. If this string is empty, the driver does not support a search-pattern escape character. This flInfoType is limited to catalog functions.

`SQL_SERVER_NAME = ACCESS`

A character string with the actual data source-specific server name; useful when a data source name is used during SQLConnect, SQLDriverConnect, and SQLBrowseConnect.

Section 2: Data Source Information

The utility uses the existing connection to the data source selected and sends the function SQLGetInfo, which returns general information about the data source associated with the connection's allocated handle.

In order to retrieve the same information, simply connect to the data source using *Connect\Full Connect*, and select *Connect\SQLGetInfo* with the specific flInfoType you want. The result is in the rgbInfoValue field in the result window.

SQL_DATA_SOURCE_NAME = try-access

A character string with the data source name used during connection. If the application called SQLConnect, this is the value of the szDSN argument. If the application called SQLDriverConnect or SQLBrowseConnect, this is the value of the DSN keyword in the connection string passed to the driver. If the connection string did not contain the DSN keyword (such as when it contains the DRIVER keyword), this is an empty string.

SQL_ACCESSIBLE_TABLES = Y

A character string:

"Y" if the user is guaranteed SELECT privileges to all tables returned by SQLTables,
"N" if there may be tables returned that the user couldn't access.

SQL_CONCAT_NULL_BEHAVIOR =

Concatenation of a NULL value with no NULL value result is concatenation of non NULL valued

A 16-bit integer value indicating how the data source handles the concatenation of NULL valued character data type columns with non-NULL valued character data type columns:

SQL_CB_NULL = Result is NULL valued.

SQL_CB_NON_NULL = Result is concatenation of non-NULL valued column or columns.

SQL_DATA_SOURCE_READ_ONLY = N

A character string:

"Y" if the data source is set to READ ONLY mode.

"N" if the data source is not set to READ ONLY mode.

This characteristic pertains only to the data source itself; it is not a characteristic of the driver that enables access to the data source.

SQL_CURSOR_COMMIT_BEHAVIOR = Close cursors

A 16-bit integer value indicating how a COMMIT operation affects cursors and prepared statements in the data source:

SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must re-prepare and re-execute the statement handle.

SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call SQLExecute on the statement handle without calling SQLPrepare again.

SQL_CB_PRESERVE = Preserve cursors in the same position as before the COMMIT operation. The application can continue to fetch data or it can close the cursor and re-execute the statement handle without re-preparing it.

SQL_CURSOR_Rollback_BEHAVIOR = Close cursors

A 16-bit integer value indicating how a Rollback operation affects cursors and prepared statements in the data source:

SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must again prepare and reexecute the statement handle.

SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call SQLExecute on the statement handle without calling SQLPrepare again.

SQL_CB_PRESERVE = Preserve cursors in the same position as before the Rollback operation. The application can continue to fetch data or it can close the cursor and re-execute the statement handle without re-preparing it.

SQL_DEFAULT_TXN_ISOLATION = SQL_TXN_READ_COMMITTED

A 32-bit integer that indicates the default transaction isolation level supported by the driver or data source, or zero if the data source does not support transactions. The following terms are used to define transaction isolation levels:

- **Dirty Read** - Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed.
- **Non-repeatable Read** - Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to

reread the row, it will receive different row values or discover that the row has been deleted.

- **Phantom** - Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 re-executes the statement that read the rows, it receives a different set of rows.

If the data source supports transactions, the driver returns one of the following bit-masks:

- `SQL_TXN_READ_UNCOMMITTED` = Dirty reads, non-repeatable reads, and phantoms are possible.
- `SQL_TXN_READ_COMMITTED` = Dirty reads are not possible. Non-repeatable reads and phantoms are possible.
- `SQL_TXN_RepEATABLE_READ` = Dirty reads and non-repeatable reads are not possible. Phantoms are possible.
- `SQL_TXN_SERIALIZABLE` = Transactions are serializable. Dirty reads, non-repeatable reads, and phantoms are not possible.
- `SQL_TXN_VERSIONING` = Transactions are serializable, but higher concurrency is possible than with `SQL_TXN_SERIALIZABLE`. Dirty reads are not possible. Typically, `SQL_TXN_SERIALIZABLE` is implemented by using locking protocols that reduce concurrency and `SQL_TXN_VERSIONING` is implemented by using a non-locking protocol such as record versioning. Oracle's Read Consistency isolation level is an example of `SQL_TXN_VERSIONING`.

`SQL_MULT_RESULT_SETS = N`

A character string:

"Y" if the data source supports multiple result sets.

"N" if it does not.

`SQL_MULTIPLE_ACTIVE_TXN = Y`

A character string:

"Y" if active transactions on multiple connections are allowed.

"N" if only one connection at a time can have an active transaction.

SQL_NEED_LONG_DATA_Len = N

A character string:

"Y" if the data source needs the length of a long data value (the data type is SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data source-specific data type) before that value is sent to the data source.

"N" if it does not.

For more information, see SQLBindParameter and SQLSetPos.

SQL_Owner_Term = NULL StrING

A character string with the data source vendor's name for an owner. For example, owner, Authorization ID, or Schema.

SQL_NULL_COLLATION = NULLs are sorted at the low end of the list

A 16-bit integer value specifying where NULLs are sorted in a list:

- SQL_NC_END = NULLs are sorted at the end of the list, regardless of the sort order.
- SQL_NC_HIGH = NULLs are sorted at the high end of the list.
- SQL_NC_LOW = NULLs are sorted at the low end of the list.
- SQL_NC_START = NULLs are sorted at the start of the list, regardless of the sort order.

For retrieving the value of SQL_AUTOCOMMIT, described immediately below, the utility uses the existing connection to the data source selected and sends the function SQLGetConnectOption, which returns the current settings of a connection option.

In order to retrieve the same information, simply use *Connect\Full Connect* to connect to the data source, and select *Connect\SQLGetConnectOption* with the specific fOption you want (SQL_AUTOCOMMIT in this case). The result is in the pvParam field in the result window.

SQL_AUTOCOMMIT on connection = SQL_AUTOCOMMIT_ON

A 32-bit integer value that specifies whether to use auto-commit or manual-commit mode:

- `SQL_AUTOCOMMIT_OFF` = The driver uses manual-commit mode, and the application must explicitly commit or roll back transactions with `SQLTransact`.
- `SQL_AUTOCOMMIT_ON` = The driver uses auto-commit mode. Each statement is committed immediately after it is executed. This is the default. Note that changing from manual-commit mode to auto-commit mode commits any open transactions on the connection.
- Important: Some data sources delete the access plans and close the cursors for all statement handles on a connection handle each time a statement is committed. Autocommit mode can cause this to happen after each statement is executed. For more information, see the `SQL_CURSOR_COMMIT_BEHAVIOR` and `SQL_CURSOR_Rollback_BEHAVIOR` information types in `SQLGetInfo`.
- `SQL_USER_NAME` = admin
- A character string with the name used in a particular database, which can be different than login name.

Section 3: SQL Statements Supported by the Datasource

The utility uses the existing connection to the data source selected and sends the function `SQLGetInfo`, which returns information about the attributes of SQL statements supported by the data source associated with the connection's handle allocated.

In order to retrieve the same information, simply use *Connect\Full Connect* to connect to the data source, and select *Connect\SQLGetInfo* with the specific `fInfoType` you want. The result is in the `rgbInfoValue` field in the result window.

`SQL_CORRELATION_NAME` = Correlation names are supported and can be any valid name.

A 16-bit integer indicating if table correlation names are supported:

- `SQL_CN_NONE` = Correlation names are not supported.

- `SQL_CN_DIFFERENT` = Correlation names are supported, but must differ from the names of the tables they represent.
- `SQL_CN_ANY` = Correlation names are supported and can be any valid user-defined name.
- `SQL_IDENTIFIER_CASE` = Case sensitive, stored in mixed case

A 16-bit integer value as follows:

- `SQL_IC_UPPER` = Identifiers in SQL are case insensitive and are stored in upper case in system catalog.
- `SQL_IC_Lower` = Identifiers in SQL are case insensitive and are stored in lower case in system catalog.
- `SQL_IC_SENSITIVE` = Identifiers in SQL are case sensitive and are stored in mixed case in system catalog.
- `SQL_IC_MIXED` = Identifiers in SQL are case insensitive and are stored in mixed case in system catalog.
- `SQL_NON_NULLABLE_COLUMNS` = All columns must be nullable

A 16-bit integer specifying whether the data source supports non-nullable columns:

- `SQL_NNC_NULL` = All columns must be nullable.
- `SQL_NNC_NON_NULL` = Columns may be non-nullable (the data source supports the NOT NULL column constraint in CREATE TABLE statements).
- `SQL_ODBC_SQL_CONFORMANCE` = Minimum grammar supported

A 16-bit integer value indicating SQL grammar supported by the driver:

- `SQL_OSC_MINIMUM` = Minimum grammar supported
- `SQL_OSC_CORE` = Core grammar supported
- `SQL_OSC_EXTENDED` = Extended grammar supported
- `SQL_QUOTED_IDENTIFIER_CASE` = Case insensitive, stored in mixed case

A 16-bit integer value as follows:

- `SQL_IC_UPPER` = Quoted identifiers in SQL are case insensitive and are stored in upper case in system catalog.
- `SQL_IC_Lower` = Quoted identifiers in SQL are case insensitive and are stored in lower case in system catalog.
- `SQL_IC_SENSITIVE` = Quoted identifiers in SQL are case sensitive and are stored in mixed case in system catalog.
- `SQL_IC_MIXED` = Quoted identifiers in SQL are case insensitive and are stored in mixed case in system catalog.

