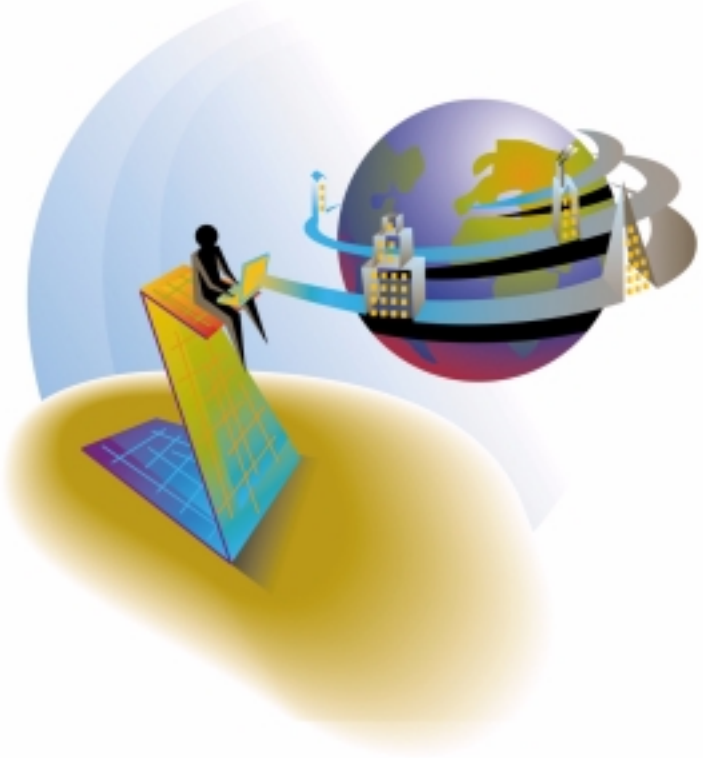
eDeveloper V9.4



How to...

Working with eDeveloper

MAGIC
www.magicsoftware.com
Enabling Business with Superior Technology

Revised February 2004

04 2 1 12 11 10 9 8 7 6 5

4190-07000

Contents

2 Creating Data Objects

3 Creating the Task Dataview

4 Using Basic Programming Techniques

5 Building an Online Task

6 Building an Interactive Web Task

7 Using Transaction and Recovery Techniques

8 Securing Your Application

9 Using Advanced Programming Techniques

10 Printing With eDeveloper

11 Defining Application Menus

12 Creating Application Helps

13 Using eDeveloper Toolkit Utilities

14 Using eDeveloper Components

15 Partitioning eDeveloper

16 Connecting to External Applications

17 Improving Performance

18 Deploying eDeveloper

19 Building a Batch Task

20 Integrating With the J2EE Environment

21 Using COM Support

22 Sending and Receiving Data

23 Sending and Receiving Messages

24 Using Drag-and-Drop Functionality

Introduction

This *How To* . . . book is a list of answers to questions that might arise during a developer's programming cycle and during the deployment phase.

The book covers all aspects of writing code with eDeveloper, starting from configuration, continuing on to infrastructure issues, and relating to all types of available programs. It mainly deals with application dilemmas, questions, and needs raised by programmers.

The *How To* .. . book is intended mainly for programmers, new and veteran alike. No prior knowledge of previous versions is assumed. To understand the topics in this guide, you must have some basic knowledge of how to use eDeveloper as a development tool.

The guide gives answers to specific questions raised by the programmer and application deployer. The book makes no attempt to deal with development standards, such as how to divide your application into components, and does not supply code examples.

The *How To* .. . book is not limited to any one version, or to features that belong to a specific version. It tackles questions that one might have arisen in previous versions as well. The answers, however, are based on the current capabilities of eDeveloper.

The questions raised in this guide will be continually updated based on input from programmers in the field. Programmers who know of questions that are left unanswered are more than welcome to direct these questions to MSE, at **How2@magicsoftware.com**.

Configuring the eDeveloper Environment

1

You can tell eDeveloper to change default settings for the Toolkit and Runtime environments using options on the **Settings menu** or directly in eDeveloper's configuration files. You can change the names of most of the configuration files in the **Environment dialog**, and you can also use other settings to directly edit the content of the configuration files.

This chapter covers the topics listed below:

- Starting a New eDeveloper Session
- Changing eDeveloper Settings for Each Session
- Defining an SQL Database
- Changing the Application Default File Name
- Setting up eDeveloper Language Support
- Creating Color Combinations
- Creating Font Definitions
- Displaying the eDeveloper Executable File Version
- Defining Reusable Application Objects
- Using and Breaking Inheritance
- Defining Logical Names
- Defining Default Date Values and NULL Display Strings
- Defining an External Configuration File for a Specific Application

Starting a New eDeveloper Session

After installing eDeveloper, there are two executable files in the eDeveloper directory. The first file, Mgrntw.exe, is for the deployment environment, and the second, Mggenw.exe, is for the development environment. Double-click Mggenw.exe to open the eDeveloper toolkit environment.

This section includes the topics listed below:

- Creating a New Application
- Defining Application Properties

Creating a New Application

To create a new application:

1. On the **Settings** menu, click **Applications**. The Application repository opens.
2. Enter the application's name. The name can be up to 30 characters.
3. You must enter a two-letter prefix for the application. You can add a path before the prefix to save the application file in a specific directory.
4. In the **Application File** column, you select a specific path and control file name. Zoom from this column to select a specific file or path. This property is optional.
5. Zoom from the **Database** column to open a selection window that shows all the databases defined in the **Database** repository. Select the database where you want to store your application control file. Click **OK**.
6. From the **File** menu, click **Open** to access your application.

Defining Application Properties

To define application properties in **Startup** mode, place the cursor on an application line in the **Application** repository and press **CTRL+P**.

To define application properties in **Toolkit** mode, select **Application Properties** on the **File** menu to open the **Application Properties** dialog box and define the properties.

Changing eDeveloper Settings for Each Session

To change the eDeveloper settings of a single session, you can use:

- eDeveloper command line options - When starting eDeveloper from a command line or a Windows shortcut, you can override settings in the Magic.ini file that only affect the current eDeveloper session.
- The INIPut function - The INIPut command modifies environment settings at runtime, and enables you to modify eDeveloper settings from a running application.
- Authorization functions - The eDeveloper authorization system can be modified at runtime. You can modify the security settings (usr_std) from an application that is running. The authorization functions that modify security settings are:
    - UserAdd (adds a user)
    - GroupAdd (adds a user to a group)
    - RightAdd (assigns a right to a user).

Defining an SQL Database

To define an SQL database:

1. eDeveloper works with many SQL databases. eDeveloper can work with an SQL database on an open client or client server.
2. Working with different SQL databases is similar. For an example of how to define open client connectivity using Oracle Version 8i, follow the steps listed below:

This section includes the topics listed below:

- Configuring an Open Client Environment
- Setting up eDeveloper to Work with an SQL Database
- Setting the eDeveloper Environment to Work with SQL Databases
- Checking for Connectivity Between eDeveloper and the SQL Database

Configuring an Open Client Environment

To configure an open client environment, you need:

- eDeveloper
- A Magic SQL gateway for a specific database. For example, you need the Mgora8.dll file for an Oracle database.
- Database. For example, when you use Oracle you need to install the Oracle open client software on the client computer.
- Database server

Setting up eDeveloper to Work with an SQL Database

To set up eDeveloper to work with an SQL database:

1. Check that the **Mgora8.dll** file is in your eDeveloper directory.
2. Edit the **Magic.ini** file. In the **[MAGIC_GATEWAYS]** section, remove the semicolon before **MGDB13**.
3. Run eDeveloper.
4. On the **Help** menu click **About Magic** and under **Loaded modules** check that the Oracle gateway appears. If not, check that the line you edited in the **Magic.ini** file points to the **Mgora8.dll** file.

Setting the eDeveloper Environment to Work with SQL Databases

To set the eDeveloper environment for SQL databases:

1. On the **Settings** menu, click **Databases** to open the **Databases** repository.
2. Create a new line.
3. Enter a name for the entry.
4. Zoom from the **DBMS** column, and select a DBMS system from the **DBMS** list. For example, select Oracle.
5. Press **CTRL+P** to open the **Database Properties** sheet.
6. On the **Login** tab, enter the database server. This is your Oracle alias. Enter your user name and password.
7. Click **OK**.

Checking for Connectivity Between eDeveloper and the SQL Database

To check the connectivity between eDeveloper and the SQL database:

1. From the **Table** repository, park the cursor on the header line.
2. From the **Options** menu, click **Get Definition**. If eDeveloper cannot work with the database, an error message appears.

Changing the Application Default File Name

When creating a new application, eDeveloper assigns a name to the control file. The default file name is `<prefix>CTL.MCF`. To change the application default file name, you first need to select a name for the application file:

To change the application default file name:

1. From the **Settings** menu, click **Applications** to open the **Application** repository.
2. Enter the file name in the **Application File** column. Specify a full path name
   -OR-
   zoom to select a specific file.

Setting up eDeveloper Language Support

eDeveloper's multi-lingual support (MLS) allows you to develop an eDeveloper application in one language and deploy it in another language without the need to modify the application file for each language. For example, you can write an application that displays text strings in English and also deploy it in French and German. eDeveloper multi-lingual support translates text strings based on a language-specific translation file that you created.

This feature works only for left-to-right languages.

This section includes the topics listed below:

- Building an eDeveloper Language File
- Setting a Starting Language

Building an eDeveloper Language File

eDeveloper needs a translation file that includes the original string, that is the string as entered when developing the application, and the translated string. For each language, you need a separate translation file.

The format of the translation file, a text file, is:

```
OriginalString1
TranslatedString1
OriginalString2
TranslatedString2
OriginalString3
TranslatedString3
```

Each text string must be on a separate line. A blank line must be left at the end of the file.

The translation file must be converted to an eDeveloper language file using the MLS_BLD.EXE utility (on Windows platform from a DOS windows).

The two parameters to this utility are the translation file (input) and the language file (output): **`MLS_BLD <translation file> <language file>`**

Setting a Starting Language

To set a starting language:

1. On the **Settings** menu, click **Languages** to open the **Language** repository.
2. Enter a name and full path location of the eDeveloper language file.

3. On the **Settings** menu, click **Environment** to open the **Environment** dialog box, and click the **External** tab.
4. Zoom from the **Starting language** setting and select the required language. If no language is selected, the default language is used. This setting determines the language display in Runtime and Toolkit modes.
5. To set the language dynamically, use the **SetLang** function. Use the **GetLang** function at runtime to check which language is selected.

MLS only affects the display of text strings. The actual data value used to set the active selection remains as defined in the application, regardless of the displayed translation string.

Creating Color Combinations

The Color repository specifies the foreground and background colors for 114 entries that represent pre-set and user-defined, or reserved, color assignments and Operation repository colors.

In the Color repository, you can change and save both foreground and background colors.

This section includes the topics listed below:

- Changing Color Combinations
- Changing Foreground and Background Colors

Changing Color Combinations

The Color repository assigns a foreground (FG) color and a background (BG) color for each display entity, one per row.

Rows 1- 79 are for user- defined colors.

Rows 80- 99 are used for system definitions and for reserved uses.

Rows 100 to 111 are used for Operation Repository colors.

Rows 80- 111 should not be changed. The names of colors 100- 111 cannot be modified, but the FG and BG colors can be changed. The Color repository can store an unlimited number of color values.

Changing Foreground and Background Colors

To change foreground or background colors:

1. Place the cursor on the **Foreground (FG)** or **Background (BG)** column, and zoom to the **Color Assignment** palette to select a color.
2. The name of the current row of the Color repository appears on the Title Bar of the **Color Assignment** palette. Select a color. The selected color displays a 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). You can specify background and foreground colors for different system items using the System list on the Color Assignment Palette.
3. Click **OK** to accept the new color assignment and close the **Color Assignment** palette.
4. Click **Cancel** to cancel all color assignment changes in the current entry, and close the **Color Assignment** Palette.
5. From the **Color** repository, click **OK** to accept the changes and end the color editing session. The color settings are saved in a special file that eDeveloper uses in the Environment settings. The default name for this file is **Clr_ std.eng**, but you can also create and use other color files.

Notes:

When you end an editing session in the Color repository, eDeveloper prompts you to save the changes. The changes will take effect the next time you load eDeveloper, unless you specify Yes for Effective Immediately.

Avoid choosing the same values for both the foreground and background of a display item.

You can save changes in the Color repository to a different file than the one currently used by eDeveloper by specifying a different name in the Save As prompt.

Within a particular application, you can replace the Environment default color file with an application-specific file.

Creating Font Definitions

The **Font repository** associates specific fonts to each kind of available output. 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.

The data in the Font repository is stored in the file specified in the Environment.

Changing the Name and Look of a Font

To change the name and look of a font:

1. In the **Font repository**, double-click a specific font to open the **Font Assignment** window.
2. Select the **typeface, size, font style** (such as Bold or Italic), **orientation**, and **effects** (such as Strikeout or Underline).
3. Click **OK** to accept your changes.

Notes:

The font settings are saved in a special font file that eDeveloper uses in the Environment settings.

You can create and use other font files.

When you finish an editing session in the Font repository, eDeveloper prompts you to save the changes.

Displaying the eDeveloper Executable File Version

To display the eDeveloper executable version on a Windows platform:

1. Start eDeveloper from the command line selecting the version option.
   For **Toolkit** use: **mggenw version**
   For **Runtime** use: **mgrntw version**
2. A window opens indicating the eDeveloper version.

Defining Reusable Application Objects

In eDeveloper, you can define models that act as objects throughout the application.

Changes made to the model are inherited by all the objects that use this model, except for those attributes that have had the inheritance broken.

Models can consist of many object types: Helps, Properties, Browser, and different types of forms.

For some types of Models, such as GUI Display and Browser forms, you can use elements in that Model type to define a model, such as Edit, Buttons, Combo Boxes, and Lists Controls.

Defining Model Properties

To define model properties:

1. Select **Models** from the Navigator to open the **Model repository**.
2. Enter the name of the model.
3. Select the **Model** class. The options are: **Help**, **Field**, **Browser**, **GUI Display**, GUI Output, HTML, Frame Set, HTML Merge.
4. Select the attribute related to the class you choose (for example, Alpha for the Field class).
5. You can choose this model to be the default model when creating an object of

that class.

6. From the **Context** menu, choose **Properties** to view and change the property sheet of the model you are creating.

Notes:

Using models helps maintain consistency throughout the application, and allows for better maintenance.

You may create models based on other models to inherit another model's properties.

Using and Breaking Inheritance

For each class type, property defaults 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 of the value. A broken property value does not appear in italics.

When defining an object you can choose to inherit all the attributes of a model of that same class. You can overwrite some attributes, breaks the chain of inheritance. You can also return the inheritance of that attribute.

Defining an Object

To define an object:

1. Open the **Properties sheet**.
2. In the **Model column**, you can choose one of the models of the same class as the object you created. Choosing a model creates an inheritance chain between the new object you defined and the model to which you connected it.

Breaking and Returning an Inheritance

To break the inheritance, change the value of that attribute.

To return the inheritance:

1. Position the cursor on the attribute that you want to re-inherit.
2. Click the button so that the + sign will change to **x**.

Notes:

Inherited attributes appear in *italics*, and when you select them you can see the button with the **x** sign.

Changing an attribute means breaking the inheritance. The button next to the attribute shows a +.

For better maintenance of your application, creating different models is preferable to breaking inheritances.

Defining Logical Names

Logical names can be considered as application constants. These constants are stored outside the application in the Magic.ini file and can be shared among applications.

Logical names let you make User-specific or Installation-specific customizations to the application. Logical names are also used to simplify the deployment method, and they let you place external files in any location.

Logical names can also be used as the index name for Tables in SQL databases.

To define Logical names:

1. On the **Settings** menu, select **Logical Names**.
2. In the **Name** column, type a name to use for the logical name, for example `MyPath`.
3. In the **Value** column enter the value that the logical name should store. For example: `C:\Program Files\eDeveloper\`

To define Logical names using the INIPut function, set any logical name by defining an Evaluate operation with an expression using INIPut. An example of an INIPut expression is: `INIPut('[MAGIC_LOGICAL_NAMES]MyPath=C:\Program Files\eDeveloper\')`

Using Logical Names

Using Logical names for external files allows deployment of external files in different locations in the environment, and permits easy switching between them using the INIPut function.

To use Logical Names, enter the expression `%MyPath%`
For example, you can point to the Mgconstw.eng file by inserting the following expression: `%MyPath%SUPPORT\MGCONSTW.ENG`.

You can also use the expression:
`INIGet('[MAGIC_LOGICAL_NAMES]MyPath')`
instead of `%MyPath%`

Note: Logical names can be nested within another Logical name, for example:

```
MyAppPath = %MyPath%MyAppDir\
```

This makes the logical name MyAppPath point to:

```
C:\Program Files\eDeveloper\MyAppDir\
```

Defining Default Date Values and NULL Display Strings

You can define default NULL display strings and a default date value for the entire application in the Magic.ini file under [MAGIC_DEFAULTS.]

You can only define these settings by editing the Magic.ini file. The settings are not available in eDeveloper's Environment dialog box. These settings are regarded as the system default values for the default value of a date field definition and the NULL display string for all other field attributes. If these settings do not exist in the Magic.ini file, eDeveloper uses its own already defined defaults.