Section 4: SQL Limits

The utility uses the existing connection to the data source selected and sends the function `SQLGetInfo`, which returns information about the limits applied to identifiers and clauses in SQL statements, such as the maximum lengths of identifiers and the maximum number of columns in a select list.

Important note: Either the driver or the data source may impose limitations.

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Connect\SQLGetInfo* with the specific `fInfoType` you want. The result is in the `rgbInfoValue` field in the result window.

`SQL_MAX_COLUMN_NAME_Len = 64`

A 16-bit integer value specifying the maximum length of a column name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

`SQL_MAX_QUALIFIER_NAME_Len = 260`

A 16-bit integer value specifying the maximum length of a qualifier name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

`SQL_MAX_TABLE_NAME_Len = 64`

A 16-bit integer value specifying the maximum length of a table name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

SQL_MAX_INDEX_SIZE = 255

A 32-bit integer value specifying the maximum number of bytes allowed in the combined fields of an index. If there is no specified limit or the limit is unknown, this value is set to zero.

SQL_MAX_COLUMNS_IN_INDEX = 10

A 16-bit integer value specifying the maximum number of columns allowed in an index. If there is no specified limit or the limit is unknown, this value is set to zero.

SQL_MAX_COLUMNS_IN_ORDER_BY = 10

A 16-bit integer value specifying the maximum number of columns allowed in an ORDER BY clause. If there is no specified limit or the limit is unknown, this value is set to zero.

SQL_MAX_COLUMNS_IN_SELECT = 255

A 16-bit integer value specifying the maximum number of columns allowed in a select list. If there is no specified limit or the limit is unknown, this value is set to zero.

SQL_MAX_COLUMNS_IN_TABLE = 255

A 16-bit integer value specifying the maximum number of columns allowed in a table. If there is no specified limit or the limit is unknown, this value is set to zero.

Section 5: DBMS Type Support

The utility uses the existing connection to the data source selected and sends the function `SQLGetInfo` with the `fInfoType` as `SQL_ALL_TYPES`, which returns information about all the data types supported. After receiving this information, the utility sends the function `SQLGetTypeInfo` for every data type in order to see if the data type is searchable (in other words, if the data type can be used in a WHERE clause).

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Catalogue\SQLGetTypeInfo* with `fSQLType = 'SQL_ALL_TYPES'`. After that, select *Results\GetData All*. You will see the list of data types in the result window. To retrieve the `SEARCHABLE`

parameter for each data type, select *Catalogue\SQLGetTypeInfo* with fSQLOption of the data type you want. Than select Results\SQLBindCol when icol = 9 (the SEARCHABLE parameter is the ninth parameter in this function). After that, select Results\SQLFetch. The last parameter in the result window (rgbValue) is the SEARCHABLE parameter value. The values are:

- 0 – SQL_UNSEARCHABLE if the data type cannot be used in a WHERE clause.
- 1 – SQL_LIKE_ONLY if the data type can be used in a WHERE clause only with the LIKE predicate.
- 2 – SQL_ALL_EXCEPT_LIKE if the data type can be used in a WHERE clause with all comparison operators except LIKE.
- 3 – SQL_SEARCHABLE if the data type can be used in a WHERE clause with any comparison operator

The information in the utility log file is listed in 3 columns:

- 1. DBMS Type Name – The name of the data type in the DBMS.
- 2. SQL Data Type – the equivalent core SQL data type defined by ODBC
- 3. Searchable – a True\False value that indicates if the data type can appear in a WHERE clause.

DBMS TYPE NAME	SQL DATA TYPE	SEARCHABLE
GUID	INVALID SQLTYPE	FALSE
BIT	SQL_BIT	TRUE
BYTE	SQL_TINYINT	TRUE
LONGBINARY	SQL_LONGVARBINARY	FALSE
VARBINARY	SQL_VARBINARY	FALSE
BINARY	SQL_BINARY	FALSE
LONGCHAR	SQL_LONGVARCHAR	FALSE
CHAR	SQL_CHAR	TRUE
CURRENCY	SQL_NUMERIC	TRUE
INTEGER	SQL_INTEGER	TRUE

DBMS TYPE NAME	SQL DATA TYPE	SEARCHABLE
Counter	SQL_INTEGER	TRUE
SMALLINT	SQL_SMALLINT	TRUE
REAL	SQL_REAL	TRUE
DOUBLE	SQL_DOUBLE	TRUE
DateTime	SQL_TimeStamp	TRUE
VARCHAR	SQL_VARCHAR	TRUE

Section 6: SQL Keywords

The utility uses the existing connection to the data source selected and sends the function SQLGetInfo, with 'SQL_KEYWORDS' in the fInfoType field.

This will return a character string containing a comma-separated list of all data source-specific keywords. This list does not contain keywords specific to ODBC or keywords used by both the data source and ODBC. Keywords are reserved words – you can use them only in specific places according to the syntax.

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Connect\SQLGetInfo* with SQL_KEYWORDS in the fInfoType field. The result is in the rgbInfoValue field in the result window.

```
SQL_KEYWORDS = ALPHANUMERIC,AUTOINCRE-
MENT,BINARY,BYTE,Counter,CURRENCY,DATABASE,DATABASENAME,DateTime,D
ISALLOW,DISTINCTROW,DOUBLEFLOAT,FLOAT4,FLOAT8,GENERAL,IEEEDOUBL
E,IEEESINGLE,IGNORE,INT,INTEGER1,INTEGER2,INTEGER4,Level,Logical,Logical1,
LONG,LONGBINARY,LONGCHAR,LONGText,MEMO,MONEY,NOTE,NUMBER,OLEO
BJECT,OPTION,OwnerACCESS,PARAMETERS,PERCENT,PIVOT,SHORT,SINGLE,SIN
GLEFLOAT,SMALLINT,STDEV,STDEVP,String,TableID,Text,TOP,TRANS-
FORM,UNSIGNEDBYTE,VALUES,VAR,VARBINARY,VARP,YESNO
```

Section 7: Functions

The utility uses the existing connection to the data source selected and sends the function SQLGetFunction, with SQL_API_ALL_FUNCTIONS in the fFunction field.

This will return information about whether a driver supports the list of ODBC functions. Those functions are implemented in the Driver Manager. They can also be implemented in drivers. If a driver implements SQLGetFunctions, the Driver Manager calls the function in the driver. Otherwise, it executes the function itself.

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Misc\SQLGetFunctions All*. The result in the result window will show a list of ODBC function names with a True\False value that identifies if they exist in the data source.

The information in the utility log file is listed in 3 columns:

1. Function – The name of the ODBC function.
2. Supported by driver – Supported\Not Supported according to the True\False value returned from the SQLGetFunction results.
3. Used By eDeveloper – Required\Not Required by eDeveloper.

Important note: Functions that are required by eDeveloper but are not supported by the data source may cause further problems when working with eDeveloper and ODBC with this data source.

Function	Supported by Driver	Used by eDeveloper
SQLAllocConnect Allocates memory for a connection handle within the environment identified by the handle.	Supported	Required
SQLAllocEnv Allocates memory for an environment handle and initializes the ODBC call level interface for use by an application. An application must call SQLAllocEnv prior to calling any other ODBC function.	Supported	Required
SQLAllocStmt Allocates memory for a statement handle and associates the statement handle with the connection specified by a connection handle. An application must call SQLAllocStmt prior to submitting SQL statements.	Supported	Required
SQLBindCol	Supported	Required

Function	Supported by Driver	Used by eDeveloper
Assigns the storage and data type for a column in a result set, including: A storage buffer that will receive the contents of a column of data The length of the storage buffer A storage location that will receive the actual length of the column of data returned by the fetch operation Data type conversion		
SQLCancel	Supported	Required
Cancels the processing of a statement's handle.		
SQLColAttributes	Supported	Not Required
Returns descriptor information for a column in a result set. It cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.		
SQLConnect	Supported	Required
Loads a driver and establishes a connection to a data source. The connection handle references storage of all information about the connection, including status, transaction state, and error information.		
SQLDescribeCol	Supported	Required
Returns the result descriptor column name, type, precision, scale, and nullability for one column in the result set. It cannot be used to return information about the bookmark column (column 0).		
SQLDisconnect	Supported	Required
Closes the connection associated with a specific connection handle.		
SQLError	Supported	Required
Returns error or status information.		
SQLExecDirect	Supported	Required