An example of the section and its available entries in the Magic.ini file is shown below:

```
[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 that the date format should adhere to the defined date mode.

Any modification to these settings during the execution of an application is effective upon the next session of eDeveloper.

Defining Application Default Values and Null Definitions

There are some cases when the user needs to define a variable to hold a specific value. Once you create a record, the fields hold the default values set in the application. This also applies to real fields from database tables.

You can set the way eDeveloper computes expressions when one of the objects in it is holding the value Null.

1. In the **File** menu, select **Appl. Properties**.
2. In the **Application Properties** dialog box, click on the **StartUp** tab and select one of the following **Null Arithmetic** property values:
   - Nullify – eDeveloper computes the entire expression as Null, once one of the objects in the expression is Null.
   - As Default – eDeveloper computes the expression as if the object holding Null is holding the Default Value.

Example:

If two variable are defined as follows:

A (Numeric) = 5

B (Numeric with default value 0) = Null()

If *Null Arithmetic* is set to Nullify, the expression: A+B will be computed to Null.

If *Null Arithmetic* is set to As Default the expression: A+B will result in the value 5.

Defining an External Configuration File for a Specific Application

eDeveloper supplies default external configuration files, defined in the Magic.ini file and in the Environment Settings dialog, that affect application behavior, such as Color Definitions, Font Definitions, and HTML Styles Definitions.

You can, however, change the external files for a specific application. You may want to ensure that your application uses a specific external definition file so that it always behaves the in the same way. For example, you might want to use a specific color definition, even if the end-user has specified a different color definition in the Color Table.

To define a specific external configuration file in Toolkit mode:

1. On the **File** menu, click **Application Properties** or press **SHIFT+F9**.
2. Select the **External Files** tab.
3. Specify the locations (path) of the external files to each of the external files in eDeveloper that can be related to this specific application.

Notes:

Zooming in any field except the Internet Development File Root and the European Currency Conversion file will cause eDeveloper to open the external file and display its definitions.

You can use Logical names to specify the external file locations.

Creating Data Objects

2

eDeveloper has adaptable properties for enhancing or modifying the environment of a specific application. These properties can override the overall eDeveloper environment settings. They are relevant to a specific Magic application file (MCF), unlike the system-wide parameters specified in the Settings menus, which apply to all eDeveloper applications in the system.

This chapter covers the topics listed below:

- Defining a Model
- Forcing Row Uniqueness in a Data Table
- Retrieving Records from the Database in a Specific Order
- Automatic Linking Between Data Tables
- Preventing Changes in the Data Structure
- Modifying eDeveloper Field Attributes
- Converting a Data Table's Physical Definition
- Accessing Existing Tables Using the Get Definition
- Restoring the Structure of a Converted Data Table
- Accessing Database Views
- Adding a Virtual Unique Key
- Using the APG to Manipulate Table Data
- Printing Table Data Directly from the Table Repository
- Automatically Generating Basic Programs

Defining a Model

To define a model:

1. Enter the **Model** repository by pressing **SHIFT+F1**.
   Select the following:
   From the **Name** column, enter the model name.
   From the **Class** column, select the required class type.
   From the **Attribute** column, select the required attribute type.
2. Press **CTRL+P** to view or change the properties of the model that you are creating.

Forcing Row Uniqueness in a Data Table

You use a unique index to enforce the uniqueness of rows in a data table. A unique index ensures that duplicate rows do not exist in the table.

Usually one or more segments are defined for an index. The order of the rows in the index is determined by selecting whether each segment is sorted in ascending or descending order.

If an index is defined as unique, duplicate values for the segments of the index are not allowed.

In eDeveloper, an index can be defined as real or virtual.

- **Real** - If an index is defined as real, the index is created in the database. The database index can help improve performance by allowing the database optimizer to access the data more efficiently.
- **Virtual** - If the index is defined as virtual when created, it is not defined in the database. A virtual unique index does not prevent duplicate rows from being added in the database.

In certain SQL databases such as MSSQL, Sybase, or Informix, a unique index can be defined as clustered when creating a table. This means that the physical data is stored in the order of the index, and the access to the data is faster using that index. Only one index can be clustered.

This section includes the topics listed below:

- Creating a Unique Index for a Table
- Defining Segments in an Index
- Defining Index Properties (for SQL Databases)

Creating a Unique Index for a Table

The following procedure is based on the assumption that the number of columns has already been selected for the table.

To create a unique index for a table:

1. In the **Table** repository, select the **Indexes** column.
   The index details appear in the bottom half of the screen.
2. Move the cursor to the **Indexes** section and press **F4** to create an index. An empty line appears.
3. Type a name for the new index.
4. Select the **Index type** as **Unique** (default).

Defining Segments in an Index

To define segments in an index:

1. Move the cursor down to **Segments** and press **F4** to add a segment to the index.
2. Zoom to the list of columns in the table that appears on the right side of the screen.

3. Choose the column from the list and click **OK**.
4. Tab to the **Size** column if the segment is an Alpha field model. You can shorten the size of the segment to improve performance.
5. Tab to the **Order** column and select **Ascending** (default) or **Descending** for the sort order of each segment.

Defining Index Properties (for SQL Databases)

To define Index properties:

1. Press CTRL+P to define index properties, and click the **SQL** tab.
2. In the **DB Index Name** column, enter the name of the index as it is identified in the SQL database.
3. Specify the Index type as **Real** or **Virtual**. Select Real if the index is stored in an SQL database.
4. In the **Clustered** field, select **No** (default) or **Yes**. If you select **Yes**, the unique index will be created in the SQL database as clustered when the table is created.

Retrieving Records from the Database in a Specific Order

The order of retrieved data has a crucial part in navigating, understanding, and analyzing the result. eDeveloper lets you define the order by using:

- Table keys
- An eDeveloper sort task
- The SQL ORDER BY clause

Notes:

The SQL ORDER BY clause is only available for embedded SQL tasks.

The ORDER BY procedure is first built according to the task index segments. Position segments are also added to the ORDER BY clause if it is a non-unique index.

When using sort, in addition to the task key, the ORDER BY procedure will be built only from the sort segments. If the sort is not unique, eDeveloper adds the position segments to the sort segments.

This section includes the topics listed below:

- Defining Table Indexes and Position
- Choosing an Index Within a TaskSetting the Task Sort Order
- Using the ORDER BY Clause Within eDeveloper

Defining Table Indexes and Position

To define Table indexes and position:

1. Define table indexes (see).
2. On the **SQL** tab in the **Table** properties, define the position as one of the following:
   Default - for the eDeveloper default position
   Unique Index - for a specific index to be the position
   Row ID – for a Row ID position (only in a database that has Row ID for records)
3. From the **Index** parameter, zoom to the **Unique Index** list to select a unique index. Click **OK** to confirm.

Choosing an Index Within a Task

To choose an index within a task:

1. Zoom from the **Program** repository to the **Task** window.
2. From the **Task** menu, select **Task Properties** or press CTRL+P.
3. On the **Properties** tab select a table.
4. Enter the index number or zoom to the **Table Indexes** list.
5. Select the index and click **OK**.

Setting the Task Sort Order

To set the task sort order:

1. From the **Program** repository, zoom to the **Task** window.
2. From the **Task** menu, select **Sort** or press CTRL+S.
3. Press F4 to create a new line in the Sort segment table.
4. Enter the variable alias in the **Var** column
   -OR-
   Zoom to the **Variable** list and double-click the variable.
5. Select **According to Index** to let eDeveloper determine if the sort is unique.
6. Select **Unique** to define the sort as unique, regardless of the table's key definition.

Using the ORDER BY Clause Within eDeveloper

To use the ORDER BY clause within eDeveloper:

1. On the **Task** menu, select the **SQL Command** option, or press **CTRL+Q**.
2. Place the cursor in the source **Database** field and zoom to the **Database** list. Select the database, and click **OK**.
3. From the **Result Database** field, select **Optional**. Follow step 2 above to select the database.
4. Write a valid SQL command with an ORDER BY clause.
5. Click **APG** to generate the program. The **Program Generator** window opens.
6. Click **OK** to create the program.

Notes:

eDeveloper keys can be defined without a real index in the database.

Define keys as Unique in Order to prevent adding the position to the Order by Clause.

Avoid Task Sort, which includes virtual fields or fields from linked tables (except join links).

The SQL Command option is available only if there is no main table.

Automatic Linking Between Data Tables

Data model relationships are maintained by integrity constraints such as foreign keys.

A connection between table segments and their referenced table is defined as a foreign key.

eDeveloper has foreign keys on two levels:

- **Data level** - eDeveloper lets you define foreign keys that may also be created in the database.

- **Application level** - eDeveloper can generate a program that consists of query links to the referential tables.

When you use the Get Definition utility, eDeveloper retrieves the foreign keys defined in the database. When using Get Definition, it is best to first perform the operation on the referenced table.

This section includes the topics listed below:

- Defining a Foreign Key
- Generating Programs Containing Referenced Tables

Defining a Foreign Key

To define a Foreign Key:

1. On the **Workspace** menu, select **Tables**, or press **SHIFT + F2**.
2. Zoom from the **Foreign Key** column.
3. Create a new line in the **Foreign Keys** list.
4. Zoom from the **Referenced Table** column to the **Table** List.
5. Zoom from the **Primary Key** column to the **Index** list, and select the relevant key (the key resembling the connection). Lines for each segment open in the **Columns** list, according to the key's segments.
6. Select the **Create in DB** option to create the foreign key in the database.
7. Zoom from the **Current Table** column to open the **Variable** list.
8. From the **Variable** list, select the required column.
9. Repeat the same procedure for all Key segments.

Generating Programs Containing Referenced Tables

To generate programs containing referenced tables:

1. On the **Workspace** menu, click **Tables**, or select **SHIFT + F2**.
2. Select the required table.
3. On the **Options** menu, click **Generate Program**.
4. Zoom from the **Links** field to the **Foreign Key selection** list.
5. Select the foreign keys individually, or select **the Select All Foreign Keys** check box for all referenced tables.

Preventing Changes in the Data Structure

eDeveloper maintains its own Table repository while providing access to data stored in various DBMSs that maintain their own data dictionaries. The data structure can be modified both by eDeveloper and external DBMS utilities.

The following parameters help control data structure:

- Change Tables in Toolkit - Determines if eDeveloper can alter the table structure of the underlying database
- Check Existence - Determines if eDeveloper will check the existence of every table accessed
- Owner - The owner of the table or view

This section includes the topics listed below:

- Regulating Changes in Toolkit Mode
- Setting the Check Existence Setting
- Setting the DBMS Default Value
- Setting the Database Default Values
- Setting the Table Default Values
- Defining the Owner of a Table

Regulating Changes in Toolkit Mode

To regulate changes in Toolkit mode:

1. On the **Settings** menu, click **Databases**.
2. Select your database.
3. Press CTRL+P to open the **Properties** dialog box and click the **Options** tab.
4. Clear the **Change Tables in Toolkit** check box.

Setting the Check Existence Setting

When you set **Check Existence** to **No**, eDeveloper behaves as if the tables already exist in the database. Therefore the user will not be able to create the tables using eDeveloper.

Setting the DBMS Default Value

To set the DBMS default value:

1. On the **Settings** menu, click **DBMS**. The **DBMS** list opens.
2. Select your DBMS.
3. Press CTRL+P to open the **DBMS Properties** dialog box.
4. In the **DBMS Settings** section, leave the **Check Existence** check box blank.
5. The **DBMS Check Existence** setting provides a default value for all databases for that DBMS.

Setting the Database Default Values

To set the database default values:

1. On the **Settings** menu, click **Databases**.
2. Select your database.

3. Press CTRL+P to open the **Database Properties** dialog box, and click the **Options** tab.
4. Clear the **Check Definition** check box.

Setting the Table Default Values

To set the table's default values:

1. After opening your application, open the **Table** Repository.
2. Select the table you are working with, and open the **Table Properties** dialog box.
3. Click the **SQL** tab. From the **Check Existence** parameter, select **No**.

Defining the Owner of a Table

Every object in the database is a combination of a database, owner, and table. Users can only access objects for which they have access rights.

You can define owners for specific tables to restrict access.

To define the owner of a table:

1. From your application, open the **Table** Repository.
2. Select the table and open the **Table Properties** dialog box.
3. Click the **SQL** Tab.
4. In the **Owner** field enter a name for the required owner.

Modifying eDeveloper Field Attributes

You can modify the way eDeveloper field attributes are stored in the SQL database using the Model or Table repositories in the Column Properties screen. Some of the Column properties only apply to columns in SQL tables.

eDeveloper's default mapping for field attributes assigns a specific SQL data type when it is added to the SQL database. For a particular column attribute, there may be several default SQL types. eDeveloper assigns the default SQL type according to the column attribute, picture, storage, and the specific RDBMS selected.

If the table already exists in the SQL database, the Get Definition option is used and the table definition is retrieved from the database. eDeveloper assigns the column properties according to the data type defined in the SQL database.

There may be situations where you do not want to use the default mapping, and, you can therefore also specify a different SQL type. When you use the Get Definition option, eDeveloper may not define a column attribute as you expected. When that happens, you can modify the column attribute.

This section includes the topics listed below:

- Default Mapping Using the Table Repository
- Mapping a Table Column Using the Get Definition Utility
- Mapping a Table Column Using the SQL Type Option
- SQL Type Options for an SQL Database

Default Mapping Using the Table Repository

To map a table's column:

1. From the **Table** repository, select the table and zoom from the **Columns** field.
2. Select the column you wish to modify.
3. On the **Workspace** menu, click the property sheet to display the **Column** properties.
4. Modify the column attributes.

Mapping a Table Column Using the Get Definition Utility

Sometimes the table already exists in the SQL database and you want to use this table definition in eDeveloper. The Get Definition utility is used to retrieve the table definition from the RDBMS to the eDeveloper Table Repository.

eDeveloper creates the column attribute, picture, and storage properties from the column definition in the database. eDeveloper determines the default mapping of a table column according to the SQL data type and size. The default mapping may vary among different SQL databases.

After performing a Get Definition utility, the Type field contains the column's SQL data type.

However, in some cases the information in the database is not sufficient, and programmer intervention is required. Some examples:

- LONG RAW, Image, and Text data types have no length, so the picture column is left empty to be filled in by the programmer.
- LONG RAW, Image, and Text may be used in eDeveloper for both Memo and Blob variables. After performing a Get Definition operation, you may want to change the attribute that eDeveloper assigned as the default.
- In Sybase and MS-SQL, binary data types may be used for several attributes. After performing the Get Definition operation, you may want to override the default attribute that eDeveloper assigned.

After performing a Get Definition operation on an Oracle table, eDeveloper defines the storage type of a numeric column as Float. You may want to change the storage to Integer.

Mapping a Table Column Using the SQL Type Option

To map different data types:

1. On the **Column Properties** sheet, place the cursor on the **SQL Type** option for the property.
2. If this field is empty, eDeveloper uses the SQL data type that is the default for the

particular SQL database.

3. Enter a different data type to override the default mapping.
   For example, an Alpha column (size 1) is stored as VARCHAR(1) by default. If you enter CHAR(1) in this field, the column will be defined as CHAR(1) in the SQL database.

SQL Type Options for an SQL Database

Examples of when you may want to override default mapping are listed in the table below:

Example	Solution
The eDeveloper date attribute is mapped by default to SQL type DATE.	In order to allow dates with a zero value, store the date as a string field by specifying CHAR(8) in the Type field.
You want to define an identity column in MSSQL.	Enter "INTEGER IDENTITY" in the Type field. The database automatically assigns a value for the column when a row is inserted.
You want to include a logical column in an index in MSSQL.	Change the Stored as field. The default storage for a logical column is Integer Logical, and eDeveloper assigns the bit data type. However, MSSQL does not allow bit fields in an index. Therefore, select String Logical in the Stored as field, and eDeveloper will map the column to the binary data type in MSSQL.
With MSSQL, you want to display date and time in one column.	Define an alpha column of size 23 with DATETIME in the Type field, or an alpha column of size 16 with SMALLDATETIME in the Type field.

Converting a Data Table's Physical Definition

Sometimes you may need to make modifications to a table's structure or to move the tables to a different database. eDeveloper provides automatic data and structure conversion utilities.

To convert a table's physical definition:

1. From the **Table** repository select the relevant table and make any changes to the table structure or database location.
2. Exit the table. eDeveloper prompts you to confirm conversion. Click **Yes** to save your changes.
3. A **Confirm Backup** window opens. Select **Yes** to back up the table's old records and structure. This lets you undo the changes later. The table will be backed up as a table named "??BCK###" where ?? is the application prefix and ### is the table's number.
4. Select an index key that eDeveloper implements when converting the data. eDeveloper now converts the table.
5. It is best to confirm the backup process. To recover the table later you can rename ??BCK### to the real table's name.

Accessing Existing Tables Using the Get Definition

In many cases where the application data resides on SQL database servers, a Database Administrator (DBA) defines the tables, and eDeveloper reads and uses the definitions defined by the DBA.

To execute the Get Definition operation for a single table:

This option is only relevant when you know the exact name of the table.
This option is enabled for tables of a DBMS that support the Load Table Defintion utility, provided that the table does not have columns defined.

1. Open the **Table** repository.
2. Create an entry in the **Table** repository.
3. Select the database where the table will be fetched.
4. From the **DB Table** column, enter the name of the table.

5. On the **Options** menu, select **Get Definition**. The **Load Table Definition** dialog box opens, and then closes automatically. The table now has appropriate fields, keys, and foreign keys.

To execute the Get Definition operation for two or more tables:

1. Open the **Table** repository and place the cursor on the title line (#).
2. From the **Options** menu, select **Get Definition**. The **Load Table Definition** dialog opens.
3. Park on the **Database** field and zoom to select the database where the definitions will be fetched.
4. Park on the **Tag Tables** field and select an option.
   The choices are:
   All
   Several - If you select **Several**, the **Table Selection** list opens.
   None
5. In the **Table Selection** list, park on the desired table and press the spacebar. A check appears in the Select column to indicate that the table has been selected for loading.
6. Press the spacebar on each table that you want to load. In the **Load Definition** dialog box, the **Table** field reflects the number of selected tables.
7. Click **Select** to close the window.
8. Press **OK** to execute the **Get Definition** operation. The selected tables appear in the **Table** repository.

Restoring the Structure of a Converted Data Table

After modifying and converting a physical data table, you may need to recover the changes from the backup table you made during the conversion process.

To restore the structure of a converted data table:

1. Create a new entry in the **Table** repository.
2. In the **Database** column define the database as the one to be recovered.
3. In the **DB Table** column, enter the name of the backup table in that database, in the `??BCK###` format.
4. On the **Options** menu, select **Get Definition**.
5. Park on the existing structure you want to overwrite. On the **Edit** menu, select **Replace** or press **CTRL+W**. Choose the entry just fetched from the database.
6. Confirm the overwriting of the table entry where you are positioned.
   If you want to recover the data as well as the definition, you should use the DB Manager to copy the contents of the backup table you created before the modifications were made. The format of the table name is `??BCK###`, where:
   `??` is the Application Prefix, and
   `###` is the number of the table in the **Table** repository.
7. Renaming the backup table or copying it to the active table recovers the data.
8. You can delete the entry created at **step 2**.

Accessing Database Views

eDeveloper lets you access views in a way similar to the way you access table definitions.

A view is defined as a virtual database object that lets you view the data of one or more tables. A view does not contain a real index. You must define a virtual unique index.

To access the View definition:

1. In the **Table** repository create a new line.

2. Select the database.
3. Write the name of the view to be loaded.
4. On the **Options** menu, select **Get Definition**.
5. Press CTRL+P to open the properties dialog box. Click the **SQL** tab, and in the **Table Type** field select View. Click **OK**.

Adding a Virtual Unique Key

To add a Virtual unique key:

1. In the **Table** repository select the required table and zoom from Indexes.
2. Create a new line and enter an index name.
3. Zoom to **Segments** and create a new line.
4. Select the required index segments and exit the **Segments** section.
5. Press CTRL+P to open the properties dialog box and click the **SQL** tab.
6. Change the **Index type** to **Virtual** and save the changes.

Using the APG to Manipulate Table Data

You can use the Automatic Program Generator (APG) to manipulate the data of a table in eDeveloper and to either view or edit a table's contents immediately.

To manipulate data using the APG:

1. From the **Table** repository select the required table.
2. Press CTRL+G, and on the **APG** tab select Execute Mode and the **Browse** option.
3. Zoom from the **Columns** field to open the **Column Selection** list.
4. Select the columns and order you want. To remove columns from the execution, set the order for each column to 0.

5. Click on the **Style** tab and select the **Line** or **Screen Display** mode. You can also select either **3D** or **2D Style** display.
6. Click **OK**.

Printing Table Data Directly from the Table Repository

You can send the contents of a table directly to a printer, ASCII text file, or console.

To select table content, I/O media, and screen display:

1. In the **Table** repository, select a table you want to print and press **CTRL+G** to open the **Automatic Program Generator** (APG**)**.
2. In the **Option** field select **Print**.
3. Give the selected columns numbers higher than 0 to define which columns will be printed and the order in which they will be printed. Use 0 to disable a column from being displayed.
4. In the **File Name** field select the print destination:
5. **Text file** - Enter the location of the file and its name. Only specifying the file name will cause eDeveloper to place the file in eDeveloper's starting directory.
6. **Console** - Enter *console* in this field.
7. **To Printer** - Enter the name of a printer. From the **Settings** menu, click **Printers list**.
8. Click the **Style** tab and select a screen style, **Line** or **Screen**, and then click **OK**.

Notes:

In the **Style** tab you can choose appearance parameters, such as 3D or 2D, and tell eDeveloper to print out captions that will be used as headers on a printout.

You can select a form style model.

You can execute the **APG** from the **Table** repository to print your tables.

Automatically Generating Basic Programs

You can automatically generate a basic program.

To generate a program for a single table:

1. In the **Table** repository select a table that you want to print and press **CTRL+G** to open the **APG**.
2. Select **Generate** and select any of the following options: **Browse**, **Export/ Import**, **Print**, **Internet** or **Browser Client**. Then enter your information in the relevant fields.
3. Click **OK** to generate a program in the **Program** repository.

To generate a program for more than one table:

1. From the **Tag Tables** field, select **All** or **Several**.
2. Zoom on the selected field, and select or deselect a table from the APG process.

Creating the Task Dataview 3

This chapter discusses some issues related to the task dataview. An eDeveloper task is a set of rules that guide the database engine to perform a predefined function. Each eDeveloper task works with a set of records and their fields that you select from the application database. The set of logical records selected from the main file and linked-files determines the task's dataview.

This chapter covers the topics listed below:

- Defining a Program or Task Dataview
- Determining the Result Set of a Program
- Executing the Same Program with Different Data Tables
- Dynamically Changing the Program Record Order Display
- Adding a Record Sort Order to a Program
- Preventing Modification of Opening of a Data Table
- Caching the Dataview
- Minimizing Unnecessary Links in a Program
- Defining the Dataview

Defining a Program or 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. Real and Virtual variables 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*.

This section includes the topics listed below:

- Defining a Main Table for the Program or Task
- Defining Different Linking Options to Additional Data Tables
- Selecting Virtual Variables
- Direct SQL SELECT Statement

Defining a Main Table for the Program or Task

To select the main table of a program:

1. In the **Program** repository, zoom from the **Main** program to the **Task** window.
2. Open the Program Properties dialog box (**CTRL+P**).
3. On the **Properties** tab, zoom from the **Main Table** field to select a table. After choosing the main table, eDeveloper automatically chooses the first defined index of the table. eDeveloper uses this index for sorting purposes.

4. If the table has more than one index, you can change this index by zooming from the **Index** field.

5. After choosing the **main table**, you can select the referenced object index by using the **Select Real** command and zooming to the **Variable** list.
   Note: This command is available only in the Record Main level.

Defining Different Linking Options to Additional Data Tables

eDeveloper's Link operation establishes one-to-one relationships between related database tables. The current record of the task main file is correlated, or linked, to a specific single record of another database table, the linked file, whose index segments contain values that match the Link criteria.

Links can be established to:

- Perform validity checks in order 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.

Since the Link is part of the dataview definition, the link operations are included in the Record Main level definition of the task only.

There are five link types: Query, Create, Write, Inner Join and Left Outer Join. The last two are for SQL databases only.

Selecting Virtual Variables

Almost all tasks need to store certain information for the duration of the task execution. For this purpose, you can define Virtual variables that are selected from a temporary scratch file called a Virtual file. The Virtual file may be edited by selecting Variables from the Task menu. All Selected Virtual Variables appear as entries within a task's Virtual Variable repository.

The Virtual Variable repository behaves just like the Column repository of a table. The Virtual Variable repository is different from the Table's Column repository in that 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 operation in the Record Main Execution repository.

Direct SQL SELECT Statement

With eDeveloper's embedded SQL support, the developer can provide the SQL statements explicitly and have them transferred to the underlying DBMS for processing. eDeveloper then manipulates the results of these SQL statements.

The SELECT statement retrieves data from a database and returns it to you as a table. The columns specified in the SELECT statement are the names of the database columns that contain the data you want to retrieve.

Determining the Result Set of a Program

The result set comprises the data selected according to the criteria specified in the program or task. The dataview is sorted by default according to the Task Properties Main Table Index. You define the selection criteria of the result set in the Record Main or in the Task's Range/Locate dialog box.

- You can define a Range or locate expression in the Range/Locate dialog box. You can access the Range/Locate dialog box from the Record Main.
- With SQL databases, you can use the eDeveloper SQL expression and DB SQL clause in the Range/Locate repository:
- eDeveloper SQL expression - When working with an SQL database and deferred transactions, you can write an eDeveloper expression that eDeveloper translates in Runtime to SQL syntax.
- Database SQL clause - When working with an SQL database and physical transactions, you can use SQL table columns and virtual columns in SQL syntax to specify a range.

This section includes the topics listed below:

- Selecting an Index from the Main Table
- Defining the Requested Range
- Defining Locate Expressions
- Defining the Magic SQL Where Clause for an SQL Database
- Defining the DB SQL Where Clause for an SQL Database

Selecting an Index from the Main Table

To select an index from the main table:

1. Select an index from the **Main table's** index list to determine the order in which the records are displayed.
2. Zoom from the **Index** column to select an index.

Defining the Requested Range

Range is one of the ways of defining a task's dataview. There are two ways to specify the range values:

- On Record Main - Range expression in the Range or Locate property sheet accessed from the task menu (**CTRL+H**).
- Record Main - The rows are initially included in a dataview based on the lower and upper limit expressions of the Range, as specified in the task's Select operations.

Range and Locate Property Sheet

The Range Expression allows you to refine the Range criteria further, and base these criteria on more complex and dynamic conditions:

- 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:

- The criteria depend on values contained in variables not available at task initialization.
- The Range criteria are not definable by the lower and upper limit expressions of the Select operations.

Defining Locate Expressions

Locate 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 initial point 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 and forth within the dataview using the direction keys.

There are two ways to specify the Locate values:

- On Record Main - In the Locate expression field on the Range and Locate property sheet
- Record Main - To use the Locate expression, you must specify the value of the record you want the cursor to park on.

Range and Locate Property Sheet

The Locate Expression allows you to refine the Locate criteria further, and base these criteria on more complex and dynamic conditions:

If the Locate expression evaluates to True, the cursor parks on the first row that meets the criteria of the expression.

If the Locate expression evaluates to False, eDeveloper looks for another row to park on.

If a Locate expression is not specified, that is, its number is zero, it evaluates to True.

A Locate expression is required when the criteria depend on values contained in variables not available at task initialization.

Note: The Locate Expression parameter is relevant for Online tasks only. All variables included in the Locate expression are evaluated at the task's initialization. As such, they may contain only parameter variables or variables of ancestor tasks. 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.

Defining the Magic SQL Where Clause for an SQL Database

To define the Magic SQL Where Clause, for the SQL DB:

1. Open the Task's **Range/Locate** dialog box.
2. Click the **SQL Where** tab.
   The Magic SQL expression allows SQL Where range functionality for deferred transactions. It includes expressions that the eDeveloper engine can translate to SQL.
3. Zoom from the Expression field to the **Expression Rules** repository.
   The expanded expression is appended to the Full Where Clause.
   eDeveloper translates the expression in runtime to the appropriate syntax for each SQL database. For example, the eDeveloper SQL expression G<DATE(), for a Customer Contact Date field, will appear in Runtime as:

   MSSQL:

   ```
   CAST (CONVERT (CHAR, Customer_Contract_Date, 112) AS
   DATETIME) < CAST (CONVERT (CHAR, GETDATE(), 112) AS
   DATETIME)
   ```

   Oracle:

   ```
   TO_DATE(Customer_Contract_Date, 'DD-MON-YY') <
   ```