Function	Supported by Driver	Used by eDeveloper
Executes a preparable statement, using the current values of the parameter marker variables if any parameters exist in the statement. SQLExecDirect is the fastest way to submit an SQL statement for one-time execution.		
SQLExecute	Supported	Required
Executes a prepared statement, using the current values of the parameter marker variables if any parameter markers exist in the statement.		
SQLFetch	Supported	Required
Fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with SQLBindCol.		
SQLFreeConnect	Supported	Required
Releases a connection handle and frees all memory associated with the handle.		
SQLFreeEnv	Supported	Required
Frees the environment handle and releases all memory associated with the environment handle.		
SQLFreeStmt	Supported	Required
Stops processing associated with a specific statement handle, closes any open cursors associated with the statement handle, discards pending results, and, optionally, frees all resources associated with the statement handle.		
SQLGetCursorName	Supported	Not Required
Returns the cursor name associated with a specified statement handle.		
SQLNumResultCols	Supported	Required
Returns the number of columns in a result set.		
SQLPrepare	Supported	Required
Prepares an SQL string for execution		

Function	Supported by Driver	Used by eDeveloper
SQLRowCount	Supported	Required
Returns the number of rows affected by an UPDATE, INSERT, or DELETE statement or by a SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in SQLSetPos.		
SQLSetCursorName	Supported	Not Required
Associates a cursor name with an active statement handle. If an application does not call SQLSetCursorName, the driver generates cursor names as needed for SQL statement processing.		
SQLSetParam	Supported	Not Required
In ODBC 2.0, the ODBC 1.0 function SQLSetParam has been replaced by SQLBindParameter. For more information, see SQLBindParameter.		
SQLTransact	Supported	Required
Requests a commit or rollback operation for all active operations on all handle statements associated with a connection. SQLTransact can also request that a commit or rollback operation be performed for all connections associated with the environment handle.		
SQLColumns	Supported	Required
Returns the list of column names in specified tables. The driver returns this information as a result set on the specified statement's handle.		
SQLDriverConnect	Supported	Required

Function	Supported by Driver	Used by eDeveloper
<p>SQLDriverConnect is an alternative to SQLConnect. It supports data sources that require more connection information than the three arguments in SQLConnect, dialog boxes to prompt the user for all connection information and data sources that are not defined in the ODBC.INI file or registry.</p> <p>SQLDriverConnect provides the following connection options:</p> <p>Establish a connection using a connection string that contains the data source name, one or more user IDs, one or more passwords, and other information required by the data source.</p> <p>Establish a connection using a partial connection string or no additional information; in this case, the driver Manager and the driver can each prompt the user for connection information.</p> <p>Establish a connection to a data source that is not defined in the ODBC.INI file or registry. If the application supplies a partial connection string, the driver can prompt the user for connection information.</p> <p>Once a connection is established, SQLDriverConnect returns the completed connection string. The application can use this string for subsequent connection requests.</p>		
SQLGetConnectOption	Supported	Required
Returns the current setting of a connection option.		
SQLGetData	Supported	Required
<p>Returns result data for a single unbound column in the current row. The application must call SQLFetch, or SQLExtendedFetch and (optionally) SQLSetPos to position the cursor on a row of data before it calls SQLGetData. It is possible to use SQLBindCol for some columns and use SQLGetData for others within the same row. This function can be used to retrieve character or binary data values in parts from a column with a character, binary, or data source-specific data type (for example, data from SQL_LONGVARBINARY or SQL_LONGVARCHAR columns).</p>		
SQLGetFunctions	Supported	Required

Function	Supported by Driver	Used by eDeveloper
Returns information about whether a driver supports a specific ODBC function. This function is implemented in the Driver Manager; it can also be implemented in drivers. If a driver implements SQLGetFunctions, the Driver Manager calls the function in the driver. Otherwise, it executes the function itself.		
SQLGetInfo	Supported	Required
Returns general information about the driver and data source associated with a connection handle.		
SQLGetStmtOption	Supported	Required
Returns the current setting of a statement option.		
SQLGetTypeInfo	Supported	Required
Returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set. Important: Applications must use the type names returned in the TYPE_NAME column in ALTER TABLE and CREATE TABLE statements. SQLGetTypeInfo may return more than one row with the same value in the DATA_TYPE column.		
SQLParamData	Supported	Not Required
Is used in conjunction with SQLPutData to supply parameter data at statement execution time.		
SQLPutData	Supported	Not Required
Allows an application to send data for a parameter or column to the driver at statement execution time. This function can be used to send character or binary data values in parts to a column with a character, binary, or data source-specific data type (for example, parameters of the SQL_LONGVARIABLE or SQL_LONGVARCHAR types).		
SQLSetConnectOption	Supported	Required
Sets options that govern aspects of connections.		
SQLSetStmtOption	Supported	Required

Function	Supported by Driver	Used by eDeveloper
Sets options related to a statement handle. To set an option for all statements associated with a specific connection handle, an application can call SQLSetConnectOption.		
SQLSpecialColumns	Supported	Required
Retrieves the following information about columns within a specified table: The optimal set of columns that uniquely identifies a row in the table. Columns that are automatically updated when any value in the row is updated by a transaction.		
SQLStatistics	Supported	Required
Retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns the information as a result set.		
SQLTables	Supported	Required
Returns the list of table names stored in a specific data source. The driver returns the information as a result set.		
SQLBrowseConnect	Not Supported	Not Required
Supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source. Each call to SQLBrowseConnect returns successive levels of attributes and attribute values. When all levels have been enumerated, a connection to the data source is completed and a complete connection string is returned by SQLBrowseConnect. A return code of SQL_SUCCESS or SQL_SUCCESS_WITH_INFO indicates that all connection information has been specified and the application is now connected to the data source.		
SQLColumnPrivileges	Not Supported	Not Required
Returns a list of columns and associated privileges for the specified table. The driver returns the information as a result set on the specified statement's handle.		

Function	Supported by Driver	Used by eDeveloper
SQLDataSources	Supported	Not Required
Lists data source names. This function is implemented solely by the Driver Manager.		
SQLDescribeParam	Not Supported	Not Required
Returns the description of a parameter marker associated with a prepared SQL statement.		
SQLExtendedFetch	Supported	Required
<p>Extends the functionality of SQLFetch in the following ways:</p> <p>It returns rowset data (one or more rows), in the form of an array, for each bound column.</p> <p>It scrolls through the result set according to the setting of a scroll-type argument.</p> <p>SQLExtendedFetch works in conjunction with SQLSetStmtOption. To fetch one row of data at a time in a forward direction, an application should call SQLFetch.</p>		
SQLForeignKeys	Not Supported	Not Required
<p>SQLForeignKeys can return:</p> <p>A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables).</p> <p>A list of foreign keys in other tables that refer to the primary key in the specified table.</p> <p>The driver returns each list as a result set on the specified statement handle.</p>		
SQLMoreResults	Supported	Not Required
Determines whether there are more results available on a statement handle containing SELECT, UPDATE, INSERT, or DELETE statements and, if so, initializes processing for those results.		
SQLNativeSql	Supported	Not Required
Returns the SQL string as translated by the driver.		

Function	Supported by Driver	Used by eDeveloper
SQLNumParams	Supported	Not Required
Returns the number of parameters in an SQL statement.		
SQLParamOptions	Supported	Not Required
Allows an application to specify multiple values for the set of parameters assigned by SQLBindParameter. The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. An application can, for example, specify three sets of values for the set of parameters associated with an INSERT statement, and then execute the INSERT statement once to perform the three insert operations.		
SQLPrimaryKeys	Not Supported	Not Required
Returns the column names that comprise the primary key for a table. The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.		
SQLProcedureColumns	Supported	Not Required
Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. The driver returns the information as a result set on the specified statement handle.		
SQLProcedures	Supported	Not Required
Returns the list of procedure names stored in a specific data source. Procedure is a generic term used to describe an executable object, or a named entity that can be invoked using input and output parameters, and which can return result sets similar to the results returned by SQL SELECT expressions.		
SQLSetpos	Supported	Not Required
Sets the cursor position in a rowset and allows an application to refresh, update, delete, or add data to the rowset.		
SQLSetScrollOptions	Supported	Not Required

Function	Supported by Driver	Used by eDeveloper
<p>Sets options that control the behavior of cursors associated with a statement handle. SQLSetScrollOptions allows the application to specify the type of cursor behavior desired in three areas: concurrency control, sensitivity to changes made by other transactions, and rowset size.</p> <p>Note: In ODBC 2.0, SQLSetScrollOptions has been superseded by the SQL_CURSOR_TYPE, SQL_CONCURRENCY, SQL_KEYSET_SIZE, and SQL_ROWSET_SIZE statement options. ODBC 2.0 drivers must support this function for backwards compatibility; ODBC 2.0 applications should only call this function in ODBC 1.0 drivers.</p> <p>If an application calls SQLSetScrollOptions, a driver must be able to return the values of the aforementioned statement options with SQLGetStmtOption. For more information, see SQLSetStmtOption.</p>		
SQLTablePrivileges	Not Supported	Not Required
<p>Returns a list of tables and the privileges associated with each table. The driver returns the information as a result set on the specified statement handle.</p>		
SQLDrivers	Supported	Not Required
<p>Lists driver descriptions and driver attribute keywords. This function is implemented solely by the Driver Manager.</p>		
SQLBindParameter	Supported	Required
<p>Binds a buffer to a parameter marker in an SQL statement.</p> <p>Note: This function replaces the ODBC 1.0 function SQLSetParam.</p>		

Section 8: Syntax of DML Commands Example

The utility uses the existing connection to the data source selected and sends the function SQLExecDirect with a SQL command in the szSqlStr field.

After each command, the log lists if the command was successful or not.

Note that the table and index are actually created and dropped from the database that the data source points to.

```
CREATE TABLE T11025 (FLD1 CHAR(1), FLD2 CHAR(1))
```

```
**** Succeeded ****  
CREATE UNIQUE INDEX I11025 ON T11025 (FLD1)  
**** Succeeded ****  
DROP INDEX I11025 ON T11025  
**** Succeeded ****  
DROP TABLE T11025  
**** Succeeded ****
```

The utility disconnects from the data source using the SQLDisconnect function.
In order to do the same, simply select *Connect\Full Disconnect*.

```
***** Disconnected Successfully ... *****  
*****
```


Index

Entries

A

- A LIKE A function 553
- Abort error strategy 998
- Abort task 1001
- ABS function 554
- Access I/O files 387
- Access key 1055
 - applications 61, 1048
 - Force MVCS 1049
 - public rights 1048
 - super rights 1049
 - tables 251
- Access key for a component 1062
- Access mode 1410
- Access to application tables
 - data security 1054
- Access to multi-user environment 381
- ACOS function 554
- ACT literal 511
- Action functions 518
- Actions 134
 - display list of actions 367
 - example 136
 - key 138
 - name 137
 - state conditions 138
 - state qualifications 135
- Activating team development 1418
- ActiveX control 939
- ActiveX data field 207
- ActiveX default 228
- AddDate function 554
- AddTime function 555
- Allow abort 338
- Allow access to
 - applications 77
 - Checker messages 82
 - Color repository 78
 - Communication repository 79
 - Database repository 79
 - DBMS repository 79
 - Environment dialog 78
 - Font repository 78
 - HTML styles repository 80
 - Keyboard mapping repository 78
 - Language repository 80
 - Logical Names repository 80
 - Logon dialog 81
 - Print attribute 81
 - Printer repository 80
 - Server repository 78
 - Service repository 79
 - toolkit 81
 - Visual connection display 79
- Allow create in modify mode 76
- Allow create mode 358
- Allow delete mode 358
- Allow index change 358
- Allow index optimization 359
- Allow input and output files 358
- Allow locate in query mode 359

- Allow locate option 358
- Allow modify mode 358
- Allow options 357
- Allow query mode 358
- Allow range option 358
- Allow sorting 358
- Allow testing environment 81
- Allow update in query mode 77
- Alpha attribute 205
- Alpha String functions 547
- Alpha string manipulation functions 523
- Alpha/Numeric conversion functions 523
- Alternate collating sequence 155
- Alternate collating sequence file 110
- Alternate text 938
- AND operator 515
- ANSI to OEM conversion 1270
- Ansi to Unicode 109
- ANSI2OEM function 555
- API implementation 1432
 - DB2 1433
 - Informix 1433
 - MS-SQL 1433
 - ODBC 1433
 - Oracle 1432
- Application access and share modes 1427
- Application access key 67, 1048
- Application database 1062
- Application Event keywords 1333
- Application file 1061
- Application launch from the command line 190
- Application partitioning
 - benefits 1203
 - functions 1220
- Application properties dialog rights assignment 1047
- Application property keywords 1330
- Application repository 56
- Application startup mode 74
- Application wizard 1223
- Applications
 - access key 61
 - compression 60
 - database 59
 - Magic Flat File (MFF) deployment 60
 - MCF file 59
 - multiple developer environment 60
 - prefix 58
- AppServer priority 119
- Area 793
- Argument list 460
- Arguments 462
- Array size 170, 256
- As strategy, error 1001
- ASC function 555
- ASIN function 556
- Astr function 556
- ATAN function 557
- Attach context menu structure 341
- Attribute
 - alpha 205
 - blob 206
 - date 205
 - logical 205

- memo 206
- numeric 205
- time 205
- Authorization environment 1035
- Authorization for an application 1035
- Authorization for menus 1021
- Auto-exiting 91
- Automatic documentation 1247
- Automatic program generator 1223, 1229
- Automatic program generator rights 1043
- Automatic retry 1002

B

- Background colors 127
- Background engine 123
- Background window 360
- Base currency 64
- Base64ToBlob function 558
- Batch task 1198
- Batch task event handling 285
- Batch task type 334
- Batch tasks
 - deleting records 310
 - record/row loop 316
 - valid Magic operations 421
- Before Task Prefix
 - physical transactions at the task level 979
- Binding variables 355
- Blank keyword parameter 1314
- BLB2FILE function 743
- Blb2File function 557

- Blob attribute 206
- Blob support 1188
- Blob2Req function 558
- BlobFromBase64 function 558
- BlobSize function 558
- Block loop 457
- Block operation 456
- Block Operation keywords 1363
- BOM function 559
- Bookmarks 50, 98
- Border style 795, 1029
- Border width 940, 943
- BOY function 559
- Break variables 312
- Broker error messages 1169
- Browse operation 500
 - keywords 1367
- Browser client cached alias 122
- Browser client cached path 122
- Browser client network error recovery timeout 122
- Browser client technology 121
- Browser client technology error url 122
- Browser control properties 772
- Browser forms 765
- Browser IFRAME control 771, 790
- Browser Opaque control 771, 790
- Browser subform 766
- Browser Table control 788
- Browser task type 334
- Browser-based help screens 1033
- Browser-based program
 - program generator 1225
- Buffer management 518

- BufGetAlpha function 559
- BufGetBit function 561
- BufGetBlob function 562
- BufGetDate function 563
- BufGetLog function 563
- BufGetNum function 564
- BufGetTime function 565
- BufGetVariant function 565
- BufGetVector function 566
- BufSetAlpha function 567
- BufSetBit function 568
- BufSetBlob function 568
- BufSetDate function 569
- BufSetLog function 570
- BufSetTime function 571
- BufSetVariant function 572
- BufSetVector function 572
- Button format 772, 776, 779
- Button type 939

C

Cache

- activating the cache size 994
- changes to program behavior 995
- client/server 996
- internal implementation 996
- linked tables 382
- resident tasks 995
- rollback 996
- tables 252
- task size 76
- what can be cached 993
- when is the cache used 994

Call a public program 471

- Call COM operation 477
- Call expressions 470
- Call operation
 - keywords 1363
 - synchronize data before call 993
- Call program 469
- Call Remote 477
- Call task 465
- Call to a 3rd Generation Language 1152
- Call to a DLL 1151
- Call UDP 475
- Call UDP operation 1153
- Call Web Service 478
- CallDLL function 573
- CallDLLF function 574
- CallDLLS function 574
- Calling a batch task from the Browser Client 1198
- CallJS function 575
- CallOBJ function 575
- CallProg function 576
- CallProgURL function 576
- CallURL function 577
- CASE function 577
- Case sensitivity 225
- CDOW function 578
- Center screen in online 95
- Century Start setting 75
- Check box control 781, 804, 811
- Check definition 166
- Check existence 157, 170
- Check image change time 102
- Check in object 1420

- Check index 167
- Check out object 1419
- Check syntax utility 1231
- Checked Out Object list 1420
- Checker message groups 102
- Checker minimal level 102
- Checker result 39
- CHeight function 578
- Child forms 797
- Child window 374, 793
- Chk_std.dat file path 111
- ChkDgt function 579
- Choice controls
 - combo box control 808
 - list box control 808
 - radio button 807
 - tab control 808
- CHR function 579
- Chunk size 341
- Cipher function 580
- Class numbers 371
- CLASSPATH 125
- CLeft function 582
- CLeftMDI function 583
- ClickCX function 583
- ClickCY function 583
- ClickWX function 584
- ClickWY function 584
- ClientCertificateAdd function 585
- ClientCertificateDiscard function 586
- Client-side identification 345
- ClipAdd 586
- ClipAdd function 586
- ClipRead function 587
- ClipWrite function 587
- ClrCache function 587
- Clustering indexes 272
- CMonth function 588
- CndRange function 588
- CodePage function 589
- Color 795, 923, 940, 943
- Color assignment palette 128
- Color definition file 66, 106
- Color palette 804
- Color print attribute 972
- Color repository 126
- Color setting sample 129
- Column repository 258
- Column repository keywords 1317
- Columns
 - adding a Null value 262
 - ANSI and OEM character sets 263
 - defining the Null default 263
 - displaying Null strings 263
 - field attributes 258
 - field models 258
 - Null as a valid value 262
 - setting a Null default value 263
 - size and definition 264
 - SQL properties 265
 - update style 264
 - visual display of a field 259
- COM objects 1089
- Combo box control 785, 805
- COMError function 589
- COMHandleGet function 590
- COMHandleSet function 590
- Command line 190