```
TO_DATE(SYSDATE, 'DD-MON YY')
```

Defining the DB SQL Where Clause for an SQL Database

To define the DB SQL Where Clause for SQL DB:

1. Open the Task's **Range/Locate** dialog box.
2. Click the **SQL Where** tab.
   The **DB SQL field** is available only for tasks in **Physical Transaction** mode with an SQL main table. (In Deferred/Nested Deferred transaction mode tasks, this field is disabled, and the eDeveloper SQL expression should be used instead.) You can specify a range based on SQL syntax. The Range is added to the eDeveloper Where clause and is displayed in the Full Where Clause.
3. Zoom from the **DB SQL** field to display the list of variables available.
   Real columns from main and joined tables will be replaced with their DB column name. Virtual and other real columns will be replaced with their values.

Variable List		
#	Name	From table
A	v_Operation	Virtual
B	v_Customer_Name	Virtual
------	------------------------------	----------
C	Customer_No	Customers
D	Customer_Name	Customers

Specify selected variables as follows:

- A column # (for example, A) prefixed by the ':' sign.
- A column # prefixed by the '@:' sign. The @ character prevents eDeveloper adding quotes to an alpha column.
  In the above example, D is a main table column, A and B are virtual columns with Runtime values `LIKE` and `Jones` respectively.

`:D @:A:B` is displayed as
`Customer_Name [v_Operation] ["v_Customer_Name"]`
and will translate in Runtime to
`Customer_Name LIKE 'Jones'`
The column values are evaluated when entering the task. Therefore, virtual columns must receive their values from the calling task. The init expression cannot be used because it is calculated afterwards.

- Click **Show** to display the Full SQL Where clause. It appears as follows: The Where clause expression from the Record Main AND (The eDeveloper SQL Where clause) AND (The DB SQL Where clause)

Note: Blobs and Memo fields cannot be used in the SQL range.
eDeveloper does not check the syntax of the SQL range. If the SQL syntax is invalid, a message from the RDBMS will occur in Runtime.

Executing the Same Program with Different Data Tables

You use the DB Tables repository to view and edit the parameters of the database tables associated with the task, as well as to add other tables to be opened when the task starts.

Note:
A non-zero value in the Expression column identifies an expression that will be evaluated to a physical database table name at runtime.

You can also 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, for example:

```
c:\temp\table.dat
```

```
%temp%table.dat where the translation of %temp% is
c:\temp\
```

Accessing the Database Table Repository

To access the database Table repository and Table list:

1. On the **Task** menu, click **DB Tables**.
2. Zoom from the **Table** column.

Dynamically Changing the Program Record Order Display

The default index for a Main table is the first index in the Index repository.

By choosing a specific index, you can determine the fetching sequence of the dataview records for this task. You can also use an expression for the index, making it more flexible.

This section includes the topics listed below:

- Changing the Display Order of Records
- Using the Expression Index Parameter

Changing the Display Order of Records

To change the display order of records:

1. Open the **Program** repository and zoom to the **Task** window.
2. From the **Edit** menu, select **Properties**, or press CTRL+P, and then click the **Properties** tab.
3. Leave the **Main Table Index** field as zero, enabling you to use the **Index Expression** parameter. The expression number you enter in the Expression parameter of the Main Table points to an expression in the **Expression Rules** repository that is evaluated at runtime.

Using the Expression Index Parameter

When using an Index expression:

- The result of the evaluation of the expression must be a valid index number for the selected Main Table. At runtime, if no legal value is 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 the default option.
- The Index expression is evaluated as soon as the task starts executing and before any of the task's variables are available. Therefore, an index expression based on variables, rather than constant values, must either use virtual variables that receive parameters, 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 and you want eDeveloper to update it automatically, you must qualify it explicitly using the KEY literal. For example, to specify the index number *nn* you would enter: `nn KEY`

Adding a Record Sort Order to a Program

You set the default sort order by specifying the main table and index in the task properties. You might want to change this order and not depend on an existing key in the table definition.

eDeveloper helps you by using the Sort repository (in each program).

The Sort repository is a compound window that includes a:

- Segment area on the left-hand side of the window
- Variable list on the right-hand side of the window

This section includes the topics listed below:

- Selecting the Database for Sort/Temporary
- Using a Virtual Key to Re-use a Sort

Selecting the Database for Sort/Temporary

Sort files are created using the Database for Sort/Temporary as defined in the Environment dialog box.

To select the Database for Sort/Temporary:

1. On the **Settings** menu select the **Environment** option.
2. Select the **Preferences** tab.
3. Place the cursor on the Database for Sort/Temporary row in the Parameter column.
4. Zoom to the **Database** list and select the database name to be used by eDeveloper for creating Sort tables or other temporary database tables needed as temporary storage, such as the embedded SQL result table.
   For example, a fast ISAM database that uses memory for storage on a RAM disk may be used to provide good performance for sorts when the original tables are stored in a traditional disk-based database.

Notes: The location of a sort table's database is always used if the location has been specified in the Location parameter of the Databases repository option on the Setting menu.

The path specified in the Database for Sort/Temporary parameter is used only if the Location parameter in the Databases repository, accessed on the Settings menu, is blank.

Using a Virtual Key to Re-use a Sort

eDeveloper helps you define a sort index without the need to define this index in the database table. You can define a new index on the SQL tab of the Index Properties dialog. The definition looks exactly like that of a regular index except you set the Index type field to Virtual instead of Real.

This setting tells eDeveloper that the index is defined only in eDeveloper.

Preventing Modification of Opening of a Data Table

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.

This section includes the topics listed below:

- Using Access Mode
- Using Share Mode
- Defining Table Modes
- Table Sharing Interaction

Using Access Mode

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 the process can access the table. 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 are conflicting. This is the reason for the second column, the Share Mode.

Using Share Mode

Share mode is relevant in a multi-user environment only. It determines how other concurrent tasks can access the table.

In addition to declaring the database access mode, you must declare its share mode. The share mode specifies which access modes may be granted to other processes.

The share mode is set to **None** for no shared access (Table locking). This means that no other process can be granted access to the table.

Defining Table Modes

You define the access and share modes of a task in its DB Table repository.

The DB Table repository of a task includes the two columns labeled as:

- **Access**, whose value can be W for read/write (the default) and R for read only.
- **Share**, which can have the value of **W** for read/write (the default), **R** for read, and **N** for None. **None** means the table is not shared in any mode.

Table Sharing Interaction

The following examples describe the sharing interaction between two tasks that open the same table.

- If process A opens 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.
- If process A opens the table in R/N mode, process B cannot open the table in any modes.

Caching the Dataview

The Magic cache can be used for main and linked tables. The cache is used mainly when you need to access the same data more than once. The cache reduces disk I/O, thus enhancing overall performance. Caching of a main table can be implemented at two levels:

- Table Properties (the default value for Linked and Main table)
- Task Properties

Cache can be implemented by selecting **DB Tables** from the **Task** menu.

Note: Using the Magic cache may cause inconsistencies in the data.

This section includes the topics listed below:

- Defining a Main Table Cache
- Defining a Linked Table Cache
- Defining the Cache Strategy Parameter

Defining a Main Table Cache

To define the cache for a main table:

1. Open the **Table** repository.
2. Choose the table you want to cache.
3. Open the **Table Properties** dialog box.
4. In the **Cache Strategy** field, select **Position and Data** or **Position** (for Rows position only).

To override the Table property default by changing the cache default in the Task Properties dialog box:

1. Open the **Program** repository.
2. Click the **Enhanced** tab of the **Task Properties** dialog box.

3. In the **Cache Strategy** field of the **Data Management Options** section, you can overwrite the table's task properties default and choose one of the following cache strategies: **Position and Data**, **Position**, or **None**.

Defining a Linked Table Cache

To define the cache for a Linked Table:

You can define the cache for a linked table in a task using the DB Tables option.

1. Open a program from the **Program** repository.
2. Select the **DB Tables** option from the **Task** menu. The DB Table repository opens.
3. Select the linked table you want to enable or disable the cache.
4. In the **Cache** column choose **Yes** or **No**. The default value is inherited from the **Table Properties** dialog box (**Position and data** = **Yes**).

Defining the Cache Strategy Parameter

The Cache Strategy parameter can have one of three possible values: Position, Position and Data, or None.

- **Position** - The cache holds information about the position of the fetched rows. This is relevant only when the table is used as the main table. In this case, when the user scrolls backwards, the data will be re-fetched by reading the rows by their physical position.
- **Position and Data** - In addition to the position, eDeveloper stores the actual row data. If you refetch that row you will get the old values stored in the cache.
- **None** - No caching.

Minimizing Unnecessary Links in a Program

The Cnd column of the Flow table of a task lets you define the new Link Condition. This column enables the user to define a link condition using the values of a Yes/No/ Expression. The expression should be evaluated to a logical value. The Link Condition determines whether eDeveloper will try to fetch a record or not. Link Condition behaviors are:

- False Link Condition

If a Link operation is evaluated to a false condition, it behaves as a failed link, a link for which the requested record was not found, as follows:

  - All values of the selected columns of the Link operation return 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 false at the end of a Record Suffix, any modification of linked records that might have been fetched since the Record Prefix of the current logical record are disregarded.

- Link Create - In a Link Create there is no case of a failed link. However, a Link Create that is evaluated to a false condition acts as follows:

  - The link does not create any new records.
  - Any Init setting or Update operation on this link is not updated in the database until the condition evaluates to True, and is kept as true until the Record Suffix is completed.
  - Any Init setting or Update operation on this link is reflected in the Dataview regardless of the link condition.
  - If the link condition is false at the end of a Record Suffix, any modification of the supposedly new record is disregarded and no new record is created.

- Computation
    - The link condition is evaluated for every record during the creation of the dataview.
    - The link is also evaluated before the Record Prefix of every browsed record.
- Re-computation - The link condition is re-computed by any changed value that is part of the condition expression from the start of Record Prefix phase until the end of the Record Suffix phase.

Defining the Dataview

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.

The Direct SQL command can be used to perform different DML (Insert, Update, Delete) operations.

The RDBMS can perform vertical updates and deletes with one SQL statement within a simple transaction.

You can use explicit SQL where DDL (Create Index) operations are specific to Runtime. For example, you may also want to create a temporary table in the RDBMS.

The Direct SQL Command can also be used to execute the Database Stored procedure.

This section includes the topics listed below:

- Defining Direct SQL Select Statements
- Defining a Direct SQL DML Operation
- Executing a Stored Procedure with Input and Output

Defining Direct SQL Select Statements

To define Direct SQL Select statements:

1. Open the **Program** repository.
2. Create a new program and enter the program name.
3. Zoom to **Task Properties** on Online defaults.
4. From the **Task** menu, select the **SQL Command** option.
5. In the **Database** field, select the relevant SQL database. This example is for the Oracle database using the Scott schema.
   The Result Database should be the same database. This means that you either select it or leave it blank.
6. Enter the SQL Command. Note that you can use eDeveloper's Assist button to help you build the query and the APG button when return values are expected.

   Example: SELECT ENAME, JOB,SAL FROM SCOTT.EMP GROUP BY JOB, ENAME,SAL HAVING SAL > 1000

Defining a Direct SQL DML Operation

To define a Direct SQL DML operation:

1. Open the **Program** repository.
2. Create a new program and enter the program name.
3. Ensure that the **Task** properties are set to **Batch** defaults.
4. From the **Task** menu, select the **SQL Command** option.
5. On the **Database** field, select the relevant SQL database (this example is for the Oracle database using the Scott schema.
6. Enter the DML statement.
   In this example we perform vertical updates.
   UPDATE SCOTT.EMP SET COMM=1000

Executing a Stored Procedure with Input and Output

To execute a Stored Procedure with Input and Output Parameters:

1. Create a Stored Procedure in your database using the RDBMS tool, for example Oracle's SQL*Plus.
2. In eDeveloper, open the **Program** repository.
3. Create a new program and enter the program name.
4. Zoom to ensure that the **Task** properties are set to **Batch** defaults.
5. From the **Task** menu, click the **SQL Command** option.
6. In the **Database** field, select the relevant SQL database (this example is for the Oracle database using the Scott schema.
7. In the **SQL Command** field, enter the EXEC command, as follows:
   `exec procedure_name (parameter)`
8. You must separate the placeholder for the parameters with a comma, as the example that follows:
   `EXECUPD_EMP(:1,':2')`

Note: eDeveloper executes the SQL command before it executes the Task Prefix level.

Using Basic Programming Techniques

4

This chapter describes Basic Programming techniques that you can use for creating and modifying eDeveloper programs and tasks.

This chapter covers the topics listed below:

- Automatically Generating a Simple Program
- Defining Global Variables for an Application
- Controlling Execution of the Main Program
- Defining Application Level Events
- Conditionally Terminating the Program
- Preventing a User from Manipulating Program Data
- Displaying Information after the Task is Closed
- Removing the Need to Confirm Record Deletion

Automatically Generating a Simple Program

A program can be generated from within the Table repository or from an entry in the Program repository.

eDeveloper's Automatic Program Generator (APG) 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 programs.
- General Web programs (HTML report and an HTML query screen).

The Program Generator dialog is defined by:

- APG
- Style
- Internet

This section includes the topics listed below:

- Generating Programs from the Table Repository
- Generating Programs from the Program Repository

Generating Programs from the Table Repository

To generate a program from the Table repository:

1. Park on a table in the **Table** repository.
2. From the **Options** menu, click **Generate Program** or press CTRL+G. The **Program generator** dialog box opens.

3. Choose the mode and type of program.
4. Enter information in the fields on all three tabs as explained in the following sections:

Generating Programs for Multiple Tables

To execute or generate programs for more than one table in one run:

1. In the **Table** repository, move the insertion point above the first entry.
2. On the **Options** menu, select **Generate Program** to open the Automatic Program Generator dialog box.

You can run the generator immediately with the default values, or you can make any appropriate changes to the option settings before you run the Automatic Program Generator. The APG tab has the fields and parameters shown in the table below:

Field	Options	Purpose
Tag tables	All	If chosen, the Selected field displays the total number of tables on the Table List.
	Several	If chosen, the Table List opens automatically. See the procedure for selecting the tables below.
	None	Specifies that the Program Generator is disabled for this session and will not create any programs
Selected		Displays the number of tables selected
Mode	Execute	Executes the program
	Generate	Generates the program
Options	Browse, Export, Import, and print	

Note: The Web tasks are not available when generating multiple programs.

Selecting Several Tables

To select the tables:

1. From the **Tag Tables** field, select **Several**. The **Table** list opens.

2. Press the space bar to mark the tables you want to select in the right-hand column, and click **Select** to confirm the selection. The field displays the number of tables selected.
3. eDeveloper creates a program for each table in the **Table** repository.

Generating Programs from the Program Repository

To generate a program from the Program repository:

1. Open the **Program** repository and press CTRL+G to open the **Automatic Program Generator** dialog.
2. Choose the type of program you wish to generate by selecting an option. **Note:** The **Execute** option is not available when generating a program from the **Program** repository.
3. Place the cursor on the **Main Table** field and zoom to the **Table** list to select a table.

Defining Global Variables for an Application

Opening an application in runtime automatically executes the Main program. The variables defined in that program are global, and shared by future executed programs.

Closing an application that is in Runtime mode causes the Main program to end, and executes the operations defined in its task suffix.

Changing from Toolkit to Runtime mode, and vice versa, re-executes the Main program.

You can define global variables in one of the following ways:

- Select virtual variables.
- Link to a table and selecting that table's real variables.
- Use the SetParam and GetParam functions.

- Use INIPut and INIGet functions. In this case, you can use the option of sharing the Magic.ini in the server for the different instances.

When defining Global variables, updating of any Real variables are ignored since no locking nor transactions are allowed at this point.

Re-computing of variables from the parent task, which are used in sub-tasks are controlled by a flag in the eDeveloper environment. Thus, changing a Main program's variable causes eDeveloper to re-compute all the affected programs, in which this variable is used, in Initial or Link operations.

Controlling Execution of the Main Program

The Main program is executed automatically when moving into Runtime mode. eDeveloper Runtime mode is defined as one of the following:

- Runtime engine - foreground runtime, background runtime, and generator engines.
- When a program is run under the toolkit engine and the application initially is opened in Runtime mode.
- When a program is run under the toolkit engine and the application was switched to Runtime mode (**CTRL+T**).
- When a program is run under the toolkit engine and the program was run from the Toolkit mode (**F7**).
- You can control the execution of the Main program by using the RunMode function.
- Define the operations in the Main program to be executed conditionally using the RunMode function.

Each operation or block of operations should have the requested execution condition. The condition for each eDeveloper Runtime mode above is:

- Runtime engine, ()=0
- Initial Runtime mode, ()=1
- Switch to Runtime mode (CTRL+T), RunMode()=2
- Toolkit mode (F7), ()=3

Note: The condition of the RunMode function can be used in all the places in the MCF where expressions are evaluated.

Defining Application Level Events

In eDeveloper V9, you specify an application event in the Main Program event. The Application Event repository (**SHIFT+F8**), from previous versions, has been removed.

To define an application event:

1. Define the event in the **Program Event** repository (**CTRL+K**).
2. Define a handler for this event.

To define an application handler level:

1. Zoom from the **Program** repository to the **Task** screen.
2. From the **Task** menu, select the **User Events**, or press **CTRL+K** to open the **Event** repository.

To open a new line and enter a description:

1. Select the handler type. **System** is the default.
2. Zoom from the **Trigger** column to the **Key Definition** screen and enter the shortcut key.
3. Select the **Force Exit** option, and click **OK**.
4. From the **Events** screen, click **OK** to return to the **Task** screen.
5. The event definition is similar to a regular event in the program's Event repository.

Defining a Handler

The Handler definition looks like the regular Handler level in a program. However, there are differences, which are listed below:

- There is no Object in the Main program handler definition.
- There is no Global scope in a regular program handler definition.

The Scope parameter on the Task screen has three values:

Option	Purpose
Task	The handler executes the event in that task. It is not relevant for application events.
Subtree	The handler executes the event in that task and all of its subtasks. This is an Application event, and only for this application.
Global	Once you define an event in the Main program it appears in every program, as if it were part of that program. In spite of this, you cannot modify or delete it from the program itself.

Conditionally Terminating the Program

When you send an exit action, such as pressing ESC, you terminate online programs, by default.

- Batch Programs with a Main Table terminate once all records in the range are processed.
- Batch Programs without a Main Table are infinite loops unless you use the ending condition discussed in this document.

Terminating a Program

To terminate a program:

1. Press **CTRL+P** in the task dataview to open the **Task Properties** dialog box.
2. Select an option in the **End Task Condition** field:

No (default)
Yes, or
F5 to select an expression.
This is the condition that must apply for the task to end.

3. Select the attribute in the **Evaluate Condition** field.
   The attributes are:

   - Before entering record: the condition is checked before Record prefix.
   - After updating record: the condition is checked after record suffix operations.
   - Immediately when condition is changed: the condition is checked before each operation.

Notes:

When the condition is evaluated to **Yes** (True), the task closes. Without an **End Task** condition, an Online program will only close when **Close**, **Exit** or **Esc** are sent.

If there is no Main Table in a **Batch** task, it is important to enter an **End Task** condition. If this is not done, eDeveloper views the task as an endless loop. If there is a Main Table, once all records in the range are processed the Task finishes.

A common usage for conditionally terminating a task is in **One-To-Many** tasks. When browsing, the Header File should show the Detail Range, and both tasks should be Online. You do this by placing a condition for the Detail Task to end when the Header Task has called it from a specific task-level (such as Record Prefix).

Another way to terminate a program according to a condition is to send the **Exit** action manually from within the task, with some specific condition or action.

Using the Raise Event

The Raise Event, new to eDeveloper V9, can raise an event from any execution level in the task. Using this operation to Raise an internal event such as Exit allows for the termination of the task, or handling of such an event from the task itself.

This operation has the Wait attribute that indicates whether eDeveloper will pause all other executions of the task until this event is handled, or continue other operations while the event is Raised.

Another added powerful attribute, is the ability to send Arguments with the event itself, using the Arg attribute. Zooming allows you to define the parameters.

To use the Raise Event:

1. From the **Program** repository, select a program.
2. Zoom to the task's **Record Main** and park the cursor at the beginning of the first field.
3. Zoom to open the **Operation** list, and select **Raise Event**.
4. Move the cursor to the **Name** field and zoom to the **Event** dialog.
5. Select one of the Event type options: **System**, **Internal**, or **User**. The selection of the type determines the **Event Name** options.
   - System - Key Definition dialog box
   - Internal - Action list
   - User - Event list

In versions prior to eDeveloper V9, to send such an action to eDeveloper, as though made by a user you would use the functions:

KbPut(Exit Act) or KbPut(Esc KBD).

Placing an evaluate expression with such KbPut() causes the Task to end.

Triggering such an operation can be done in many ways:

- Placing an Evaluate operation in some Task level. Once this operation is executed, it will close the task.
- Triggering this Operation by zooming from a field.
- Placing a button on the Main Form with the return action Exit.

Note: eDeveloper recommends avoiding the KbPut operation as much as possible because it is affected by Keyboard Idle time and might cause unwanted behavior. Instead, eDeveloper recommends using the Raise Event Operation since it allows better control of the execution using the Wait and Arg attributes. You can also use a Virtual Field acting as a flag to indicate when to close the task, and to use this flag in the End Task Condition, as explained above.

Preventing a User from Manipulating Program Data

There are situations, when designing an Online task with intended user interaction, where you want to design a display for the end user but limit the ability of the user to make changes to that display. The following are some examples:

- Preventing modifications to data through the basic eDeveloper methods.
- Preventing changes to elements that are displayed on the screen.
- Preventing modifications of the display itself.
- Disregarding a user's attempts to change the data through non-specific programmed screens.

This section includes the topics listed below:

- Selecting Task Property Attributes
- Controlling Use of the Options Menu
- Disabling the Options Menu
- Controlling User Positioning of the Cursor
- Preventing Data Manipulation

Selecting Task Property Attributes

To select the Initial mode:

1. From the **Task Properties** dialog box, click the **Properties** tab to indicate the mode of action the program or task starts with. The Initial mode field default is **Modify**.
   Any selected mode determines the starting attribute when the user runs the program.
2. Select the **By Exp** option if you want to define the initial task mode with an expression.
   You use **By Exp** when you want flexibility in selecting the initial mode due to specific circumstances. For example: IF (TDEPTH()>1,'Q'MODE,'M'MODE)
3. The task should be in **Modify** mode when it's launched by itself, but if another program calls this task, it should start in **Query** mode.

Controlling Use of the Options Menu

Even when a task is set to an initial mode, the end user may change the mode from the Options menu. Using this menu the end-user can change to Modify Mode or Create Mode even if the initial mode is Query. Using Options the user is able to:

- Locate records
- Range record
- Change the Key used to access the Main Table of the task
- Sort records according to user defined conditions.

If allowed, the end-user can delete records form the Database using the Edit menu.

In many cases, a developer may want to prevent the user from using the Options menu actions that may change the display or database. The Task Control Properties dialog helps you deal with this.

Disabling the Options Menu

To prevent the user from having access to the Options menu, you must either specify No in the Allow Options, or zoom to the Expression rules window and specify a Boolean condition for limiting the user's options. Selecting No or a limiting expression rule prevents the user from performing specific actions such as Modify, Create, Delete, Query, Locate, Range, Index change, or Sort.

Controlling User Positioning of the Cursor

To control user positioning of the cursor:

1. In **Record Main**, you use the **Condition** column for the **Select** operation of the task.
   This allows the user to position on some variables, permitting modifications.
2. You can set this condition to Yes, No, or zoom to the **Expression Rules** repository. If set to No, or the Boolean condition is evaluated to No, eDeveloper does not allow the user to position the cursor on the selected field.
3. Another method for allowing the user to position the cursor is by using **CTRL+U** to open the **Main** form. You can edit the **Main** form and use control properties to allow the user to position the cursor on the selected field but prevent modification of its contents.

Preventing Data Manipulation

On the Control Properties screen, if you set the Modifiable attribute to No or specify an expression that will be evaluated to False, eDeveloper prevents the end user from modifying the selected field, and treats it as if in Query Mode.

Another option for preventing data manipulation is:

1. Press **CTRL+D** to open the **DB Tables** screen.
2. Select **Read** for the **Access** parameter. This prevents any Write actions to the table. Any changes sent to a Read table, are disregarded.

Displaying Information after the Task is Closed

Many application designs require task data to continue to be presented even when the task has ended.

To force a task or program to keep the last displayed screen in view:

1. Press CTRL+C to open the **Task Control** dialog box.
2. Select **No** in the **Close Task Window** field.
   This keeps the last program display viewable after the program or task ends.
   If you select Yes as the attribute, and exit the parent task or program, the display is cleared.

Using the Task Properties Dialog

Another way to keep a task or program display viewable is to control the end condition of the task or program. You can do this in the Task Properties dialog.

To force a task or program to keep the last displayed screen in view:

1. Press CTRL+P to open the **Task Properties** dialog box.
2. From the **End Task Condition** field, select **Yes**
   -Or-
   Zoom to **Expression Rules** repository and select or create an expression.
   Setting a condition for ending the task in the **End Task Condition** field indicates that a condition must apply for the task to end.
3. Select a condition in the **Evaluate Condition** field.
   This attribute has the following values:
   - Before entering record: the condition is checked before the **Record Prefix** operations.
   - After updating record: the condition is checked after **Record Suffix** operations.
   - Immediately when condition is changed: the condition is checked before each operation.

4. The task closes once the condition is evaluated as Yes (True).Without an End Task Condition, an Online program closes only when the Close, Exit or Esc is sent.
5. In a **Batch** task without a Main table, the absence of an End Task Condition creates an infinite loop. If there is a Main table, once all records in the range are processed, the Task finishes.

Removing the Need to Confirm Record Deletion

When developing your application you can set up a mechanism that lets end-users delete a record without confirming the action.

To avoid the need to confirm deletion of a record, do the following:

1. In the **Task Execution** repository, create a **Handler** level for the task.
2. From the **Event** column, zoom to open the **Event** dialog box.
3. From the **Event Type** list, select **System**. From the **Event** field zoom to open the **Key Definition** dialog box and press **F3** to assign the **F3** key to the event.
4. Create a subtask. In the **Task Properties** dialog box, set the following:
   From the **Task Type** list select **Batch**.
   From the **Initial Mode** list select **Delete**.
   Make the **End Task Condition** as **Yes**.
   From the **Evaluate Condition** list select **After updating record**.
   Select a Main table.
   Then click **OK**.
5. In the subtask's Record Main, create a **Select Real** operation and zoom to select the variable that appears in the **Index** property in the **Task Properties** dialog box.
6. From both sections of the **Range** column, zoom to the **Expression Rules** repository and select the same variable that you selected for the **Select Real** operation.

7. Return to the main task.

8. In the **Operation** repository press **F4** to create a new line and from the **Operation** list select the **Call Task** operation. From the space next to the words "Call Task", zoom to open the **Subtask List** and select the subtask that you created in step 4.

9. Press **F4** again to create another line, and from the **Operation** list select the **Raise Event** operation.

10. From the **Name** column zoom, to open the **Event** dialog box and from the **Event** list select **Internal**. From the **Event** field zoom to open the **Action List**. Select **Previous Row** from the **Action Name** list and then click **Select** to return to the **Event** dialog. Click **OK**. Set the **Wait** property to **No**.

11. Press **F4** again to create another line and from the **Operation** list select the **Raise Event** operation.

12. From the **Name** column zoom to open the **Event** dialog box and from the **Event** list select **Internal**. From the **Event** field zoom to open the **Action List**. Select **View Refresh** from the **Action Name** list and then click **Select** to return to the **Event** dialog box. Click **OK**. Set the **Wait** property to **No**.

Building an Online Task 5

This chapter discusses online tasks and how to create them.

This chapter covers the topics listed below:

- Evaluating a Program's Initial Starting Mode
- Creating a Selection List Program
- Defining a Context Menu for a Program
- Automatically Saving Changes to the Dataview
- Reconfirming User Steps in a Program

Evaluating a Program's Initial Starting Mode

Most tasks have specific Initial modes. You can also specify an Initial mode, by using an expression. An expression would be used in instances where you want only one program to show a screen, instead of two programs showing the same screen with the same logic. An example of this would be the same Online screen used to query a record and to modify or add a record.

This section includes the topics listed below:

- Selecting the Initial Mode Expression
- Using an Expression in the Initial Mode

Selecting the Initial Mode Expression

To select the Initial mode expression:

1. Open the **Task Properties** dialog box.
2. On the **Properties** tab, in the **Initial Mode** field, select the **By Exp** option.
3. Enter the expression number in the **Exp** field, or zoom (**F5**) to select from the available options.
   The expression will be evaluated at runtime to determine the Initial mode.

Using an Expression in the Initial Mode

The Initial mode is evaluated as soon as the 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 parameters 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 appears informing the user and prompts the user to terminate the task.

By using the MODE literal in the Initial Mode expression, you instruct eDeveloper to check the string content for valid task modes. Any character in that string which is not a valid task mode will be cleared automatically. The values represented by the characters in the string are stored in an internal representation. If the application is used with a non-English language version of eDeveloper, the values in the string will automatically be changed to the corresponding values of the relevant language.

Creating a Selection List Program

User-friendly online data-entry applications often provide lists that enable the end-user to zoom from a specific field to select values from a related list.

You can incorporate this type of list in your application by utilizing the Call Program or Call Task operation, passing parameters from the calling program or task to the called program or task.

An alternative, easier method is to define Selection Table programs and associate them, using Select Program specifications, to automatically zoomable fields. This also lets the end-user select values from tables during the Locate and Range operations, which simplifies the process.

This section includes the topics listed below:

- Creating a Selection List Program by Calling a Program
- Defining a Selection Table Program
- Associating a Selection Table Program to a Column
- Executing a Selection Table Program

Creating a Selection List Program by Calling a Program

Selection List programs have a built-in mechanism that enables them to return the selected value to the calling program, only when the user explicitly selects it and ignores the selection in any other instance.

The program accepts a parameter and returns the value selected, or the previous value if no selection was made, to the variable in the calling program.

To create a Selection List program:

1. In the **Program** repository create a new line and enter a name for the new program.
2. Zoom to open the **Task Execution** repository and press CTRL+P to open the **Task Properties** dialog box.
3. Set the **Task type** as **Online**, and from the **Main table** field zoom to select the table you want to be available for selection.
4. Click the **Advanced** tab and set the **Selection table** property to **Yes**. You can also set an expression evaluating to true in runtime. Then click **OK**.
5. In the **Record Main** task set the **Operation** as **Select** and choose **Parameter**. The parameter should match the value that should be returned to the calling program, both in type and in picture. Usually, this value is also the unique key of the table.
6. Select the corresponding real variable, which is usually the code, and any other relevant variables from the **Main table** for display purposes.
7. From the **Task** menu choose **Forms** to open the **Form** repository. Zoom to open the **Form Editor**.
8. On the form add a **Push Button** from the **Control** palette. In the button's property sheet zoom from the **Raise Event** property to define an **Internal** event type and **Select** event for the button. On the form add another button and in this button's property sheet zoom to select an **Internal** event type and **Exit** event for the button. You can also define a virtual variable in the program or use the built-in mechanism of enter=select and escape=exit.
9. In the **Record Suffix** level, update the parameter with the value of the respective real field.

When you call the program, ensure that you pass the variable for which the value is supposed to be returned as a parameter.

Defining a Selection Table Program

A Selection Table program includes as its root task a regular online task, which browses the Main table containing the values to be selected by the end-user.

1. Open the **Task Properties** dialog box and click the **Advanced** tab.
2. Select **Yes** or a logical expression that evaluates to True for the Selection Table field.
3. Define a Parameter that will return the selected value.
4. Insert an **Update Var** operation in the Record Suffix to update the parameter with the relevant column value.

Associating a Selection Table Program to a Column

eDeveloper lets you associate a Selection Table program to a column. To do this, specify the Select Program and the Select Program Call parameters at these locations:

- Model Repository
- Column Repository
- Program Field Properties
- Form Control Properties

If you associate a Selection Table program with a Class type, eDeveloper automatically associates it to all the columns using that type and to their Control properties on the screen forms. You can override this setting in the Column Properties dialog box or in the Control Properties dialog box of the relevant columns.

If you associate a Selection Table program to a column in the Column Properties dialog box, eDeveloper automatically associates it to all the occurrences of that column in screen Edit controls. You can disable a program associated to a column, by setting Select Program to No in the Control Properties dialog box.

Model Repository

1. Open the **Model** repository.
2. Press CTRL+P to open the **Model Properties** sheet.
3. Open the **Input** section on the **Categorized** tab and park the cursor on **Selection program**.
4. Zoom to the **Program List** and select a program.
5. For the Select mode, specify how the program will be called, depending on the column type you want to associate it with.

Column Repository

1. Zoom from a column in a **Table** repository and press CTRL+P to open the **Column Properties** sheet.
2. Follow **steps 3-5** above.

Program Field Properties

1. Zoom from the **Select Virtual** operation and press CTRL+P to open the properties sheet.
2. Follow **steps 3-5** above.

Form Control Properties

1. Zoom from the **Form List** to display the **Control palette**.
2. Select any control, place it on the form, and press CTRL+P to open the properties sheet for that control.
3. Follow **steps 3-5** above.

Executing a Selection Table Program

To use a Selection Table program, you normally associate it with a column.

When a Selection Table program is associated with a column, it automatically becomes a zoomable column.

If the Select type is set to Prompt, the select program is activated automatically on entering the column. After the program is terminated, it is still accessible by pressing **F5**.

At runtime, when the end-user zooms while parked on this column, or when the insertion point first enters the column if the Select type is set to Prompt, the following happens:

- The Selection Table program associated with the column is activated, and the value of the zoomable column is passed as a parameter to it.
- The engine executes the Selection Table root task as a regular online task, but its termination rules are different:

If the end-user presses Select, the engine:

1. Terminates the Record Main.
2. Executes the Record Suffix executing the Update Var operation.
3. Exits the task, displaying the picked value in the calling task's zoomed column.

If the end-user selects Exit, the engine terminates the task without picking a value.

The positioning of the insertion point, when the Selection Table program ends, depends on the setting for the Select Program Call property associated with the Select Program property of the zoomable column:

- If the Select type is set to Before or Prompt, the engine leaves the insertion point on the zoom column.
- If the Select type is set to After, the engine moves the insertion point to the column that follows the zoom column in the Operations repository, regardless of its position on the screen.

Defining a Context Menu for a Program

eDeveloper provides you with two standard menus. In the Menu repository you can define additional context menu structures for your application. You can attach the additional context menus to a program in the Task properties setting of a program.

Note: The **Attached context** property is only relevant for online tasks.

This section includes the topics listed below:

- Defining an Additional Context Menu
- Defining the Context Menu in a Program or a Task

Defining an Additional Context Menu

To define additional context menus:

1. In the **Menu** repository create a new line and enter a name for your new menu.
2. Zoom to open the **Menu Definition** screen and create a new line.
3. In the **Entry Type** column select an entry from the **Entry Type list**.
   - For a **Program** type, zoom from the **Menu Params** column to select the program to associate with the menu.
   - For an **OS Command**, zoom from the **Menu Params** column to select the required OS command.
   - For an **Event**, zoom from the **Menu Params** column to select the required System, Internal or User Event.
   - For a **Separator**, you only need to select **Separator** from the **Entry Type** list.
   - For a **Menu**, zoom from the **Menu Params** column to create the required submenu.

Defining the Context Menu in a Program or a Task

To define the context menu in a program or task:

1. Zoom from the program to which you wish to connect the new menu.
2. Press CTRL+P to open the **Task Properties** dialog box and click the **Advanced** tab.
3. From the **Attached context** field, zoom to open the **Menu List**.
4. Select the desired menu and click **OK**. The number of the menu entry appears in the **Attached context** field.

At runtime, when the end-user invokes a context menu while the particular task is running, the menu defined for this task is the one that will be displayed, rather than the default context menus for the application.

Automatically Saving Changes to the Dataview

In eDeveloper, a record is only written to the database, modified, created or deleted, when the engine goes through the Record Suffix.

With batch tasks, every record automatically goes through the Record Suffix. In online tasks the engine only reaches this level automatically if the user has modified one or more real fields. If the user does not modify, there is no need to update the record and go to the Record Suffix.

You can however set the engine to go through the Record Suffix level, even if the user has not modified any fields.

This section includes the topics listed below:

- Using the Task Control Property: Force record Suffix
- Defining a User Event with Force Exit=Record
- Defining a User Event with Force Exit=Record

Using the Task Control Property: Force record Suffix

One way to get the engine to go through the Record Suffix level, even if the user has not modified any fields, is to:

1. Zoom from the relevant program to open the **Task Execution** repository.
2. From the **Task** menu choose **Task Control** to open the **Task Control** dialog box.
3. Click the **Behavior** tab and set the **Force record suffix** property to **Yes**. You can also zoom to enter an expression that evaluates to a logical 'True'.

Defining a User Event with Force Exit=Record

Sometimes you need a record to be written before the handler is activated. Therefore, the engine should go to the Record Suffix and modify the record in the database before handling the operations in the handler.

By defining a user event with **Force Exit=Record** the engine is forced to go to the Record Suffix as soon as the event is triggered.

1. Zoom from the relevant program to open the **Task Execution** repository.
2. From the **Task** menu choose **User Events** to open the **User Events** repository.
3. Create a new line and enter a name for the event in the **Description** column.
4. Set the **Force Exit** column to **Record**.

Updating an Operation with Undo=No

The Undo property determines whether the Edit/Cancel (F2) command is available to the end-user after the execution of the Update operation.

1. Zoom from the relevant program to open the **Task Execution** repository.
2. Create a new line in the **Operations** repository and choose the **Update** operation.
3. When **Undo** is set to **Yes**, the default, the end-user can use Edit/Cancel, after an **Update** operation has been executed, to reset the current record dataview to its

original state without saving the changes of the update.

4. When **Undo** is set to **No**, Edit/Cancel is disabled after the **Update** operation. The end-user will not be able to undo changes made to the record dataview at the task level of the updated variable.

You would generally use this "hard update" in the Record Suffix level, when resetting the updated variable to its original value could damage data integrity. For example, if the operation is intended to change data in the Record dataview of a parent task. If you set **Undo** to **No**, the user will not be allowed to cancel the parent record after the subtask's records have already been accepted.

Only one Update operation with **Undo** set to **No** in any one of the subtasks that is updating the parent record's dataview variable, is sufficient to prevent the user from undoing the update.

Reconfirming User Steps in a Program

Usually the only built-in confirmation screen in eDeveloper is the **Confirm Delete** message.

The following two confirmation messages can also be set using the **Task Control** dialog box:

- Confirm Update
- Confirm Cancel

To use these confirmation messages with a program do the following:

1. Zoom from the relevant program to open the **Task Execution** repository.
2. From the **Task** menu choose **Task Control** to open the **Task Control** dialog box.
3. Click the **Behavior** tab.
4. **Confirm Update**
   If you set this property to **Yes**, or zoom to enter an expression evaluating to **True**, eDeveloper confirms the Create and Modify operations.

You can also use an expression to override this behavior. For example, the following, which will confirm Create operations only:
STAT(0,'C'MODE)

5. **Confirm Cancel**
   If you set this property to **Yes**, or zoom to enter an expression evaluating to **True**, eDeveloper prompts the user on cancellation of updates in the current record. Confirming the cancellation cancels the changes and the record reverts to its original values. Declining the cancellation saves the new values in the records.

Building an Interactive Web Task

6

This chapter discusses questions relating to some of the new interactive Web application features of eDeveloper Version 9.

This chapter covers the topics listed below:

- Defining the Interactive Web Application Developing Environment
- Creating a Simple Interactive Web Application Program
- Preparing an HTML Template
- Handling HTML Controls
- Defining an HTML Table Control in a Browser Task
- Increase Record Scrolling Performance
- Creating a Browser One-to-Many Relationship of Tasks
- Working with a Third Party HTML Authoring Tool
- Opening a Browser Task in a Specific Frame
- Changing the Task Mode in a Browser Client
- Defining and Using the eDeveloper VCR
- Setting eDeveloper to Work with the Persistent Client Mode
- Defining a Subform
- Opening Two Browser Programs in the Same Frame

Defining the Interactive Web Application Developing Environment

Several elements are important to this configuration, mainly the eDeveloper engine, a browser, a Web server, Internet requesters, and the eDeveloper Broker.

The procedures below explain how to construct a working environment on a single machine. In general the modules that take part in this configuration may reside on different machines.

The interactive Web application paradigm requires an installed browser. You use the browser to connect to the eDeveloper application server. It serves as the interface for the browser tasks.

You must install a Web server in order to permit the browser to contact the eDeveloper application.

Configure this Web server with eDeveloper Internet modules.

A TCP\IP protocol is required on your machine. The prerequisites are:

- Installation
- Setting up the Web server
- Setting up the Internet requesters
- Running the eDeveloper Broker
- Setting up the eDeveloper engine
- Testing the configuration

This section includes the topics listed below:

- Installation
- Setting Up the Web Server
- Setting Up the Internet Requester
- Running the eDeveloper Broker
- Setting up the eDeveloper Engine

Installation

In the typical eDeveloper installation, eDeveloper automatically configures your environment to enable an interactive Web application to run.

In order to customize your installation you need to install the following modules:

- The eDeveloper Engine
- The eDeveloper Broker
- The Internet requesters

Setting Up the Web Server

To set up the Web server:

1. Define a virtual directory in your Web server.
   This directory's path leads you to a directory in which the following files exist:

   Mgrqgnrc94.dll

   Mgrqhttp94.dll

   Mgrqispi94.dll **(if you are using a Microsoft Web server)**

   Mgrqcgi94.exe **(if you are using a different Web server)**

   MGBC940_02.js

   MGBC940_02.cab

   MGBC940_02S.cab

   Mgreq.ini

   In the typical eDeveloper installation, all of the required Internet modules are in the working directory. The virtual directory points to the working directory of eDeveloper.

2. Give this virtual directory an appropriate name (alias).
3. The typical eDeveloper installation calls this directory **magic9scripts**.
4. Enable this virtual directory for **Read** and **Execute**.

Setting Up the Internet Requester

To set up the Internet requester:

1. Direct the Internet requester to the eDeveloper broker you are using.
2. Indicate the broker address and port in the **MessagingServer** setting in the Mgreq.ini file.
   The default value for this setting is 3001, meaning port 3001of the same machine.
3. If your eDeveloper Broker is loaded on a different machine you should define this setting with the machine name and port number.
   For example, `MessagingServer=Machine_name\3001`.
4. The value of the **BrokerPort** setting in the Mgrb.ini file in the working directory of the broker indicates which port your broker listens to.

Running the eDeveloper Broker

Run the **eDeveloper Broker** prior to running the **eDeveloper engine**. You can use the default setting of the eDeveloper Broker.

Setting up the eDeveloper Engine

To set up the eDeveloper engine:

You must define the eDeveloper engine before it is loaded as an application server, as described below.

1. Run the **Magic.exe** file.
2. Define the setting of the eDeveloper Broker with the **Server** option on the **Settings** menu.
   You may use the default setting provided by the typical installation of the server entry name Default Broker.
3. Define the following settings on the **Enterprise Server** tab of the **Environment**

option on the **Settings** menu:

- Set the **Activate as Enterprise server** to **Yes**. (The default is Yes.)
- Zoom from **Messaging Server** to select the server definition for your eDeveloper Broker. (The default is Default Broker.)
- In the **HTTP Requester** setting define the location and the name of the requester you are using relative to the Web server. This is set to **magic9scripts/Mgrqispsi.dll**, if a Microsoft Web server was detected on your machine.
- In the **Web Document Alias** setting, define the path where the browser client module files (Mgbrowserclient.jar, Mgbrowserclient.js) relate to the Web server. This is set to **magic9scripts/**, if a Microsoft Web server was detected on your machine.
- In the **Web Authoring Tool** setting, zoom to select your HTML file editor.

4. Run the eDeveloper engine with the above settings and make sure that the status bar indicates **App Server**.

To run browser tasks:

1. Create a browser task from the **APG** on an existing table.
2. Select the **Generate** mode and the **Browser Client** options. Return to the Program repository and execute the newly created browser task.
   The default browser opens, the toolkit switches to runtime, and an implicit request is sent to the engine through the requester to run the browser task.

To call a program in runtime from a browser:

Construct a URL according to the following example:

```
Http://[your_machine][your_requester_location]
[requester_name]?appname=
[application_name]&prgname=[program_name]
```

Creating a Simple Interactive Web Application Program

This topic explains how to manually create a simple running interactive Web application on the query task. You can also perform these steps using the Automatic Program Generator.

This section includes the topics listed below:

- Building an Interactive Web Application Task
- Defining the Properties for the Interactive Web Application Task
- Defining HTML Controls

Building an Interactive Web Application Task

To build an interactive Web application task:

1. Before starting to build your interactive Web application make sure that you have the required configuration setup for interactive Web application tasks.
2. Create a table on which you wish to run the Browser task.
3. Open the **Program repository** and create a new program. Zoom to the new program and in the **Task Properties** dialog box set the **Task type** to **Browser** and select your table as the Main table.
4. Define the dataview for this task by selecting the fields of the Main table.
5. Open the **Form repository**.
6. Press **CTRL+P** to open the property sheet of your main class 0 Browser form.

Once you have completed these steps you need to define the task properties. This procedure is explained in the next section.

Defining the Properties for the Interactive Web Application Task

To define the properties for the interactive Web application task:

1. On the property sheet, set the following properties:
   - **Repeated lines** - define the number of rows of data you wish the table to have.
   - **HTML file** - the interface you create is in HTML format and should be stored in an HTML file.
2. Define the name of the HTML file in which you store the HTML information. For new tasks, you usually define a name of a new file.
3. Zoom into the Form.
   eDeveloper prompts a message stating that this file does not exist and that a new file will be created using the default HTML template.
4. Click **OK** to create the file and open your preferred Web authoring tool.
   You may now create a table to present the data in line mode.
5. Using your authoring tool, create a table of two rows. The Table control should be set with an ID attribute to be identified by eDeveloper. If your authoring tool does not support the ID attribute you should place it yourself in the HTML script as follows:

```
<table border="1" width="500" id="Table">.
```

The ID may be any descriptive name you choose.

1. In the first row write the name of the columns.
2. In the second row, place a control for each variable you want in each column.
3. The controls for the variables can be placed automatically by selecting and dropping a variable from the floating Variable palette, or, you can create the controls yourself.
4. If you create the controls, first insert them into the form using the authoring tool, and provide a unique descriptive name for each control.
5. Save your page and return to eDeveloper.

Defining HTML Controls

To define the HTML controls:

1. In the HTML Control repository, specify each object including the Table and attach the variables to each control.
   All new objects on the HTML page can be viewed and selected from the new HTML controls list.
2. Click on the **New HTML tags** button to view this list.
   This list displays the table control and the other controls you placed with the names they were given.
3. Select the controls you created including the table control. All controls will be added to your HTML Control repository.
4. Create new lines and define new controls by giving them names, and define their type in the **Type** column.
5. Associate the corresponding variable for each control. You can do this by zooming from the **Var** column to see the available variables and select the appropriate variable for each control. You can also type in the identifier code of the variable.
   The most important property to define for the table control is the **Detail line #**. In this property set a numeric value that indicates the number of the line to be repeated. This line will be repeated a number of times as defined in the **Repeated lines** property of the form.
   The basic browser task is complete.
6. Press F7 on the program in the Program repository to execute it.

Preparing an HTML Template

eDeveloper can easily generate browser tasks automatically. In this case, eDeveloper creates the dataview and the interface of the browser task. The interface, which is an HTML file attached to the task's browser form, can be based on a pre-defined HTML template.

Using such a template, the automated browser forms can be based on a common template. This gives the browser a uniform appearance.

Note: eDeveloper can also automatically generate just the browser form interface for an already existing browser task.

This section includes the topics listed below:

- Creating the HTML Template
- Using the HTML Template File

Creating the HTML Template

To create the HTML Template file:

1. Include the following basic HTML tags (HTML, HEAD, BODY and FORM).

   ```
   <HTML>
   <HEAD>
   </HEAD>
   <BODY>
   <FORM NAME = “”>