- examples 189
- options and Magic.ini values 187, 190
- syntax 187
- values for environment properties 192
- Command Line Requester 1214
- Command palette 375
- Command processor 112
- Commands file 178
- Comment text box for objects in an application 47
- Common Data Dictionary 169
- Communication manager 143
- Communication repository 148
- Communications driver 148
 - identification 149
 - internal code 149
 - parameters 149
- COMObjCreate function 591
- COMObjRelease function 591
- Component access key 1062
- Component event keywords 1323
- Component help keywords 1323
- Component Interface Builder
 - properties 1067
- Component model keywords 1322
- Component program keywords 1322
- Component properties keywords 1324
- Component repository keywords 1321
- Component rights keywords 1323
- Component tables keywords 1322
- Components 1058, 1065
 - repository 1059
- Composite functions 547
- Compound storage functions 523
- Compressed applications 60
- Concatenation of alpha strings 516
- Concurrency 1423
- Condition indicators 1307
- Confirm cancel 362
- Confirm task update 362
- Confirm when auto-exiting 91
- Const (constant) file 105
- Context menu keywords 1327
- Context menus 1015
- Context variables 923, 947
 - cookies 927, 928
 - domain restriction 929
 - expiration date 929
 - expiration time 929
 - hidden fields 927
- Control attributes for form models 231
- Control change event 282
- Control operation level 284, 288, 310
- Control Prefix 281
- Control properties 231
- Control Suffix 281
- Control verification event 281
- Conversion functions 526
- Cookies 927
- Copyright message 93
- Copyright messages 92
- COS function 592
- Count value 226
- Counter function 592
- CRC function 592
- Create link 444

Create rights 1041, 1042, 1043, 1044, 1045
Cross Reference utility 1237
Cross references 51, 99
CTop function 593
CTopMDI function 594
CtrlGoto function 594
CtrlHWND function 595
CtrlName function 595
CtxClose function 595
CtxGetAllNames function 596
CtxGetId function 596
CtxKill function 597
CtxLstUse function 597
CtxNum function 598
CtxProg function 598
CtxSetName function 598
CtxSize function 599
CtxStat function 599
CurROW function 600
CurrPosition function 600
Cursors 1471
CWidth function 601
Cycle record main 361

D

Data definition rules 1434
Data item qualifiers 204
Data manipulation statements 977
Data security 1054
Database functions 527
Databases
 change tables in toolkit mode 166
 check definition 166

 check index 167
 Common Data Dictionary 169
 DBMS types 162
 default 88
 for sort or temporary tables 89
 information to MS-SQL 1471
 information to Oracle 1458
 location 163
 lock path 168
 Oracle connect string 166
 password 165
 properties 164
 properties for MS-SQL 1471
 properties for Oracle 1458
 record locking 168
 repository 159
 server 165
 SQL 169
 user name 165
Data-bound choice controls 808
Dataview
 preparing the dataview 300
 record instance 301
 tuning 300
Date at logon 73
Date attribute 205
Date function 601
Date functions 529
DATE literal 512
Date mode 103
Date pictures 214, 220
Date Separator setting 104
DAY function 601
Day function 601

- DB commands 1470
- DB SQL Where 402
- DB Table keywords 1343
- DB Table repository 380, 381
- DB tables 249
 - expressions 382
 - identifier 380
 - multi-user access 381
 - open mode 381
 - sharing tasks 381
- DB2
 - data types 1481
 - database gateway 1478
 - using handles 1483
- DbCache function 602
- DbCopy function 602
- DbDel function 603
- DbDiscnt function 604
- DbERR function 605
- DbExist function 605
- DBMS
 - customization information 152
 - errors 1007
 - float picture 152
 - internal codes 153
 - nulls 152
 - properties 153
 - repository 149
 - type 162
 - variable MCF record length 158
- DbName function 606
- DbRecs function 606
- DbReload function 607
- DbRound function 608
- DbSize function 609
- DDEBEGIN function 1137
- DDEBegin function 610
- DDEEnd function 610
- DDEGET function 1138
- DDEGet function 610
- DDEPoke function 612
- DDERR function 613, 1140
- DDEXEC function 1141
- DDExec function 614
- Deadlock prevention for ISAM data-bases 86
- Deadlocks and transaction processing 991
- Debugger 1250
- Debugging 434
- Decimal separator 104
- DeCipher function 616
- Default application 74
- Default Button 794
- Default color 97
- Default database 88
- Default font setting 98
- Default image file 938
- Default menus 1017
- Deferred transactions 982
 - locking strategy 983
 - transaction mode 977, 982
- Del function 617
- Delay function 618
- Delete records 309
- Delete rights 1041, 1042, 1043, 1044, 1045
- Deployment custom copyright 93

- Destination context name 508
- Detail line number for table length 789
- Direct SQL 983, 1450
- Direct SQL Command 346
- Dirty reads 156
- DiscSrvr function 618
- Display copyright messages 92
- Display full messages 94
- Display toolbar 94
- Dockable palettes 100
- Documentation 35
 - report sections 1309
 - template facility 1303
 - template facility keywords 1313
- Documentation template file 107
 - comment lines 1309
 - condition indicators 1307
 - control lines 1305
 - data lines 1309
 - footers 1308
 - headers 1308
 - report section delimiters 1305
 - syntax 1305
- DOW function 619
- DragSetCsr function 619
- DragSetData function 619
- Drop data user formats 111
- DROPFORMAT function 623
- DropFormat function 620
- DROPGETDATA function 623
- DropGetData function 621
- DropMouseX function 622
- DropMouseY function 622
- DStr function 623
- DVal function 624
- Dynamic calls 469
- Dynamic Data Exchange 517, 1136

E

- Edit control 772, 804, 809
- EditGet function 624
- EditSet function 625
- EJB interface builder 1074
- EJBCreate function 625
- EJBExplore function 626
- Ellipse control 805
- Encrypt file 1055
- Encrypt table 252
- EncryptionError function 626
- End Block operation 458
- End Link operation 454
- End task condition 337
- End-user access to menu entries 1022
- End-user menu 1018
- End-user screen interaction 304
- End-user terminal identifier 84
- Engine directive 1001
- Engine execution rules 305
- Engine levels 278
- Enterprise server 116, 117
- Enterprise Server functions 530
- Environment authorization 1050
- Environment dialog 69
- Environment functions 532
- Environment settings 53
- EOF function 626
- EOM function 627
- EOP function 627

- EOY function 628
- ERRDATABASENAME function 628, 1008
- ErrDatabaseName function 628
- ERRDBMSCODE function 1009
- ErrDbmsCode function 628
- ERRDBMSMESSAGE function 1009
- ErrDbmsMessage function 628
- ERRMAGICNAME function 1009
- ErrMagicName function 629
- Error events 409
- Error handler 1000
 - abort task 1001
 - as strategy 1001
 - automatic retry 1002
 - DBMS errors 1007
 - enabled 1008
 - error type 1001
 - ignore 1007
 - level 1000
 - rollback and restart 1001
 - scope 1007
 - user retry 1004
- Error messages 433, 444
 - conflicting control parameters 357
- Error strategy
 - abort 998
 - importing applications from previous versions 1011
 - recover 998
- ERRPOSITION function 1009
- ErrPosition function 629
- ERRTABLENAME function 1008
- ErrTableName function 629
- Escape character 225
- EuroCnv function 630
- EuroDel function 630
- EuroGet function 630
- European currency conversion file 67, 111
- EuroSet function 631
- EuroUpd function 631
- EvalStr Function 631
- EvalStrInfo function 632
- Evaluate condition 337
- Evaluate Expression Operation keywords 1365
- Evaluate operation 483
- Event handler 815
- Event menu 1018
- Event types 283
- Execute APG rights 1042
- Execute mode
 - program generator 1224
- Execution level 408
- Execution levels 408
 - event 409
 - name 408
- Exit operation 503
 - keywords 1368
- Exit URL 341
- EXP function 634
- EXP literal 512
- Expand form 794
- ExpCalc function 634
- Export an application 1245
- Export data
 - program generator 1225

- Export-import utility 1245
 - application component types 1247
 - export application components 1247
 - export repository entry with folder 1249
 - export with models 1248
 - file name for exported components 1250
 - import folder structure 1250
 - range of export 1249
 - self-documenting report 1247
- Expression evaluated events 283
- Expression events 409
- Expression Rules repository 366
 - keywords 1347
- Expressions 367
- External event 1197
- F**
- Feature license 1271
- Field attribute 772, 777
- Field delimiting method 498
- Field Location Repository keywords 1354
- Field model 203
- Field model properties 226
- File handles 82
- FILE literal 512
- File2Blb function 634
- FILE2OLE function 635
- File2OLE function 635
- FILE2REQ function 635
- File2Req function 635
- FileDLG function 635
- FileListGet function 636
- Fill function 636
- Filter 146
- Fit to MDI 796
- Fix function 636
- Flat MCF 1062
- Flip function 637
- Float picture 152
- Floating palettes 100
- Floating window 792
- Flow function 637
- Flow monitor 83, 1250
- Flow monitor commands 1251
- Flow monitor filters 1253
- Flow monitor properties 1254
- Flow monitor utility for the browser client 1257
- Flow monitor utility for the server 1255
- Flow Monitor, Remote 1257
- FlwMtr function 638
- FlwMtrVars function 638
- Folders 1249
- Font
 - assignment window 133
 - definition file 66, 106
 - HTML controls 937
 - name 132
 - orientation 132
 - property 795, 1029
 - repository 131
 - saving changes 133
 - size 132
 - style 132

- style window 132
- Force MVCS Disabled 60
- Force MVCS key 68, 1049
- Force record delete 362
- Force record suffix 362
- Foreground colors 127
- Foreground generator 123
- Foreground window 360
- Foreign key definition 274
- Foreign key for the generated program 1226
- Foreign key repository 273
 - keywords 1320
- Foreign key segment repository
 - keywords 1321
- Foreign key segments 275
- Foreign key, definition 248
- Form
 - area 372
 - class numbers 371
 - display block keywords 1354
 - footer 372
 - header 372
 - interface type 372
 - keyboard shortcuts 376
 - model 201
 - name 371
 - repository 369
 - repository keywords 1348
 - templates 376
 - units 375
- Form area 793
- FORM literal 513
- Form model

- properties 237
- Form records 361
- Forms
 - batch tasks 370
 - browser 765
 - display form 791
 - frame set form 945
 - HTML form 921
 - HTML merge form 950
 - online tasks 370
- Frame height 798
- Frame keyword parameter 1369
- Frame set command palette 947
- Frame set forms 945
- Frame width 797
- Full Where clause 404
- Functional directives 210, 217
- Functions 517

G

- Generate mode
 - program generator 1224
- Generator context management 123
- Get Definition utility 1235
- GetLang function 638
- GETPARAM function 1220
- GetParam function 639
- Global keywords 1315, 1369
- Global parameters 1296
- Global temporary tables 1469
- Graphic image control 806, 811
- Graphic line control 805, 811
- Group check messages 102
- Group control 805

- Group handler level 287, 312
- Group Prefix 280
- Group Suffix 282
- GroupAdd function 640
- GUI display color palette 804
- GUI display commands 798
- GUI display controls 804
- GUI display forms 791
- GUI display variables palette 824
- GUI printing of a table control 966
- GUI table controls
 - set table size 966
 - title on every page 966

H

- Handler operation level 285, 288
- Handler repository 408
- Handlers 287
- Header file name 922
- HEB literal 513
- Help action support 1190
- Help display block keywords 1326
- Help file 105, 1062
- Help key 1063
- Help model 201
- Help model properties 241
- Help prompt 228, 776
- Help screen 769, 774, 776
- Help screen repository 1026
 - keywords 1324
- Help Screens repository rights assignment 1044
- Help types
 - browser-based help screens 1027

- internal help screens 1027
- prompt help screens 1027
- Tooltip helps 1027
- Windows help 1027
- Help/About dialog 93
- Hidden fields 927
- Hidden variable 939
- Hide a menu entry or toolbar button 1025
- Hint string 1457, 1466
- Hinting the optimizer 170, 272
- HitZOrdr function 641
- Horizontal slider control 805, 809
- Hour function 641
- HStr function 641
- HTML command palette 929
- HTML control palette 935
- HTML control placement 921
- HTML control repository 770
- HTML forms 921
- HTML frame set forms 945
- HTML Help 1032
- HTML internal attribute 941
- HTML merge command table 955
- HTML merge forms 950
- HTML merge syntax rules 961
- HTML merge tags 959
- HTML static table command palette 932
- HTML style repository 179, 924
- HTML styles file 66, 107
- HTML template file 112, 951, 958
- HTTP Authentication 1056
- HTTP proxy 112

- HTTP requester 118
- HTTP timeout 112
- HTTPGet Function 642
- HTTPLASTHEADER function 643
- HTTPLastHeader function 643
- HTTPPost function 644
- HVal function 645
- Hyperlink 938
 - property 922, 939, 950, 956
 - to a Magic program 925
 - to a URL 926

I

- I/O file identification number 497
- I/O File repository 383, 385
- I/O Filer repository
 - keywords 1343
- I/O files
 - access 387
 - character sets 390
 - expressions 388
 - flip line 391
 - formatting 387
 - media 384
 - name 384
 - open I/O from another program 390
 - page footer form 389
 - page header form 389
 - page orientation 390
 - page size by row 388
 - page size for a graphic printer 389
 - Print dialog 388
 - print preview 390
 - printer 385
 - properties 389
 - visual to logical 391
- I/O functions 542
- Icon file name 342
- Identity column 1467
- Idle function 645
- IF function 645
- Ignore error handler 1007
- Image cache size 101
- Image control 787, 806
- Image push button 810
- Images for a menu entry or toolbar 1024
- Images for a toolbar button 1024
- Import and export rights 1050
- Import data
 - program generator 1225
- Importing application 1245
- IN function 646
- Incremental locate 1450
- Indent character 97
- Index 248, 339, 451
 - ascending order 269
 - definition and usage 1444
 - descending order 269
 - non-unique 268
 - properties 270
 - repository 266
 - repository keywords 1319
 - segment repository 268
 - segments repository keywords 1320
 - unique type 268

- Informix database gateway 1473
 - Informix data types 1476
 - Magic data types 1473
 - text and byte data types 1478
- Inheriting property values 199, 201
- INIGet function 646
- INIGetLn function 647
- INIPUT function 1064
- INIPut function 648
- Init 425
- Inner Join link operation 445
- Input fields 951
- Input form 923
- Input Form operation 497
- Input Form Operation keywords 1367
- INS function 649
- Insertion point park 305
- Installation 35
- Instantiation 227
- INSTR function 649
- Integration functions 534
- Interface functions 541
- Internal codes for DBMS 153
- Internal events 409
- Internal help screens 1028
- Internal Magic events 283
- Internal transactions 990
- Internet client forms 765
- Internet data entry program
 - program generator 1225
- Internet development file root 67
- InTrans function 650
- IO Device Open Timing 99
- IOCopy function 650
- IOCurr function 650
- IODel function 651
- IOExist function 651
- IORen function 651
- IOSize function 652
- ISAM
 - databases 991
 - Force locking within transaction 87
 - transactions 85
- IsComponent function 652
- IsDefault function 652
- IsFirstRecordCycle function 653
- ISNULL function 653
- Isolation level 156

J

- J2EE
 - Connection Difficulties 1396
 - Creating the JAR file 1389
 - Defining the EJB 1383
 - EJB Component Builder 1385
 - Generic Messaging Layer 1393
 - Magic Installation 1396
 - Terminology 1382
 - The Component Builder 1383
- J2EE enterprise servers 1076, 1388
- Java 939
- JAVA_HOME 125
- JCall function 654
- JCallStatic function 654
- JCreate function 654
- JEXCEPTION function 655
- JException function 655
- JExceptionOccurred function 655

- JExceptionText function 655
- JExplore function 655
- JGET function 655
- JGetStatic function 656
- JInstanceOf function 656
- JSET function 656
- JSet function 656
- JSetStatic function 656

K

- KBD literal 513
- KbGet function 657
- KbPut function 657
- Keep new context 344
- Key assigned to action 138
- Key combinations 37
- KEY literal 513
- Key to identify a right 1037
- Keyboard idle seconds 90
- Keyboard mapping file 66, 106
- Keyboard mapping for forms 376
- Keywords 1313

L

- Language at startup 111
- Languages
 - deployment 176
 - multi-lingual support (MLS) 174
 - name 175
 - repository 174
 - translation file 175
- LastPark function 658
- LDAP connection string 114
- LDAP domain contexts 115

- LDAP timeout 115
- LDAPError function 658
- LDAPGet function 658
- LEFT function 659
- Len function 659
- Level Definition repository 407
 - keywords 1360
- Level function 660
- Level repository 280, 288
- License environment setting 82
- License file location 83
- LIKE function 661
- Line control 805
- Line function 662
- Line mode display 1227
- ling 1198
- List box control 783
- Literals 511
- LMChkIn function 662
- LMChkOut function 662
- LMUVStr function 663
- LMVStr function 663
- Load components 1062
- Load flow monitor 83
- Load resident tables 94
- Load tables 1235
- Local COM runtime behavior 1095
- Local keywords 1315
- Local temporary tables 1469
- Local Variable Repository keywords 1341
- Local variables
 - properties 365
 - repository 364

- Lock 451
- Lock file 87, 1416, 1424
- Lock files path 1418
- Lock function 664
- Lock path 168
- Locking 1436, 1489
 - enforcing locks 1437
 - escalating a lock to a table lock 1437
 - locking duration 1438
 - locking levels 1436
 - physical and logical locks 1440
 - table access and share mode 1439
 - transaction processing 1438
- LOG function 664
- Log level 154
- LOG literal 513
- Logic pictures 220
- Logical attribute 205
- Logical function 664
- Logical Name repository 172
- Logical names
 - syntax 173
 - translation 173
 - usage 173
- Logical operators 515
- Logo file 105
- Logon 71, 181
 - date 183
 - password 182
 - Setting 183
- Logon function 665
- LONG and LONG RAW 1457
- LoopCounter function 666
- Lower function 666
- LTrim function 666

M

- Magic actions 134
- Magic Flat File (MFF) 60, 1273
- Magic gateway name structure 1431
- Magic gateways 1431
- Magic monitor application 1219
- Magic path string 250
- Magic profiler utility 1266
 - output files 1267
 - UNIX variable 1266
 - VMS variable 1266
 - Windows variable 1266
- Magic Request Broker
 - automatic reload 1169
 - automatic termination of enterprise server 1169
 - installation 1162
 - load balancing 1169
 - queue 1168
- Magic SQL Where 405
- MAGIC.INI file 93, 183, 185, 190
- MAGIC_JAVA section 125
- Mail connection timeout 113
- Mail operation timeout 113
- MailBoxSet function 666
- MailConnect Function 667
- MailDisconnect Function 668
- MailError Function 668
- MailFileSave Function 668
- MailLastRC function 669
- MailMsgBCC function 669

MailMsgCC Function 669	Menu authorization options 1021
MailMsgDate function 669	Menu check marks 1025
MailMsgDel Function 670	Menu Definition repository 1017
MailMsgFile Function 670	Menu definition rights assignment 1045
MailMsgFiles Function 670	Menu entry enable 1025
MailMsgFrom Function 670	Menu entry hide 1025
MailMsgHeader function 671	Menu formats 1014
MailMsgId Function 671	Menu function 674
MailMsgReplyTo function 671	Menu help 1023
MailMsgSubj function 671	Menu image 1024
MailMsgText function 672	Menu name 1017
MailMsgTo function 672	Menu prompt 1023
MailSend function 672	Menu properties 1021
Main display form 341	Menu repository 1016
Main program 411, 1065	Menu rights 1022
Main table 339, 1229	Menu structures 1016
MakeKey utility 1271	Menu type 1017, 1018
Mapping transactions 990	Messaging server 117
Mask characters 224	MG_CANCEL 1192
Mathematical functions 542	MG_EDIT 1191
Mathematical operators 515	MG_EXIT 1192
MAX function 673	MG_VCR 1191
Maximize button 793	MGExternalEvent function 1198
Maximum connections 155	MGPROFM.LOG file 1270
Maximum file handles 82	MGPROFT.LOG file 1269
MAXMagic function 674	MGRB.INI file 1162
MDate function 674	MGROGNRC.DLL 1162
MDI client edge 38	MGRQMRB.EXE 1162
Media 384	Microsoft SNMP agent 1380
Memo attribute 206	MID function 675
Memo pictures 217	MIN function 675
Menu access key 1020	Minimize button 793
Menu access rights for the end-user 1045	MINMagic function 675

- Minute function 676
- MIsTrans function 676
- MMClear function 676
- MMCount function 677
- MMCurr function 677
- MMStop function 677
- MnuCheck function 678
- MnuEnabl function 678
- MnuName function 678
- MnuShow function 679
- Modal window 769, 792
- MODE literal 514
- Mode of operation 334
 - creating new records 301
 - deleting records 302
 - modifying records 301
 - querying records 302
- Model repository 200
 - keywords 1315
- Model repository rights assignments 1040
- Models
 - break inheritance 201, 203, 226
 - breaking model properties 243
 - control 231
 - creating 242
 - deleting 243
 - expressions 244
 - field 203
 - form 201
 - help 201
 - properties 201
 - removing a model from an object 243
 - rights 244
 - selecting a different model 243
 - set inheritance 199, 203, 226
- Modifying rights 1041, 1042, 1043, 1044, 1045, 1047
- Monitor Output file 83
- Monitor utility 1300
- Month function 679
- MStr function 679
- MtblGet function 680
- MtblSet function 680
- mTime function 682
- mTStr function 682
- mTVal function 683
- Multi-lingual Support 174
- Multi-marking 814
- Multi-page printing of an edit control 967
- Multi-threading 1216
 - calling external programs 1218
 - environment settings 1216
 - runtime expressions 1218
- Multi-user access 85, 381, 1415
- MVal function 683
- MVCS disabled 60
- MVCS key 68
- MVCS lock files path 65
- MVCS snapshot level 65

N

- Namespace support 1134
- NDOW function 683
- Nested deferred Transaction mode 978

- Nested transaction 985
- Net-SNMP agent 1380
- Network traffic 1449
- New application 55
- NMonth function 684
- Non-abortable update 486
- Non-procedural operations 298
- Non-repeatable reads 156
- NOT operator 516
- NULL arithmetic 64
- NULL function 684
- Null value 1443
- NULL Value Default keywords 1332
- Nulls 152, 229
- Numeric attribute 205
- Numeric field updates 983
- Numeric functions 545
- Numeric pictures 211, 219
- NumSetBuf function 570

O

- Object name 227
- ODBC Check Driver utility 1271, 1490
- ODBC database gateway 1484
 - database default values 1489
 - Direct SQL 1490
 - Magic data types 1484
 - ODBC data types 1487
 - Sort temporary database 1489
 - troubleshooting 1489
- OEM_ANSI 109
- OEM_ANSI utility 1270
- OEM2ANSI function 684
- OLE automation 1143

- OLE control 806, 811
- OLE data field 207
- One-to-many relation 430
- Online task type 334
- Online Tasks
 - deleting records 309
 - record/row loop 314
- Open files trace file 1269
- Operation Repository keywords 1361
- Operation time interval 99
- Operators
 - logical 515
 - mathematical 515
 - string 516
- OR operator 516
- Oracle database gateway 1453
 - connect string 166
 - Magic data types 1453
 - Oracle data types 1456
- OS Command menu 1018
- OSEnvGet function 685
- OSEnvSet function 685
- Output Form operation 492
 - keywords 1366
- Output title for HTML form 1228
- Owner function 685
- Owner name 71

P

- Page footer form 389
- Page function 686
- Page header form 389
- Page orientation 390
- Page size 389

- Palette 794, 923
- Palettes 100
- Parameter List keywords 1365
- Parameter type string 1144
- Parameter variables 204
- ParamsPack function 686
- ParamsUnpack function 686
- Parent and child links 802
- Password 73, 145, 165, 182, 1051
- Phantom task 468
- Phantoms 156
- Physical locking 1466, 1478, 1482
- Physical transaction mode 978
- Physical transactions for group level 980
- Physical transactions for record level
 - Before Record Prefix 980
 - Before Record Suffix 981
 - Before Record Update 981
 - On Record Lock 980
- Pick list 340
- Pick list windows 465
- Pictures 208
 - components 209
 - date 214, 220
 - functional directives 210, 217
 - logical 220
 - mask characters 224
 - memo 217
 - numeric 211, 219
 - positional directive 218
 - positional directives 222
 - special characters 222
 - syntax rules 225
 - time 221
 - usage 208
- Placement property 797
- Popup windows 500
- Port an application 1245
- Positional directives 218, 222
- PPD function 687
- Pref function 687
- Preset tool images 1191
- Primary keys 273, 274
- Print Attribute repository 180, 970
- Print attributes file 66, 107
- Print attributes for runtime resolution 974
- Print color attribute 972
- Print data 1274
- Print dialog 388
- Print multi-page edit control 967
- Print preview 390
- Print styles 970
- Printer 385
- Printer attribute support 968
- Printer commands 970
- Printer settings 969
- Printers
 - commands file 178
 - name 178
 - page size 179
 - queue 178
 - repository 177
 - translation file 178
- Printing a data file
 - program generator 1225
- Printing data 359

- Printing of a table control 966
- Prog function 687
- PROG literal 514
- ProgIdx function 688
- Program execution trace file 1267
- Program generator 1223
- Program menus 1018
- Program repository 331
- Program Repository keywords 1334
- Program repository rights assignment 1042
- Program rights assignment
 - execute the Automatic Program Generator 1043
- Programmable Protection Device (PPD) 68
- Prompt help screens 1029
- Property sheet automatic handling 101
- Public name 332
- Public rights 1037
- Public rights access key 67, 1048
- Pulldown menu close timeout 90
- Pulldown menu keywords 1328
- Pulldown menus 1015
- Push button control 778, 804, 810

Q

- Query mode 304
- Query mode locate warning 77
- Query rights 1040, 1042, 1043, 1044, 1045, 1047
- Query title for internet form 1228

R

- Radio control 775, 805
- Raise Event operation 505
- Raise Event operationkeywords 1368
- RAND function 688
- Range and locate 398
- Range and Locate keywords 1338
- Range definition 1447
- Range function 688
- Range of record 1009
- Range/Locate box popup seconds 89
- Read - Transaction 990
- Real columns 204
- Record initialization level 308
- Record locking 168
- Record Main 281
- Record Prefix 281
- Record Suffix 282
- Record termination level 308
- Record update fail before call 985
- Record/row loop 308
- Recover error strategy 998
- Rectangle control 805
- Referenced table 274
- Remote COM runtime behavior 1097
- Remote flow monitor 1257
- Remote flow monitor activation 84
- Remote flow monitor port 84
- Remote flow monitor rights 68
- Remote host 228
- Rep function 689
- Report forms 966
- Reposition after modify 96
- Repositories, working with 42

- command options 43
- entering text 42
- moving the insertion point 43
- RepStr function 689
- REQQUELST function 1220
- Requester timeout 118
- Requirements for team development 1422
- Resident MAGIC.INI 93
- Resident tables 94, 252, 256
- Resident task 341
- ResMagic function 690
- Return value 339
- Rich edit control 806, 812
- Rich text control 806
- Rich text format control 811
- Right function 690
- Right keys allocated by user 1052
- RIGHT literal 514
- RightAdd function 690
- Rights assignment
 - accessing 1039
 - application properties 1047
 - authorization 1039
 - help screens 1044
 - menu definitions 1045
 - models 1040
 - programs 1042
 - tables 1041
- Rights for menus 1021
- Rights function 691
- Rights keys allocated by group 1053
- Rights limited by user 1036
- Rights repository 1036
- keywords 1326
- Rollback 992, 1001
- ROLLBACK function 691
- Rollback function 691
- Round function 692
- Row highlighting 788
- RqCtxInf function 693
- RqCtxTrm function 693
- RQEXE function 1220
- RqExe function 694
- RqHTTPHeader function 694
- RQLOAD function 1220
- RqLoad function 695
- RQQUEDEL function 1220
- RqQueDel function 696
- RqQueLst function 696
- RQQUEPRI function 1220
- RqQuePri function 697
- RQREQINF function 1220
- RqReqInf function 697
- RqReqLst function 698
- RQREQQLSTfunction 1221
- RQRTAPP function 1221
- RqRtApp function 699
- RQRTAPPS function 1221
- RqRtApps function 699
- RqRtCtx function 703
- RqRtCtxs function 701
- RQRTINF function 1221
- RqRtInf function 702
- RQRTS function 1221
- RqRts function 703
- RqRtTrm function 704
- RQRTTRM function 1221

RqRtTrm function 703
RqRtTrmex 704
RQSTAT function 1221
RqStat function 705
RqTrmTimeout function 704
RTrim function 705
RunMode function 705
Runtime menu access rights 1045

S

Screen mode display 1227
Screen mode prompt 74
Second function 706
Secret Name repository 180, 1054
Security file 108
Security functions 546
Segment repository 392
Segments
 size 392
 sort direction 393
 sort type 393
 sort using RDBMS 393
 variable name 392
 variable type 392
Select Operation keywords 1361
Selection control 804
Server
 address 142
 communication interval 86
 name 141
 properties 142
 repository 140
Service context management for batch tasks 344

Services
 properties 145
 repository 144
SetBufCnvParam function 706
SetCrsr function 707
SetLang function 708
SETPARAM function 1221
SetParam function 708
Share mode 1411
SharedValGet function 709
SharedValPack function 709
SharedValSet function 709
SharedValUnpack function 710
Sharing in a multi-user environment 381
Sharing tasks 381
Short/Long keyword parameter 1314
Shortcut keys 1020
Show grid 792, 922
Show plan 155
Simulating key entries 657
SIN function 710
Single common context 123
Single expand palettes 100
Sizing forms 797
Slider control 805
Snapshot file 1418, 1423
SNMPNotify function 710, 1374
SOAP header 1074
Sort direction 393
Sort repository 391
 keywords 1345
Sort using RDBMS 393
Sort/Temp box popup seconds 90

- Sorting 1448
- SoundX function 711
- Split window child 793
- SplitterOffset function 711
- SQL
 - array size 170
 - command keywords 1336, 1337, 1338
 - database information 169
 - databases 991
 - gateways 1451
 - hinting the optimizer 170
 - Where clause keywords 1339
 - XA transactions 171
- SQL table properties
 - array size 256
 - check the existence of a table 254
 - cursor value 255
 - default position 254
 - default position index 254
 - hinting the optimizer 255
 - owner of the table 254
 - physical locking 1466
 - pre-defined DB table or view 255
 - selecting a unique index 254
 - supplying database-dependent information 253
 - text data type 1466
- SQL Where 401
- SSL CA certificates 115
- SSL client certificate file 116
- SSL client certificate password 116
- Startup language 111
- Startup mode 64
- Startup position 796
- Startup security file 108
- Stat function 711
- Static controls
 - group control 807
 - static ellipse control 807
 - static rectangle control 807
 - static text control 807
- Static text control 804
- Station lock file 1428
- Storage field models 208
- Stored procedures 1448
- Str function 712
- String functions 547
- String operators 516
- String time attribute mapping 1450
- StrToken function 712
- StrTokenCnt function 713
- StrTokenIdx function 713
- Subforms 766
- Sub-Object name 227
- Super right key 68, 1049
- Supervisor entry 1036
- Supervisor group 1054
- Synchronization process for team development 1426
- Syntax rules for constructing pictures 225
- Sys function 714
- System events 283, 409
- System logon 71
- System menu 793
- System model 202

T

- Tab control 805
- Table control 806, 813, 966
- Table Conversion utility 257, 1271
- Table locking 1459, 1477
- Table modes 1410
- Table properties 251
- Table repository 246
 - keywords 1315
- Table repository rights assignment 1041
- Table rights assignment
 - using the Automatic Program Generator 1042
- Table sharing interaction 1413
- Tables
 - changing table structure 257
- Tag table 951
- TAN function 714
- Task
 - cache buffer size 76
 - cycle levels 307
 - flow modification 92
- Task control keywords 1339
- Task control properties 357
- Task dataview 299
- Task Event Parameter Repository keywords 1346
- Task Event Repository keywords 1345
- Task Execution repository 280, 406
- Task flow 301
- Task functions 549
- Task levels 408
- Task Prefix 280
- Task Property keywords 1335
- Task Suffix 282
- Task window 359, 360
- Tasks 277, 332
 - properties 333
- TDepth function 714
- Team development 65, 1422
- Temporary tables 1468
- Temporary tables path 82
- Term function 715
- Terminal identifier 84, 1416
- Text control 776
- Text function 715
- THIS function 715
- Thousands separator 104
- Time attribute 206
- Time function 716
- Time functions 529
- TIME literal 514
- Time pictures 221
- Time separator 104
- Timeout 149
- Timer events 283, 409
- Title on every page 966
- Token prefix 951
- Token suffix 951
- Toolbar button enable 1025
- Toolbar button groups 1025
- Toolbar button hide 1025
- Toolbar button image 1024
- Toolkit checker minimal level 102
- Tools infrastructure 1284
- Tooltip 98, 774, 776, 780, 1025, 1032
- Top keyword parameter 1369

- Transaction begin 978
- Transaction mechanism 1435
- Transaction mode 977
 - deferred 977
 - nested deferred 978
 - physical 978
 - within active transaction 978
- Transaction processing 977
- Transaction trees 987
- Transactions 1435
- Translation 173
- Translation file 175, 178
- TransMode function 717
- Tree control 806, 815
- TreeLevel function 717
- TreeNodeGoto function 717
- TreeValue function 717
- Trim function 718
- TStr function 718
- TVal function 718
- Type library 227
- Typographical conventions 36

U

- UDF function 719
- UDFF function 719
- UDFS function 720
- Uncheck object 1421
- Units of measurement 375, 792
- UNIX operating systems 1430
- UNIX web server 964
- Unlock function 720
- Update operation 486
 - keywords 1365

- Update/Delete statements 984
- Upload 965
- Upper function 721
- URL help screens 1033
- User Event repository 292
- User events 409
- User function 721
- User Group repository 181
- User groups 1052, 1053
- User ID repository 181, 1051
- User identification 73, 182
- User name 145, 165
- User retry 1004
- UserAdd function 722
- User-defined events 292, 394
- User-defined menus 1017
- UserDel function 722
- UTF8FromAnsi function 722
- UTF8ToAnsi function 723

V

- Val function 723
- Validate link operation 443
- VAR literal 514
- VarAttr function 724
- VarCurr function 724
- VarCurrN function 724
- VarDbName function 725
- Variable functions 552
- Variable MCF record length 158
- Variables 516
- Variables palette 824
- VariantAttr function 725
- VariantCreate function 726

- VariantGet function 728
- VariantType function 729
- VarIndex function 732
- VarInp function 732
- VarMod function 733
- VarName function 733
- VarPic function 734
- VarPrev function 734
- VarSet function 735
- VecCellAttr function 736
- VecGet function 736
- VecSet function 736
- VecSize function 737
- Vector data 518
- Verify operation
 - keywords 1362
- Vertical slider control 805, 809
- ViewMod function 737
- Views 1468, 1482, 1489
- Views and fragmented tables 1477
- Virtual variables 204
- Visual Connection screen 146
- Visual function 737

W

- Wallpaper 769, 795, 923, 944, 1228
- Web authoring tool 119
- Web document alias 118
- Web document path 118
- Web online event handlers 952
- Web online page 964
- Web online response 964
- Web service interface builder 1070
- WebRef function 738

- WINBox function 738
- Windows help key 1031
- Windows help screens 1030
- Windows operating systems 1430
- Windows submenu 1025
- WinHelp function 739
- WINHWND function 740
- Within active transaction 978
- Workgroup development (MVCS) 1421
- Wrap keyword parameter 1313
- Write - Transaction 990
- Write link operation 444
- WSAttachmentAdd function 740
- WSAttachmentGet function 741
- WSDL file 1070
- WSDL file path 1073
- WSDL files path 113
- WSDL name space 1073
- WSDL service end point 1073

X

- XA transactions 171
- XCG
 - Component structure 1130
 - Generation 1132
 - Using 1121
- XML Component Generator 1121
- XML schema 1124
- XML template file 113
- XMLBlobGet function 742
- XMLCnt function 743
- XMLDelete function 745
- XMLExist function 747
- XMLFind function 749

XMLGet function 751
XMLGetAlias function 753
XMLGetEncoding function 754
XMLInsert function 754
XMLModify function 759
XMLSetEncoding function 761
XMLSetNS function 762
XMLStr function 763

XMLVal function 763
XSD 1125

Y

Year function 763

Z

Z-order of controls 801