   </FORM>
   </BODY>
   </HTML>
   ```

   This HTML file is sufficient to be used as an HTML template file. Actually, when you do not specify your own pre-defined template file, eDeveloper uses this template to create the HTML files for the browser task interface.
   Using this basic template file you can now add any other HTML based tags and properties to create a desired look. You may edit it using your preferred HTML authoring tool.

2. Define a background color for the entire HTML page using your preferred HTML authoring tool.
   You can place an image of a reserved name that will display eDeveloper's VCR toolbar that allows you to browse easily through the dataview.

Using the HTML Template File

The following procedure presents an example of where and how you can direct eDeveloper to use the new HTML template.

To run the Automatic Program Generator (CTRL+G) on a table:

1. Set the **Mode** field to **Generate**.
2. Set the **Option** field to **Browser Client**.
   eDeveloper provides a default name for the HTML file. This file keeps the interface for this task. You may also change the file name.
   If you have not used your own template file yet, the Template file name field is blank.
3. To use your own HTML template file, enter the path and name of your HTML template file in this field
   -OR-
   zoom to the **File Open** dialog box to select the template file.
4. Click **OK** to confirm the **APG**, in order to create the interface based on its internal template.
   Running the **Result** program displays the task with the added new options on the page.

Note: eDeveloper remembers the last template defined. You may remove the file name and eDeveloper will use its internal basic template.

Handling HTML Controls

The interface of the eDeveloper browser task is kept in an external HTML file.

Although the entire interface is external to the browser task, eDeveloper provides the necessary means to view, define, and handle every supported input control that is defined in the HTML file.

This section includes the topics listed below:

- Defining HTML Controls
- Creating Controls with the APG
- Dragging and Dropping Variables
- Adding an HTML Control
- Assigning Data to HTML Controls
- Defining HTML Control Properties

Defining HTML Controls

To define the HTML controls:

1. Zoom to the browser task form. The HTML controls table opens.
2. Define all the controls that reside on the HTML interface definition and handle them as if they were eDeveloper controls.

Column	Purpose
HTML control name	The name of the control as it is defined in the HTML file.
Exp	An Expression that in runtime evaluates to provide a dynamic name of the HTML control as it is defined in the HTML file.
Type	A combo box that defines the type of HTML control. This is required for the eDeveloper toolkit so it will provide you with the corresponding property sheet.

Var	The letter code of the variable attached as data for this HTML control.
Exp	The number of the expression attached as data for this HTML control.

Creating Controls with the APG

If you created your browser form using the APG utility, then eDeveloper has already created the HTML controls in the provided HTML file name and defined these controls in the HTML Control repository.

Dragging and Dropping Variables

If you dragged and dropped a variable on the HTML file (using the authoring tool) then eDeveloper automatically creates a corresponding entry in the HTML Control repository.

Adding an HTML Control

You can handle an HTML control that exists on the page.

To add an HTML control:

1. Click on the **New HTML tags** button to display the list of available HTML controls.
   This list shows all the HTML controls that have not yet been entered on the HTML Control repository.
   This list also shows the names of these controls and their types.
2. Click **Select** to add the control you are parked on.
3. You can also multi mark several controls you wish to add and then click **Select**.

Assigning Data to HTML Controls

To assign data to HTML controls:

1. Zoom from the **Var** column to select a variable to assign to the HTML control (or type in the variable letter code).
   This allows you to make the HTML control serve as the editing field of the assigned variable.
2. Assign an expression to be used as the control's data.
   This control will not be enabled for editing.

Defining HTML Control Properties

For each defined HTML control in the HTML Control repository, you can open its property sheet to define its properties values.

Note: Properties that are basic HTML properties, such as color, font, or size, can be assigned only with an expression.

For such properties eDeveloper provides the dynamic values, or expressions, whereas the authoring tool is used to set the initial or fixed value of these properties.

Defining an HTML Table Control in a Browser Task

eDeveloper enables you to easily define an HTML Table control to present your data in line mode, where the data can be easily scrolled.

This section includes the topics listed below:

- Creating the HTML Table Control
- Setting the Table's Details Line # Property

Creating the HTML Table Control

To create HTML table and input controls in a browser task:

1. Create the HTML table control using your preferred HTML authoring tool.
2. Give the table an ID recognizable by the eDeveloper toolkit and runtime engine. For example:

   ```
   <Table border="1" ID="My_Table width="100%">
   ```

3. Place the column titles in the table row you designate as the column title row.
4. Define the input and display controls of the represented data in the table row you designate as the repeated data line.
   You may also use eDeveloper's Variable palette to drag and drop variables onto your HTML table.
5. In the **HTML Control** repository, define the input controls you have just added and associate them with the variables of your task.
   If you dropped these controls using the **Variable** palette then they are automatically added to the **HTML Control** repository.
6. Click on the **New HTML tags** button to view the remaining HTML controls. Among them you see the table control you created.
   The table control appears by the name given to it in the ID attribute of the table HTML tag.
7. Select the table control to add it to the task's HTML controls.

Setting the Table's Details Line # Property

To set the table's *Details line* # property and define the number of lines to be displayed:

1. Open the table property from the HTML Control repository.
2. Define the row number you designated to be your repeated data line.
3. Enter the number of this line in the **Details line #** property.
   Any line above the Details line will be considered as the table header.
   Note: The table header cells may contain data bound controls. Such controls will reflect the value of the current record but will be shown only once for each current record.
4. Specify the desired number in the **# of repeated line** property in the Form property sheet.
   Using the defined number, eDeveloper automatically repeats the designated detail line in the number of times as defined by this form property. Any line below the detail line will be considered as the table's footer.

Note: The table footer cells may contain data bound controls. Such controls reflect the value of the current record, but are shown only once for each current record.
If this form property is set to Zero, eDeveloper repeats the detail line on top of the remaining rows below the detail line. This action may override any other content of the remaining lines.

Increase Record Scrolling Performance

The browser task of eDeveloper allows easy scrolling of the data viewed on the browser. The newly displayed records need to be fetched from the database through the eDeveloper application server.

Numerous interactions with the eDeveloper application server may hinder the performance of the browser task. In order to improve the performance of the browser task while browsing the through the records, the user must:

- Provide the browser client with a greater amount of record than it initially displays, and
- Diminish the number of times the browser clients address the application server.

This section includes the topics listed below:

- Defining the Chunk Size Number
- Resetting the Cache

Defining the Chunk Size Number

The Chunk size expression property enables you to define how many new records will be sent to the browser client upon every request for new records as a result of scrolling.

To define the chunk size number:

1. In a browser task, open **Advanced** tab of the **Task Properties**.
2. In the **Chunk size expression** property, use an expression to define a numeric expression that will define the chunk size.

This expression is computed once, upon the initial phase of the browser task.

If, for example, the Chunk size is evaluated to 100, then eDeveloper provides the client with the first 100 records of the defined view, regardless of the number of displayed rows.

In this case, the client browses through the first 100 records locally since it keeps the records it receives in its local cache of records.

If the user scrolls over the first 100 records, then the browser client automatically issues a request to the application server for the next chunk of records.

This new chunk of records is added to the client's cache of records. This allows for local back-scrolling through the 200 records.

Resetting the Cache

Upon non-sequential scrolling, such as Begin Table and End Table, the client clears its cache of records and adds the corresponding new chunk of records.

Creating a Browser One-to-Many Relationship of Tasks

The HTML that is produced can represent, in one interface, several eDeveloper tasks. Each task handles different parts of the entire HTML interface.

This situation is very common when you create a one-to-many relationship view, where one task displays and handles the parent view and another task, or tasks, handles the extended view for each record of the parent.

The eDeveloper subform control allows you to include two separate tasks in one HTML interface definition file to handle two separate dataviews.

This section includes the topics listed below:

- Creating Parent and Extended View Tasks
- Defining Subform Controls and Properties
- Recomputing the Subform

Creating Parent and Extended View Tasks

To create the parent and extended view tasks:

1. Create the parent view task by defining a browser task to display the parent's dataview, the **One** part.
2. Create the extended view task or subtask of the browser task you have just created that will show and handle the extended dataview (the **Many** view). In creating this second program you should follow the following guidelines:
   Use the same file name as the parent's browser form for the HTML file name that the browser form uses.
   Add all the HTML controls (including the table control - if required) of this program to the existing HTML file, in any location you choose.
3. Verify that the newly added HTML controls of the second program do not have the same name as the controls of the parent task. If there are controls with the same name, you must provide a new unique name to these controls and update the new names in the HTML Control repository.

4. Create **Select Parameter** operations for the required parent details that are needed to range the view, and initialize the corresponding variables.

Defining Subform Controls and Properties

To create subform controls and set subform properties:

1. Return to the parent task. Zoom to the browser form to open the HTML Control repository.
2. Create a new entry in the HTML Control repository.
   This entry is your subform control. You may provide a descriptive name for this subform control.
3. Define the type of this entry to be **Subform**.
4. Open the property sheet of the subform and define the following properties:

   a) **Connect to** - Define the type of task you created as the extended view task - program or a subtask.

   b) **PRG\TSK num** - Define the number of the program or task of the extended view. You may also zoom in to select it from a selection list.

   c) **Arguments** - Zoom to the Arguments list to pass the required arguments.

   These arguments will be retrieved by the parameters defined in the extended task to range the dataview and to initialize the corresponding variables.
5. The One-to-Many browser task is completed, and you can run the parent task.

Recomputing the Subform

The arguments passed to the extended task are used not only by the parameters that get their values but also as recomputing agents of the subform.

Whenever one of the passed arguments is changed, the values of the arguments are re-passed to the extended task and its view is refreshed to display the new view.

Whenever the subform is recomputed and the view is refreshed, the Task Prefix and Task Suffix of the extended view task are activated. The focus, however, remains at the parent's view.

Note: Only arguments that are passed as variables may cause a view refresh of the extended task. Passed expressions will not cause a view refresh of the extended task.

Working with a Third Party HTML Authoring Tool

The interface of an eDeveloper browser task is defined and edited by a third party HTML authoring tool of your choice.

This section includes the topics listed below:

- Defining Your HTML Authoring Tool
- Editing the Browser Task Interface HTML File
- Editing the HTML File
- Editing the HTML File Externally

Defining Your HTML Authoring Tool

eDeveloper enables you to define your own preferred authoring tool.

To define your preferred HTML authoring tool:

Enter its execution file name and location in the **Web Authoring Tool** environment setting under the **Enterprise Server** tab.

Note: The installation process looks for a default HTML authoring tool. The name and path of the found tool is automatically entered in the eDeveloper environment setting.

Editing the Browser Task Interface HTML File

To edit the browser task interface HTML file:

1. Zoom to the browser form entry.
   The HTML Control repository opens. This repository lists all HTML controls that are handled by the browser task.
2. Click the **HTML Editor** button to activate the HTML authoring tool.
   The authoring tool opens.
3. Edit the HTML file defined for this form entry.

Editing the HTML File

You can edit the HTML file after opening the authoring tool.

To edit the HTML file:

1. Define any object you create that you wish to use as a control in eDeveloper with a name (NAME=XXX).
2. Identify any object you create, such as a table, which does not support the NAME attribute using the ID attribute.
3. You should create\keep only one FORM object on your HTML page.
   If you add a new control that you wish to handle from the browser task, you should add this control, by its name, to the HTML Control repository.
4. You may edit your HTML file to include Cascade Style Sheets (CSS), Java Script, Java Applets, Active-X, DHTML for an enhanced interface.

To drag and drop variables:

1. Activate the **authoring tool**.
   The **Variable** palette opens floating on top of the authoring tool.
2. From this palette, drag and drop a variable onto your HTML file.

When dropping a variable, eDeveloper creates a control that was defined for the variable in its Style\Browser property. This control receives the variable's name. If a control by this name already exists, the given name is concatenated by a number. eDeveloper also automatically adds a corresponding entry in the HTML Control repository assigned with the corresponding variable.

Note: Once a variable is assigned to a control in the HTML Control repository, it cannot be dropped. The drag-and-drop capability is only available for authoring tools that support OLE Drag and Drop, such as Microsoft® FrontPage®. Therefore, it does not support Microsoft® Notepad or Macromedia® DreamWeaver®.

Editing the HTML File Externally

Although eDeveloper provides a smooth and transparent integration with the authoring tool of your choice, and since the HTML file is kept as an external file that is accessible regardless of Edeveloper, you can easily create your HTML file directly through any HTML authoring tool.

Opening a Browser Task in a Specific Frame

The user can direct the opening of a client of a called browser task in a different window than the window in which the calling task opens. The given window destination can also be a different frame in a frame set. You can direct a browser task to be opened in a different window or frame by providing the window or frame name in the Destination property of the Call operation.

To open the task:

1. Create a **Call operation** entry in your main browser task.
2. Define the number of the browser task you wish to call in the **Task Operations** repository.
3. Open the property sheet of the Call operation and in the **Destination** property define the name of the window or frame that you want this task to be opened in.
4. Run the task.
   Whenever the Call operation is executed, the new browser task opens in the given window or frame according to its name.

Changing the Task Mode in a Browser Client

A browser task maintains the basic mode rules of the Modify, Create, Query, and Delete modes that are available for a browser task. In each mode the browser client functions according to the rules of the defined mode. For example, in Query mode, data entry and data deletion are not allowed, whereas in Modify mode they are allowed.

The event type is Internal and it can be one of the following:

- Modify Records
- Create Records
- Query Records

You set the browser task's **Initial mode** in the Properties tab of the Task Properties dialog box.

1. From a program, zoom to open the **Task Execution** repository.
2. Press CTRL+P to open the **Task Properties** dialog box.
3. In the **Initial mode** property click Modify, Create or Query.

Note: You can also change the task mode by raising an event. The Raise Event operation should have the **Wait** property set to **No**. The trigger for the event can be a button or a Hot Key.

If the event is used in more than one program it is best to create a handler in the Main Program and activate the trigger from all the different tasks.

Defining and Using the eDeveloper VCR

A VCR control is a VCR panel image that provides the end user with easy navigational functionality for the dataview. The panel is for use with Web applications.

The VCR image shown below, is supplied as a .jpg file with eDeveloper. The file name is Mgvcr.jpg.

To insert the VCR image into your HTML page:

1. Using your HTML authoring tool create an HTML image tag.
2. Make sure that the source of the image tag points to the **Mgvcr.jpg** file. The name should be **MG_VCR**.
   For example:
   `<IMG SRC = ”/Magic9Scripts/mgvcr.jpg” name = “MG_VCR”>`
3. The Browser client identifies this image by its name and handles it automatically.

You can also create your own version of the VCR image. If you decide to do this and not use the one supplied with eDeveloper, you should ensure your image contains six items corresponding to the eDeveloper VCR functionality.

1. Enter the following tag:
   <HTML>
   <HEAD>
   <title></title>
   </HEAD>
   <BODY bgcolor="#00FFFF">
   <FORM NAME = “”>

<img name="MG_VCR" src="\magic9scripts\MGVCR.JPG">

   </FORM>
   </BODY>
   </HTML>
2. When eDeveloper encounters an image on the page with the name of “MG_VCR,” it treats it as a VCR toolbar. The browser client internally divides this image into six vertical parts where each part is used for one of the navigation

options (from the left):
Begin Table
Previous screen
Previous Row
Next Row
Next Screen
End Table

3. Click on each part of the image to perform the corresponding navigation action. This image file name should be defined in the Image source file, in a path that is recognized by the Web server.

This template can now be used by eDeveloper to create new browser interfaces when created using the Automatic Program Generator.

Setting eDeveloper to Work with the Persistent Client Mode

The Java applet, which is the main part of the browser client engine module, needs to be loaded by the browser client. With certain browsers this can take time. The eDeveloper Enterprise Server enables you to make the Java applet module persistent on the client side. This means that once the applet is loaded for the first time, it remains on the client machine, and every browser task requiring this module, takes it from the client.

To define your application server to work with a persistent applet module do the following:

1. From the **Settings** menu choose **Environment**.
2. Click the **Enterprise Server** tab.
3. Set the **Persistent Browser Client Module** property to **Yes**. Setting the value to **No** disables this option.

When you set the property to **Yes**, the end user must provide confirmation in the Browser Confirmation dialog box.

Note: The Browser Client Java module has been certified and digitally authenticated by VeriSign. This assures the end user that the module has not been tampered with.

Defining a Subform

You can use a subform when the browser client is designed to display extra data on top of the main view, when the extra data usually consists of more than one record. This is usually referred to as a one-to-many relationship. In such a case there is usually more than one task involved, and the Subform control enables the two separate tasks to be displayed on the same HTML interface.

To display two subtasks on the same HTML page do the following:

1. Design your HTML interface file to include the required HTML elements for both the main task and the descendant task.
   Note: The names and IDs of the elements of both tasks should be unique throughout the entire HTML page.
2. Create two browser tasks, a parent task and a child task, and define the **HTML file** property of the browser form in both tasks to point to the same HTML interface file, that was created previously.
3. In each task define the relevant HTML controls, including the table control of each task, if used.
4. Create an extra **HTML control** in the main task, and set its type as **Sub form**. Set the property of this control to call the child task.
5. Use the **Arguments** property to pass required data to the child task. Usually the passed arguments will be the variables responsible for ranging the data of the subform.
6. Run the main task.

Opening Two Browser Programs in the Same Frame

It is common to use a frameset when developing Internet-based applications, making it easier to maintain the user interface. Usually a screen is divided into several different frames including a navigator, the main working screen and a banner frame.

The procedure below explains how you can enable one browser program to call another, and open the called program within the frame containing the calling program.

You should create a simple HTML file with a frameset. The following is an example taken from the eDeveloper demo:

```
<html>
<head>
<title>Query</title>
</head>
<frameset rows="100,*"frameborder="0" framespacing="0"
border="0">
<frame name="NavigatorFrame"src="/Magic9Demo/
tf_start.html" scrolling="no" marginwidth="0"
marginheight="0" noresize frameborder="0">
<frameset cols="122, *" frameborder="0" framespacing="0"
border="0">
<frame name="LogoFrame" src="/Magic9Scripts/
mgrqispi9.dll?APPNAME=Magic_9_Demo&PRGNAME=prog_A"
scrolling="no" marginwidth="0" marginheight="0" noresize
frameborder="0">
<frame name="MainFrame" scrolling="yes" marginwidth="0"
marginheight="0" noresize frameborder="0" src="/
Magic9Scripts/Magic9Demo/HTML_Pages/Welcome.html">
</frameset>
<noframes>
</noframes>
</frameset>
<frameset>
</frameset>
</html>
```

The frameset example above contains two frames displaying a simple HTML file and the frame calling the eDeveloper browser programs.

In this example, **prog_A** is the program in the MainFrame, and it contains a button calling **prog_B**, the new program also to be opened in the MainFrame.

1. In the application's **Main Program** press **F4** to create a new **Handler**. From the Task menu choose **User Events** to open the **Events** repository. In the

Description column enter the name **Call prog_B** and from the **Type** list select **Internal**. Click **OK**.

2. In the **Handler**'s **Operation** repository press **F4** to create a new operation. From the **Operation** list select **Call** and set it to **Prog**. Zoom from the **0** property to select **prog_B**.
3. Press **CTRL+P** to open the **Call Properties** dialog box. In the **Destination frame** field enter **MainFrame**. Click **OK**.
4. In **prog_A** create a **Handler** for the button. From the **Event** column zoom to open the **Event** dialog box.
5. From the **Event type** list select **Internal** and from the **Event** zoom to the **Action List** and select **Click**. Click **OK**.
6. In the **Operation** repository press **F4** to create a new operation. From the **Operation** list select **Raise Event**.
7. From the **Name** column zoom to open the **Event** dialog box. From the **Event type** list select **User** and from the **Event** list select the Event **Call prog_B** that you created in the **Main Program**.

Using Transaction and Recovery Techniques

7

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 completed and written to disk or aborted as a single unit. Reservation and credit-checking operations are typical examples of such transactions.

This chapter covers the topics listed below:

- Working with Deffered Transactions
- Continuing a Program When the Transaction Fails
- Dynamically Logging into a Database
- Handling a Violation of the Database Constraint by a User
- Sending Nulls by Expression
- Working with MS-SQL Temporary Tables
- Committing a Transaction Every nn Records

Working with Deffered Transactions

Transaction processing is used to preserve data integrity. eDeveloper stores all Data Manipulation statements within the deferred transaction in a cache.

The statements are sent to the database in one step according to the Transaction mode.

This type of transaction handling shortens transaction time, increases performance, but also can reduce consistency.

There are 3 Transaction modes for Deferred Transactions:

- *Before Record Prefix* - Updates are accumulated at the Record level before being sent to the database in a transaction after Record Suffix.
- *Before Task Prefix* - Updates are accumulated at the Task level before being sent to the database in a transaction after Task Suffix.
- *Group* (valid in Batch tasks with Group Level only) - Updates are accumulated at the Group level and sent to the database in a transaction when a variable changes its value.

This section includes the topics listed below:

- Defining the Transaction Begin Property as Before Record Prefix
- Defining the Transaction Begin Property as Before Task Prefix
- Defining the Transaction Begin Property as Group
- Handling Deferred Transaction Errors

Defining the Transaction Begin Property as 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
- Includes all operations between the open and the commit, including subtasks, within the transaction scope.

To define the Transaction Begin property as Before Record Prefix:

1. Open the **Program** repository.
2. Create a new line and enter the program name.
3. Zoom to the **Task Properties** dialog box or Generate a Program from a table.
4. Select **Online** for the **Task type** on the **Properties** tab.
5. Click the **Enhanced** tab and select the following:
   Deferred for the **Transaction mode** field.
   Before record prefix for the **Transaction begin** field.
6. Click **OK**.

Note: When using eDeveloper's Monitor/Debugger, with the Gateways check box, all updates are sent to the database after the Record Suffix, in one transaction. Since 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.

Defining the Transaction Begin Property as 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.
- Includes all operations between the open and the commit, including subtasks, within the transaction scope.

To define the Transaction Begin property as Before Task Prefix:

1. Open the **Program** repository.
2. Open a new line and enter the program name.
3. Zoom to the **Task Properties** dialog box or Generate a Program from a table.
4. Select **Batch** for the **Task type** on the **Properties** tab.
5. Click the **Enhanced** tab and select the following:
   Deferred for the **Transaction mode** field.
   Before task prefix for the **Transaction begin** field.
6. Click **OK**.

Note: When using eDeveloper's Monitor/Debugger, with the Gateways check box, all updates are sent to the database after the Task Suffix in one transaction.

Defining the Transaction Begin Property as Group

1. Open the **Program** repository.
2. Open a new line and enter the program name.
3. Zoom to the **Task Properties** dialog box or generate a program from a table.
4. In the **Properties** tab, for the **Task type** select **Batch**.
5. Click the **Enhanced** tab and select the following:
   Deferred for the **Transaction mode** field.
   Group for the **Transaction begin** field.
6. Zoom from the **Var** field to open the **Variable list** and select the grouping variable.

Note: When using eDeveloper's Monitor/Debugger, with the Gateways check box, all updates are sent to the database in one transaction when the variable specified for the group changes.

Handling Deferred Transaction Errors

Each Deferred Transaction has its own Transaction cache.

If an error occurs when an Update operation is sent to the database when working with deferred transactions, you can use an error handler to access the cached row value at the time of the error.

Continuing a Program When the Transaction Fails

eDeveloper provides two pre-defined error behavior strategies:

- *Recover* – eDeveloper keeps the current dataview and remains on the current task where possible. Recover allows the end user to recover and continue working after an error.
- *Abort* – eDeveloper rolls back the current transaction (where the end-user cannot recover), removes the current dataview, and aborts the task in which the transaction occurs.

When eDeveloper default error behavior strategies do not apply, the Error Handling mechanism lets you overwrite eDeveloper's default behavior for different errors that can arise.

eDeveloper's Error Handling mechanism searches for a defined error handler to intercept the error. Once the error is found, you can control it and perform a user-defined operation such as Call Program or Verify, which can be defined in the error handler - and then eDeveloper performs the corresponding action, according to the Engine Directive property.

The Engine Directive property determines the action that will occur after execution of the error handler.

This section includes the topics listed below:

- Defining eDeveloper's Pre-Defined Error Behavior Strategies
- Defining an Error Handler
- Defining the Engine Directive
- Defining the Error Result
- Overwriting Database Errors
- Using eDeveloper Functions to Retrieve Error Information
- Defining the Any Error Handler
- Defining the Propagate Property

Defining eDeveloper's Pre-Defined Error Behavior Strategies

To define eDeveloper's pre-defined error behavior strategies:

1. Open the **Program** repository.
2. Create a new line and enter a program name.
3. Zoom to the **Task Properties** dialog box and click the **Enhanced** tab.
4. In the **Error Behavior** property, select the strategy you want: **Recover** or **Abort**.

Defining an Error Handler

To define an error handler:

1. Open the **Program** repository and zoom to the **Task Execution** repository.
2. Open a new line after the **Task Suffix**.
3. Select the **Handler** option for the **Level** column.
4. Place the cursor in the **Event** column and zoom to the **Event** dialog box.

5. Select the **Error** option for the **Event type** field.
6. Zoom from the **Event** field to the Error List and select the error you want to handle.
   The **Details** parameter on the Task screen changes to **Directive**.

Defining the Engine Directive

The Engine Directive property determines the action that will occur after the execution of the error handler and its operations.

There are 6 actions that can occur after the execution of the error handler:

- As Strategy – performs the action defined in the Task properties.
- Abort Task – aborts the task and rolls back the transaction.
- Rollback and Restart - retrieves the dataview's original values and causes eDeveloper to start again from the level that the transaction begins.
- Auto retry
- User retry
- Ignore – skips the error and continues to the next record.

To select the Engine Directive property:

1. On the **Handler** level of the **Task Execution** repository, place the cursor on the **Details** column.
2. From the drop-down list, select the action you want to occur after the execution of the operations.

Defining the Error Result

When the error handler is triggered with an error situation, it performs the operations that are defined for the handler.

To define the resulting action:

1. Place the cursor on the **Oper** column and zoom to **Task Operation** repository.
2. Open a new line and define the result of the error that you have encountered.

Overwriting Database Errors

eDeveloper displays the full error message whenever an error is encountered. The error message is the same message that is displayed in interactive SQL for the command in error. You use the *Display Full Messages* setting to control the display of the message.

To define whether or not to display the database error message:

1. On the **Settings** menu, select the **Environment** option.
2. In the **Preferences** tab, select the appropriate option for the **Display full messages** setting.

   Yes – eDeveloper displays the full error message.

   No – An Error message box is not displayed.

If you want to display your own message, the Error Handling mechanism lets you overwrite eDeveloper's default behavior for different errors.

When an error occurs, eDeveloper's Error Handling mechanism searches for a defined error handler to intercept the error.

Information about the error can be retrieved using different eDeveloper functions. Using this information, you can decide on the next operation in your program.

Using eDeveloper Functions to Retrieve Error Information

There are a series of functions that can be used to retrieve information regarding an error message: ErrTableName, ErrDatabaseName, ErrDbmsCode, ErrDbmsMessage, ErrMagicName, and ErrPosition.

When the error handler is triggered, it performs operations that are defined for the handler. In this operation, you can use these functions to display information about the error, or to use this information on the next step.

Defining the Any Error Handler

The **Any** error type can be executed for any error type that occurs. Once the error is captured, you can decide whether to move on to other error handlers in the lower levels of the task tree, using the Propagate property.

To define the 'Any' Error Handler:

1. Open the **Program** repository and zoom to the **Task Execution** repository.
2. Create a new line after the **Task Suffix**.
3. Select the **Handler** option for the **Level** column.
4. Place the cursor in the **Event** column and zoom to the **Event** dialog box.
5. Select the **Error** option for the **Event type** field.
   The default for the Event field is **Any**.

Defining the Propagate Property

You can propagate an event to an event handler in a higher level. When eDeveloper regards the event as not handled and the Propagate property is set to Yes, the dispatcher searches upwards through the task for another event handler:

- *Yes* - takes the task event to the next handler level.
- *No* - stops the event from going to the next handler level.

To define the Propagate property:

1. On the **Handler** level in the **Task Execution** repository, place the cursor in the **Propagate** column.
2. Select **Yes** or **No**.

When the error handler is triggered with an error that eDeveloper encounters, and the Propagate property is set to Yes, eDeveloper will execute other error handlers that are defined in low-level task tree.

When an **Any** error handler is defined in task level 1 (sub task), it is triggered and the Propagate property is set to Yes. If another error handler is defined in task level 0 (parent task), it will also be triggered and will perform its operations (Call Task, Verify, etc.).

Dynamically Logging into a Database

When working with databases, you may want to change the user login and login to the application as another user. The login to a database is static within an open connection. To change the user login you must close the connection, enter the new user name and password and reestablish the connection. The following steps explain how to achieve this in eDeveloper:

1. From the **Settings** menu select **Logical Names** to open the **Logical Names** repository.
2. Press **F4** to create a new logical name called **DB_USER**. Press **F4** again and create another logical name called **DB_PASSWORD**.
3. From the **Settings** menu choose **Databases** (you will need to exit your application) to open the **Database Properties** dialog box and click the **Login** tab. In the **User Name** field specify **%DB_USER%** and in the **User Password** field specify **%DB_PASSWORD%**.

You can change the values of the logical names as required using the **INIPut** function.

To close the database connection use the following function: **DbDiscnt(database name)**.

The next time a task needs to connect to the database, eDeveloper creates this connection using the new user and password values.

Handling a Violation of the Database Constraint by a User

In order to maintain database integrity, database constraints such as unique constraints and foreign keys are used. When there is a violation of these constraints the database throws an error and this error can trigger an **Error** event in eDeveloper.

To enable the user to resolve the problem causing the error you need to use the **Error** trigger. The following steps explain how to do this:

Note: This example is relevant to an SQL database.

1. Create an **Online** task for one of the application tables.
2. Define a new **Handler** in the task. From the **Event** column zoom to the **Event** dialog box and from the **Event type** list select **Error**.
3. In the **Handler's Operation** repository press **F4** to create a new operation. From the **Operation** list select **Call Task**. Press **F4** again to create another operation. From the **Operation** list select **Raise Event**.
4. From the **0** property zoom to open the **Subtask list**. Press **F4** to create a new subtask. Close this subtask to use the subtask in the **Call Operation**. The task should be in **Screen mode**.
5. From the **Task** menu choose **Range/Locate** to open the **Range/ Locate** dialog box. In the **Position** field enter the **ERRPOSITION ()** expression that returns the position of the record where the error occurred. From the **Usage** list select **Range On**.

When an error occurs, the task is called and the user is able to fix the problematic record. When the user exits the subtask the correction is saved. In order to view the updated record the user must refresh the records on the screen. You can enable this by using the **View Refresh** event.

1. Return to the **Raise Event** operation you created in the parent task.
2. From the **Name** column zoom to open the **Event** dialog box.
3. From the **Event type** list select **Internal** and from the **Event** list select **View Refresh**. Click **OK**.
4. Set the **Wait** property to **No**.

Sending Nulls by Expression

SQL databases can hold the Null value in a column. This procedure explains how to correctly set the Null expression:

1. Open the **Application Properties** dialog box and click the **StartUp** tab.
2. In the **Null arithmetic** property there are two values that you can set: **Nullify** and **Use Default**.

If you set the property to Use Default and use the NULL() function, the function result will be the field's default values and not Null.
For example: If you range from Null to Null on a numeric field, the Null is replaced by zero and you receive all the records having zero in this field.

If you set the property to Nullify, Null is used and you receive the expected result.

Note: This behavior is only effective for expressions. When using Nulls in the DB SQL WHERE clause a real Null is always used.

Working with MS-SQL Temporary Tables

Temporary tables are tables that are automatically dropped by MS-SQL when it has finished working with them.

There are two types of temporary tables:

- Local temporary tables
- Global temporary tables

Local Temporary Tables

A local temporary table is only visible in the current session, and is automatically dropped at the end of the current session. The temporary table's name is prefixed with a single number, for example: **#table_name**.

Global Temporary Tables

A global temporary table is visible in 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 task 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 actively referenced the table when the creating session ended.

Global temporary table names are prefixed with a double number sign, for example: **##table_name**.

Note: MS-SQL automatically creates global and local temporary tables in **TempDB**, irrespective of which database issued the **CREATE** command. Temporary tables are automatically dropped when they go out of scope, unless they have already been explicitly dropped using **DROP TABLE**.

To store temporary tables in the TempDB database using eDeveloper's MS-SQL7 gateway:

- In the **Table** repository add the number sign **'#'** or **'##'** as the table name's prefix.

When working with temporary tables you should also be aware of the following:

- Local temporary tables cannot be created in a subtask. To create a local temporary table as you work, you must define the temporary table in the parent task's DB table.
  You could also choose to work with global temporary tables.
- Local temporary tables cannot be viewed using Direct SQL.
  You should use global temporary tables instead.
- Local temporary tables cannot be created when the MCF file is also stored in MS-SQL. To be able to create temporary tables in such a situation, define two separate DB entries in eDeveloper, one for the MCF file and another for the data.
  You could also choose to work with global temporary tables.

Committing a Transaction Every nn Records

In a batch program that inserts or modifies a large amount of records, it is recommended to commit from time to time the transaction instead of using one large transaction.

To control the transaction duration:

1. Add a virtual variable in the **Record Main**.
2. Open a group for the variable you have created.
3. Update the variable in the **Record Suffix** with variable+1 when **COUNTER(0) MOD n=0** (where n is the number of records you wish to have in each transaction).
4. In the Enhanced tab of the **Task Properties** dialog box set the **Transaction begin** property to **group** and select the variable you defined in the **By var** field.
5. Run your program. After **n** records, the transaction is committed and a new transaction opens.

Securing Your Application 8

This chapter explores questions arising from eDeveloper's user authorization system. Rights assignment is used to control or restrict access to different elements of toolkit or runtime, program execution, menu selection, and other program elements. The eDeveloper Authorization System also provides a means to utilize underlying database security features for restricted data table access and for data encryption. In addition to assigning rights to individual users, eDeveloper simplifies 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.

This chapter covers the topics listed below:

- Creating a User in the eDeveloper Environment
- Changing the Logged-On User in eDeveloper Runtime Mode
- Assigning Rights to a User Group
- Using the Authorization System in eDeveloper Toolkit Mode

Creating a User in the eDeveloper Environment

eDeveloper allows us to define application users. The User ID is used to check User rights within eDeveloper applications. Whenever you start eDeveloper, the Logon dialog box appears. The User ID and Password must be entered. These values are checked against the User ID repository in order 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 box stays open. If the User ID and Password are found in the User ID repository, they are used to compile the rights of the user for later use, within an application. User IDs and Passwords have development and runtime functionality. The Logon also provides access to the eDeveloper Date parameter. The SUPERVISOR User ID without a password is the default user provided by eDeveloper. This ID, at least initially, has full rights to all applications.

This section includes the topics listed below:

- Logging on as Supervisor
- Creating a New Group
- Creating and Defining New Users and User IDs
- Retrieving User Information

Logging on as Supervisor

To log onto the system as SUPERVISOR:

1. On the **Settings** menu click **Logon** to access the **Logon** dialog box.
2. Enter **SUPERVISOR** in the **User ID** field, and click **OK**.

Creating a New Group

To create a new Group and assign rights to the group:

1. Log on as **SUPERVISOR**.
2. On the **Setting**s menu, select the **User Groups** option to access the **User Groups**

repository.

3. Create a new line to define a user group.
4. Zoom from the **Rights** column to the ***Rights Of:*** [the group's name] repository, and create a new line.
5. Click **Application** to select the application for which you want to define group rights.
6. Zoom from the **Key** column to the **Public Rights List** and select the right's key.
7. Steps 5 and 6 can be repeated for each application.

Note: Rights must be defined in the Rights repository within an application before they are available in the Public Rights List.

Creating and Defining New Users and User IDs

To create a new User ID and assign rights & groups to the User:

1. Log on as **SUPERVISOR**.
2. On the **Settings** menu, select the **User IDs** option to access the **User ID** repository, and create a new line to define a new user.
3. Enter the **User ID** and **Name**.
4. Zoom from the **Password** column, to the **Password** window. Enter a Password up to 8 digits and press **ENTER**. Retype the password for confirmation, and press **ENTER**.
5. Zoom from the **Rights** column to the ***Rights Of:***[the user's name] repository, and create a new line.
6. Click **Application** to select the application for which you want to define user rights.
7. Zoom from the **Key** column to the **Public Rights List** and select the right's key. **Steps 6** and **7** can be repeated for each User declared in eDeveloper.

8. Zoom from the **Groups** column of the **User ID** repository to the ***Groups of****:* [the user's name] list and select the group.

On the Main menu Settings Environment System tab, if Input Password and Input Date parameters are set to **Yes**, eDeveloper by default asks for the User ID and password.

Retrieving User Information

The following functions are used to retrieve information on the logged-in users.

RightAdd ()	Assigns a Right to a user in the Security File from within an application. Note: Only the user SUPERVISOR can use the function successfully. **Example:** RightAdd (Accountant, Issue Invoice) assigns the Right *Issue Invoice* to the user called *Accountant*.
Rights ()	Query whether user has been given rights. Returns TRUE if the user has rights.
User ()	Query user data. Returns user data as specified in the User ID repository. **Examples:** User (1) returns the User Name of the current user.
UserAdd ()	Add a user record into the Security file from within an application. **Note:** Only the user SUPERVISOR can use the function successfully.
Logon ()	The Logon function allows the user to access the current application. Example: Logon (CARL, PASS321) performs similarly to the User Logon. Unlike the User logon, the Logon function processes the user name and password in a batch mode.
GroupAdd ()	Assigns a user to a user group in the Security File from within an application. **Note:** Only the user SUPERVISOR can use the function successfully.

Changing the Logged-On User in eDeveloper Runtime Mode

eDeveloper allows you to define users who can access the eDeveloper application. eDeveloper comes with a default user, SUPERVISOR. Logging on with SUPERVISOR allows you to create users, and to limit their access to the application.

When working with MVCS, it is mandatory to log on with a user. eDeveloper provides a way to change the logon user during runtime. This option allows you to change the user for a specific use. This saves exiting and re-entering the application. During runtime, a current eDeveloper user can be displayed on the screen, and checked within a program.

Note: To identify a user from within a program use the User function.

This section includes the topics listed below:

- Logging onto the System
- Using the Logon Function

Logging onto the System

To log onto the system as user [XX]:

1. On the **Settings** menu, select **Logon** to open the **Logon** window.
2. Enter a **Username**. A Username is mandatory whereas a password and date are optional. Users are created with the eDeveloper default user: Supervisor.

Using the Logon Function

To use the Logon function:

1. Create an eDeveloper program.
2. Go to the **Task Prefix** and zoom to operations.
3. Select the **Evaluate** operation. Zoom to the Logon function.
   Example: Logon (CARL, PASS321). CARL is a username with PASS321 as password.
4. Execute the program. You will see the requested username active. This user

remains active until either eDeveloper is closed, or until this function is run for another user.
The current logged on User Name is displayed at the message line on the bottom of the screen.

Assigning Rights to a User Group

eDeveloper applications recognize rights. If the right required to perform a specific activity is assigned to a particular user, that activity is permitted to that user.

Rights can be assigned to individual users and to groups of users.

The Rights repository lists the names and keys of all of the rights defined for the application. The Supervisor has exclusive authority to modify the Rights repository, while other users can only view it. eDeveloper displays only those rights held by the user who is viewing the repository.

Note: Define the Right Key for every program in order to prohibit unauthorized use.

This section includes the topics listed below:

- Creating a Right
- Assigning Rights to a User Group
- Assigning Rights to a User

Creating a Right

To create a right (Key):

1. Log on as **SUPERVISOR**.
2. Open the required application.
3. On the **Workspace** menu, select **Rights** to open the **Right** repository.
4. Create a new line.

5. Enter a descriptive name for the Right in the **Name** column.
6. Assign a key to the Right. The key is the code name used to identify a right.
7. Select **Yes** to specify whether the right is Public, or visible outside the application, or **No**, if the right is Not Public or concealed outside the application.
8. Click **OK** to save your changes, and close the application.

Assigning Rights to a User Group

To assign rights (privileges) to a User Group:

1. Log on as **SUPERVISOR**.
2. On the **Settings** menu, select the **User Groups** option.
3. Create a new line and enter the group name.
4. Zoom from **Rights** column, and create a new line to add a Right Key.
5. Select the required right from **Public Rights List**.

Note: Different operations in the task or program must be conditioned by the Rights(N) function. Different places in the Toolkit must also be authorized for the user.

Assigning Rights to a User

To assign rights (privileges) to a User:

1. Log on as **SUPERVISOR**.
2. On the **Settings** menu, select the **User IDs** option.
3. Select a user to authorize the rights.
4. Zoom from the **Rights** column and select from the list of available Rights keys.
5. Zoom from the **Group** column to open the ***Groups of*:** list. Zoom again to the **Group List** and select a group.
6. Click **OK** to save the entry.

Note: Different operations in the task or program must be conditioned by the Rights(N) function. Different places in the Toolkit must also be authorized for the user.

Using the Authorization System in eDeveloper Toolkit Mode

eDeveloper has a flexible authorization system to control user operations. The authorization system lets the application main developer or system supervisor limit access to various activities with eDeveloper objects. The authorization system exercises its control through sets of rights and the use of built-in eDeveloper functions. Rights can be thought of as keys to locks. Using groups allows for certain classes of users to access certain parts of an application. The person in the role of Supervisor can assign rights that give each user access only to the activities for which that user is authorized.\

This section includes the topics listed below:

- Assigning Rights
- Assigning Global Rights to eDeveloper Repositories

Assigning Rights

Common rights are as follows:

- **Query right**: Allow holders of the selected right to view entries.
- **Modify right**: Allow holders of the selected right to update entries.
- **Delete right**: Allow holders of the selected right to delete entries.
- **Create right**: Allow holders of the selected right to create entries.
- **Execute/APG right**: Allow holders of the selected right to use the Automatic Program Generator and execute a program from the Program repository.

Note: The Execute/APG Right is not available for the Model, Help screens, and Menu repositories. Delete, Create, and Execute/APG rights are not available for Application properties.

To assign Rights:

1. Log on as **SUPERVISOR**.
2. Park the cursor on any entry of any repository other than Right.
3. On the **Options** menu, select the **Authorize** option.
   The Rights Assignment dialog box opens. You can assign rights for Query, Modify, Delete, Create, and Execute/APG.
   For each activity appearing in a Right Assignment field, you can enter the number of Rights or zoom to the Allowed Rights List to select the Right.

   Any user or developer who has a particular right displayed in one of the Rights Assignment dialog boxes can view the relevant dialog line, change the entry to a different right that they hold, and remove the right.

Assigning Global Rights to eDeveloper Repositories

The Rights Assignment dialog boxes are used to identify which activities within various application objects are to be permitted. These activities may be relevant to both Development and Deployment modes. When the developer attempts to perform an activity, for example to create a new row in the Table repository, or to invoke a program, eDeveloper executes the attempted activity only if the developer owns the right that corresponds to that activity.

Selecting the Authorize option on the Options menu from anywhere in the Model, Table, Program, Help, or Menu repositories, or from the Application Properties dialog accesses the Rights Assignment dialog box.

To assign Global rights to different eDeveloper repositories:

1. On the **Settings** menu, select the **Logon** option.
2. Enter a **User ID**, and click **OK**.

3. Park the cursor on the zero level of any repository, or invoke the application properties, for example: **SHIFT+F6** for the Menu repository.

4. On the **Options** menu, select the **Authorize** option, or press F9.
   The Rights Assignment dialog box opens. You can assign rights for Query, Modify, Delete, Create, and Execute/APG.
   For each activity appearing in a Right Assignment dialog box, you can enter the number of the Right, or zoom to the Rights repository to select the number.
   Any user or developer who has a particular right displayed in one of the Rights Assignment dialog boxes can do the following:

   - View the relevant dialog line
   - Change the entry to a different right that they hold
   - Remove the right

Using Advanced Programming Techniques

9

This chapter discusses the features you can add to provide the end-user with greater usability when working with your application. You can add elements such as a currency converter for the Euro, or design unique interface components.

This chapter covers the topics listed below:

- Creating a Dynamic List or Combo Box
- Merging Database Information into an Existing Document
- Developing a One-to-Many Relationship Program
- Saving and Re-Using a Form Display
- Defining Automatic Currency Conversion
- Modifying the EURO Text File
- Currency Conversion
- Fetching the Full Translation of an eDeveloper Logical Name
- Designing and Displaying an Image Push Button
- Changing the Image and Title Bar Text of the eDeveloper Window
- Making an eDeveloper Online Task Window Modal
- Exiting a Program from a Subtask Level
- Playing WAV Files from eDeveloper
- Calculating the Sum of Several Records in a Table

- Changing the Caption of the eDeveloper Title
- Changing the Windows Cursor
- Sending E-mail from Within eDeveloper
- Avoiding the Control Verification

Creating a Dynamic List or Combo Box

To maintain data integrity you usually save values from linked tables as codes rather than the description, and in the programs use the link command to retrieve the description. Elements such as combo boxes are used with programs to eliminate the need for the user to deal with code.

eDeveloper V9 has new features that ease the process of creating these controls.

This section includes the topics listed below:

- Combo Box Properties
- Using the Items List Property

Combo Box Properties

To access the Combo Box Property Sheet

1. Open the **Program** repository and zoom from a program to the **Task Execution** repository.
2. Press **CTRL+F** to open the **Form** repository and zoom from a form to open the **Form Editor.**
3. On the **Control palette**, select the **Combo Box** control and drag and drop it onto the Form layout.
4. Press **ALT+F2** to open the **Combo Box** property sheet.

The Details section of the Combo Box property sheet contains the following new properties, as well as the old ones, for displaying and retrieving values from a table:

Source table	The number of the linked table in the Table repository. Click the ellipsis button or zoom to the Table list.
Display field	The number of the variable you want to display (the description). Click the ellipsis button or zoom to the Variable list.
Linked field	The number of the field you want to retrieve (the code). Click the ellipsis button or zoom to the Variable list.
Index	The number of the index you want to use for the sort. Click the ellipsis button or zoom to the Index list.
Field ranges	The programmer can set a range value(s) for the linked table. Click the ellipsis button or zoom to the Field Range screen.

The eDeveloper Engine opens the requested eDeveloper query table and retrieves the displayed and linked field values from the table, in accordance with the defined range.

Using the Items List Property

The existing Items List property value is used when the eDeveloper table cannot be opened or is not defined. The Control parent and child relationship works in the same way as an expression in the Items List property.

Merging Database Information into an Existing Document

When working in an eDeveloper Web environment, eDeveloper creates HTML files. It can do this either with its own HTML editor and tools or by merging with an external template. This template is a text file with special tags that eDeveloper recognizes. eDeveloper recognizes these tags using a special prefix and suffix for each tag. There are two types of tags – name tags (placing values) and logic tags.

The template can be either an HTML file, *.txt file or MS-Word RTF file.

Using external templates and eDeveloper's merge functionality offers great flexibility.

This section includes the topics listed below:

- Using HTML Tags
- Selecting the HTML Merge Option
- Defining the Tag Values
- Defining the HTML Merge Task Controls

Using HTML Tags

The HTML Template File Tags repository is used to define data elements that are matched with and replaced by the appropriate merge tags in the associated HTML template.

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.

All trailing blanks of data strings sent as output to the merge mechanism are trimmed.

The template file may be of any type as long as the eDeveloper merge tags are stored as regular ASCII strings.

To work with HTML tags, you should carry out these steps described in the sections below:

1. Select the **HTML Merge** option.
2. Define **Tag** values.
3. Define the **HTML Merge Task** controls.

Selecting the HTML Merge Option

To select the HTML Merge option:

1. Create a program and zoom to the task screen. The **Task Properties** dialog box opens.

2. Select **Batch** as the **Task type** and click **OK**.
3. Select the **I/O Files** option on the **Task** menu.
4. Open a new line and enter a name for the file.
5. In the **Media** column, select **File**.
6. Press **CTRL+F**, create a new line.
7. In the **Interface Type** column, select **HTML Merge**.

When you have completed these steps you should then define the Tag values.

Defining the Tag Values

To define the Tag values:

1. Press **CTRL+P** from the **HTML Merge** line to open the **Form Properties** sheet.
2. Enter the RTF filename in the **HTML File** property.
3. Zoom from the **Tags Table** property to open the Tags Table.
4. Click **Select New Tags** and select the required tag.
5. Specify the tag's value (either by a variable or by an expression).

When you have completed these steps you should then define the HTML Merge Task Controls.

Defining the HTML Merge Task Controls

To define the HTML Merge Task Controls:

1. The **Picture** value is set by default. You can also use an expression.
2. In the **Task** repository, zoom to the **Record Suffix** Operation repository and create a new line.
3. In the **Operation** column, enter the letter o to select the **Output Form** operation.

4. From the **I/O** field, zoom to the **I/O File List** and select the file.
5. Press **CTRL+C** to open the **Task Control** dialog box.
6. On the **Modes** tab, select **No** for the **Allow options** property, then click the **Behavior** tab and select **No** for the **Open task window** property. Click **OK**.

Note: If the tag is an MGIF tag, the specified value should be logical (True or False). It is not necessary to specify MGREPEAT, MGENDREPEAT, MGELSE and MGENDIF tags.

Developing a One-to-Many Relationship Program

The two most important and useful relationships among tables in a relational database system are:

- One-to-One: implemented in eDeveloper by the Link operation, which establishes a relationship between the current row of the Main table and a specific row in the Linked table, inside a task.
- One-to-Many: implemented in eDeveloper by a parent task and child subtask structure. A one-to-many relationship exists between two tables, whenever each row in one table corresponds to, or owns, many rows in another table. eDeveloper uses corresponding or matching columns to associate rows of the two tables. The matching process is more efficient if you define the matching columns as indexes in the tables.

The following example of a customer order and billing application illustrates such a relationship:

- A Customer table row represents a single customer. Each Invoice table row represents an invoice associated with a particular customer.
- Each Customer table row is related to many Invoice table rows, through the Customer No. column, defined as an Index in both tables.

This section includes the topics listed below:

- Building a One-to-Many Program
- Preserving Data Integrity in the One-to-Many Relationship

Building a One-to-Many Program

The procedures below describe how to build a one-to-many program for an Order Entry screen:

You should first create a Parent Task as follows:

1. Create a parent task with an **Order Headers** table as the **Main Table**.
2. Select all necessary fields.
3. Generate a form in **Screen** mode.

Once you create the Parent task you should then create the Subtask as follows:

1. Create a subtask with an **Order Lines** table as the **Main Table** and select all necessary fields.
2. Specify the range values of an Order_Number field as the Order_Number in the parent task.
3. Specify the **Init** value of the Order_Number field using the same expression as the range value.
4. Open the **Task Control** dialog box (CTRL+C) and on the **Behavior** tab, set the **Close Task Window** property to No.
5. Open the **Task Properties** dialog box (CTRL+P) and click the **Properties** tab. From the **End Task Condition** zoom to define the property as **"level(1)='RP'"** Set **Evaluate Condition** to **Immediately when condition is changed**.
6. Generate a form in **Line** mode.
7. Once the subtask has been created, you need to call it from the parent task in two places:
   - Record Prefix
   - Record Main

Note: The subtask can also be designed as a separate program.

Make the following modifications:

1. Select a parameter for the parent task Order_Number.
2. Modify the range and initial value of the Order_Number field to the parameter value.

Preserving Data Integrity in the One-to-Many Relationship

This section includes the topics listed below:

- Deleting a Parent Record (order title)
- Aborting the Order Entry Program

Deleting a Parent Record (order title)

To prevent a user deleting a parent record, leaving non-valid records in the Order Lines table, do the following:

1. Add a batch subtask to the parent task with Order Lines as the main table.
2. Set **Delete** as the **Initial mode**.
3. Select the Order_Number field only and specify the range values as the parent task's Order_Number.
4. Open the **Task Control** dialog box (CTRL+C) dialog and click the **Properties** tab. Set both the **Allow options** and **Open window** properties to **No**.
5. Call the subtask from the **Record Suffix** level using **STAT(0,'D'MODE)** as the condition of the **Call** operation.

Aborting the Order Entry Program

To prevent a user aborting or adding the parent record modification, leaving non-valid records in the Order Lines table:

1. Add an **Update** operation in both the Record Suffix level and in the parent task of one of the fields.
2. Set **No** in the **Undo** property of the operation.

Saving and Re-Using a Form Display

You can preserve a unified form design using the eDeveloper Form templates.

This section includes the topics listed below:

- Form Templates
- Saving a Form Template
- Loading a Form Template

Form Templates

You can save a current form layout 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 have empty value parameters when loaded from the template file.

If you load a template into a pre-existing form, the template file overwrites all form settings. In addition, all controls that were on the form will be deleted. Form Templates are limited to GUI forms. HTML and Java templates are not accepted by the GUI Form Editor.

Saving a Form Template

After designing the form template:

1. On the **File** menu, select the **Save as Template** option.
2. Save the form as a template in the desired directory.

Loading a Form Template

To use an existing form template:

1. Enter the form.
2. On the **File** menu, select **the Load Template** option.
3. After loading the template, connect the object with the correct fields.

Defining Automatic Currency Conversion

eDeveloper supports European Currency Conversion methods using the Currency Conversion file and the special functions for European currency manipulation.

This section includes the topics listed below:

- Creating a Euro Conversion Text File
- Setting eDeveloper to Use the Currency Conversion File

Creating a Euro Conversion Text File

You can use the Currency.xxx file, in eDeveloper's support folder, and modify it accordingly. The xxx extension represents the local language.

The file indicates the different currency types and rates. Each line in the file defines a currency using the following syntax:

<CODE><NAME><PRECISION><RATE>

CODE	A four character string eDeveloper uses to identify the currency.
NAME	20 character string indicating the name (free text).
PRECISION	A numeric value defining the precision result of calculations made with this currency. The valid values are 0,1,2.
RATE	The currency rate compared to 1 Euro.

Setting eDeveloper to Use the Currency Conversion File

You can define the path eDeveloper uses to locate the Currency Conversion file you have created.

To locate the file:

1. On the **Settings** menu, select the **Environment** option.
2. Click the **External** tab and find the **European Currency Conversion file** setting.
3. Zoom to the **Open File** dialog box and select the required file.

Modifying the EURO Text File

During the workflow of an application, you might need to modify the currency conversion file to accommodate fluctuating currency rates.

To modify currency exchange rates and to make eDeveloper aware of the changes, you can use the following methods:

- Using the OS Text Editor.
- Using functions to modify currency rates used by eDeveloper.

Using the OS Text Editor

eDeveloper uses an external text file for reading the currency conversion table. You can use an OS text editor such as Notepad.exe to modify the rates, or add or delete currencies.

The modification takes effect when eDeveloper is restarted.

Modifying the EURO Text File Using eDeveloper Functions

Modification to the currency file using an OS text editor only takes effect when eDeveloper is restarted. Using the EuroDel() & EuroUpd() functions you can modify the currency table, currently held in the memory, without restarting eDeveloper. However, when using these functions, only the eDeveloper memory is changed and not the Currency File itself. Therefore, changes will be lost when eDeveloper is restarted.

- EuroDel() is used to remove a currency from the conversion table held in memory.
- EuroUpd() is used to modify a currency entry.

  The syntax is:

  EuroUpd(‘Currency Code’, ’Currency Name’,’Precision’,’Rate’)

  If there is no currency code, a new currency entry will be created.

Currency Conversion

When working with currencies you may need to convert from one currency to the other.

This is more important when working with the Euro currency since in many cases you also need to show its value in other European currencies.

You should make sure that you have already defined the European Currency Conversion File.

You can use the following two methods for currency conversion:

- Converting Currencies Using Different Display Types
- Converting Currencies Using the Euro Functions

Converting Currencies Using Different Display Types

eDeveloper can automatically convert currencies, using the currency file's data, whenever it is required to present a different currency on the Output or GUI form.

1. Open a **GUI** or **Output** form and on the **Form** layout drag and drop an **Edit** field.
2. Select the **Edit** field and press **ALT+F2** to open the control's property sheet.
3. Zoom from the **Viewed Currency** property and set as any currency code that is currently available in the Currency Conversion File.

The default is **ASIS**, meaning that the value of the field will not be converted while it is displayed. If you enter the value, it will be of the current default Euro Code.

Once you enter a different Conversion Currency, whenever a value that does not match the currency code is presented in that field, it will automatically be converted to that currency using the Currency Conversion file.

For example: If a value of the EURO is passed and displayed in a field on the form marked as "Viewed Currency = DEM", eDeveloper will automatically convert the numeric value from the Euro currency to German Marks.

Converting Currencies Using the Euro Functions

During calculations you might need to switch the numeric variables of some currencies. You can do this by using the EuroCnv() function. The function converts a numeric value from one currency to another, using the currency conversion file/table stored in eDeveloper's memory.

For example, to convert variable A from the EURO currency to DEM, you can use the expression: EuroCnv('EURO','DEM',A)

Note: Use the display method only when you need to show the user the value in different currencies. When internal calculations are required, you should use EuroCnv() instead.

Use the EuroSet() and EuroGet() functions to set and retrieve the default currency of the current eDeveloper instance.

Fetching the Full Translation of an eDeveloper Logical Name

Sometimes you create a Logical Name based on other logical names.

For example:

Path = `c:\myApp\`

Images = `%Path%images`

When using INIGet('[MAGIC_LOGICAL_NAMES]Images') you will get the value %Path%Images and not c:\myApp\Images

The program needs to parse the logical name and look for the '%'signs.

To fetch the translation:

1. Isolate each one of them (starting from the last to the first).
2. Concatenate them all back together.

Designing and Displaying an Image Push Button

There are 3 types of buttons you can use in eDeveloper:

- Regular
- Aperient: Resembles a Web hyperlink (underlined blue text).
- Image: This is for a more displayable window.

This section includes the topics listed below:

- Designing the Image Button
- Displaying the Image Button in an eDeveloper Program

Designing the Image Button

Image buttons have four potential statuses, each requiring a variation of the image.

The four statuses are:

On Park	The way a button appears when the cursor parks on it.
On Click	The way a button should appear when clicked.
Disabled	The way a button should appear when disabled.
Default	The default view of the button, when the cursor is not parked on it, and it is not in use.

If the button has a single status image, eDeveloper breaks the image into the four statuses described above, giving the impression that it is not working properly.

Displaying the Image Button in an eDeveloper Program

To display the button in an eDeveloper program:

1. Select an **Alpha** virtual variable. The size must be big enough for the full path of the image.

2. In the **Attributes** field, specify that this is a **GUI object** button and set the type of button as an **Image Button**.
3. In the **Init** column set the value as the location of the image. A Logical name can be used.
4. Place the object on the screen.

Changing the Image and Title Bar Text of the eDeveloper Window

eDeveloper has its own Icon. You can use the "RtUserCopyright" Magic.ini switch to set the Title Bar label in Runtime. This option is only available in Runtime and cannot be changed while eDeveloper is already running.

Using dlls, you can change the displayed icon and title of eDeveloper.

The eDeveloper window icon however, cannot be changed.

This section includes the topics listed below:

- Changing the Main Window Title
- Changing the eDeveloper Icon

Changing the Main Window Title

Set the following in the Record Main level:

1. A **Select Virtual** operation, Numeric 8, to get the parent (the eDeveloper Window) of the program's window.
   The **Init** value should be
   CallDLLS ('USER32.GetParent','44',WINHWND (0)).
   This virtual field's **alias** is A.
2. A **Select Virtual** operation, Numeric 8, to get the window handle of the eDeveloper Window.
   The **Init** value should be

CallDLLS ('USER32.GetParent','44',A).
This virtual field's **alias** is B.

3. A **Select Virtual** operation, Alpha 30, for the new caption to be set.
   This virtual field's **alias** is C.

4. A **Select Virtual** operation, Alpha 10, for the push button executing the change.
   This button should raise an Internal Zoom event.

5. An **Evaluate** operation with the **Flow** column set to 'After'.
   The expression should be
   CallDLLS ('USER32.SetWindowTextA','4A4',B,C).

6. On the form place two fields - the caption and the button.

This example is for an Online task. This program could also be a batch program, in which case you do not need the button, and the Evaluate operation should be in the Task Suffix.

Changing the eDeveloper Icon

Set the following at the Record Main level:

1. A **Select Virtual** operation, Numeric 8, for getting the parent (the eDeveloper Window) of the
   program's window.
   The **Init** value should be CallDLLS ('USER32.GetParent','44',WINHWND (0)).
   This virtual field's **alia**s is A.

2. A **Select Virtual** operation, Numeric 8, for getting the window handle of the eDeveloper Window.
   The **Init** value should be CallDLLS ('USER32.GetParent','44',A).
   This virtual field's **alias** is B.

3. A **Select Virtual** operation, Alpha 30, for the new icon filename to be set.
   This virtual field's **alias** is C.

4. A **Select Virtual** operation, Numeric 10, for the Icon handle.
   This virtual field's **alias** is D.

5. A **Select Virtual** operation, Alpha 10, for the push button executing the change. This button should raise an Internal Zoom event.

6. A **Block-If** operation with flow 'After'.
   The block should contain the following operations:
   a. **Update** D with CallDLLS ('shell32.ExtractIconA','4A44',0,C,0).
   b. **Evaluate** with expression
   CallDLLS ('USER32.SendMessageA','44444',B,128,1,D)
   c. **Evaluate** with expression
   CallDLLS ('USER32.SendMessageA','44444',B,128,0,D)

7. On the form place two fields – the Icon filename and the button.

This example is for an Online task. This program could also be a batch program, in which you do not need the button and the all the operations in the Block should be in task suffix.

Making an eDeveloper Online Task Window Modal

A Modal Window means that you cannot click another window without first closing the current (modal) window.

To make the task window Modal:

1. Press **CTRL+F** to Enter the task's **Form** repository.

2. Press **CTRL+P** to open the Form's Property sheet.

3. Set the **Modal Window** property to **Yes**, or an expression that evaluates to **True**.

Exiting a Program from a Subtask Level

The Exit operation exists only on the current level. If you are on a task on a lower level, the Exit operation only takes you up one level. The procedures below, discuss how to exit a program from the lowest subtask level using a button on the task flow.

Two procedures are given, one uses the Fill function familiar to Magic V8, and the other is unique to eDeveloper V9. Both procedures assume the user will click a button in order to exit.

Through Magic V8, an event consists of two components, the trigger and what to do with the trigger. In eDeveloper V9, these components are separated. The trigger is "raised" with either the new Raise Event command, or with a push button. The second comment, or the action, is called a Handler. You can place as many commands as you want inside the handler. One property of the handler is to propagate. This means that the handler will "push" the trigger to upper levels of the task.

This section includes the topics listed below:

- Using the KbPut Function with the Fill Function
- Using Events in eDeveloper

Using the KbPut Function with the Fill Function

1. In the **Push Button** handler, or action, add an **Evaluate** operation.
2. Enter the expression
   KbPut (Fill ('Exit'ACT,TDepth ()))

Using Events in eDeveloper

1. Add a **Raise Event** operation in the lowest task level in the **Push Button** handler.
2. The event should be an **Internal Exit**, and the **Wait** property should be set to **No**.
3. The **Push Button** trigger should be propagated.
4. Define the **Push Button** handler with the same operation in all tasks except the parent task.

Playing WAV Files from eDeveloper

The following steps describe how you can use WAV files in your application using the **winmm.PlaySoundA** function:

1. To call this method use the **CallDLL** function.
2. Evaluate the following expression:
```
CallDLL
('winmm.PlaySoundA','A48','c:\path\MsgSent.wav',13107
2)
```

c:\path\MsgSent.wav is the complete path of the WAV file. The **PlaySoundA** function receives the following three parameters:

- A **string** that specifies the WAV sound to play.
- The executable file's **Handler**, containing the resource to be loaded. This must be **Null** if requesting a file on disk.
- A **flag** specifying that the first parameter is a filename.
  The number 131072 is Hex 0x20000, which is the flag value for **snd_filename**.

Calculating the Sum of Several Records in a Table

Sometimes when you present a table on the screen you want to give the user the ability to simultaneously perform certain actions on several of the records, such as calculating the sum value of several records. With the table multi-marking capabilities and with a set of eDeveloper functions you can add this functionality.

Assume that you have a table with two columns, product number and product price. If you want to calculate the sum of the price of several products you can multi-mark them and activate a handler to calculate the sum; for example, by pressing the **F9** key.

To calculate the sum of several records in your table:

1. Create a handler for the **F9** key, a system handler.
2. In the handler create a virtual numeric field. This field is used to accumulate the values.

3. Create an **Update** operation for the virtual; the updated value should be zero. The update condition should be **MMCurr (0)=1**. This function makes sure that this update is performed only for the first record you marked.
4. Create an **Update** operation for the virtual. The updated value should be the virtual + the column you are adding.
5. Create a **Verify** operation and display **The Sum is '&STR (A,'##')** where A is the accumulated variable. The condition for this operation should be **MMCurr (0)=MMCount (0)**. This condition causes the verify to appear only for the last record.

Changing the Caption of the eDeveloper Title

The default caption of the eDeveloper window is "eDeveloper". You can specify your own caption for the eDeveloper window. This topic explains how to work with the windows user32.dll to perform the required task.

To change the eDeveloper caption title:

1. Create a batch program with the following three virtual variables:

   A. **Programs Parent** (Numeric 8) - to get the parent (the eDeveloper Window) of the program window.
   B. **MAGIC Window Handle** (Numeric 8) - to get the window handle of the eDeveloper Window.
   C. **New Caption** (Alpha 30) - to set the new caption.
2. In the **Task Prefix**, update the **Programs Parent** variable with the following expression:

   ```
   CallDLLS ('USER32.GetParent','44',WINHWND (0))
   ```
3. In the **Task Prefix**, update the **MAGIC Window Handle** variable with the following expression:

   ```
   CallDLLS ('USER32.GetParent','44',A)
   ```

   - where A is the 'Programs Parent' variable.

4. In the **Record Suffix**, update the **New Caption** variable with the following expression:

```
CallDLLS ('USER32.SetWindowTextA','4A4',B,'My new title')
```

- where B is the ‘MAGIC Window Handle' variable. You can replace 'My new title' with any title you wish.

Changing the Windows Cursor

You can change the Windows cursor to different shapes throughout the application using the SetCrsr function.

To change the cursor shape:

1. Define an **Evaluate** operation.
2. Add the **SetCrsr**(number) function to the **Evaluate** operation.

This function expects a numeric parameter that determines the cursor shape. The expected numeric values and the cursor shape they produce are:

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

Sending E-mail from Within eDeveloper

To add the ability to send e-mail messages from your application:

1. Create a handler in your program for the **Send** action.
2. In your handler, define a numeric virtual variable (**N6**). This variable is used to call the **Mail Connect Error Code**.
3. Update the **Mail Connect Error Code** variable with the function:

```
MailConnect (1,SMTP_Server_Name,'','')

The MailConnect function has the following parameters:

 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
```

   This method of calling creates the connection. If the connection establishment fails, an error code is returned and the developer can decide what to do next, such as displayiing the error with the **MailError** function.
4. If the connection is successful, select another numeric virtual variable (**N6**) with the name **Mail Send Error Code**.
5. Update the **Mail Send Error Code** with the following function:

```
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 attach-
ments to the mail

Returns:0 if Success<0 - Error code
```

You can use the return value for validity check or check the error with the **MailError** function.

Avoiding the Control Verification

The Control Verification is 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.

If you want to add a button on your screen that will perform cancel and exit, and you have Verify operations in the Control Verification level, then whenever the button is clicked, the verification will be performed.

To avoid the control verification:

Use the CtrlName() function and check that the name is not the Cancel button name. When the cancel button is clicked, the control verification will not be entered and you will not have the problem with the Verify operations that you defined.

Printing With eDeveloper 10

This chapter discusses many of the printing issues in eDeveloper

This chapter covers the topics listed below:

- Automatically Generating a Printing Program
- Creating a Simple Printing Program
- Defining a Standard Header and Footer
- Printing to the Same I/O File from Several Subtasks
- Printing to the Same I/O from Several Programs
- Changing the Printing I/O Media
- Changing the Default Printer from Within eDeveloper
- Allowing Print Preview
- Creating More Than One Report Set in a Program
- Creating a Report in PDF Format
- Printing Different Length Multi-Line Texts
- Controlling the Number of Lines Displayed in a Table
- Printing a Table Within a Subtask

Automatically Generating a Printing Program

Using eDeveloper's Automatic Program Generator (APG) you can create programs automatically. This helps build the basic structure of a task that can be modified and customized to implement more sophisticated needs.

You can access the APG in one of the following ways:

- On the **Options** menu, click **Generate Program**.
- Press **CTRL+G**.

Generating a Printing Program

This topic explains how you can use the APG to create a printing program.

To set the APG parameters

1. From a table in the **Table** repository press **CTRL+G**.
2. Select **Generate** mode to save the printing program in the **Program** repository, select the **Execute** mode for execution only.
3. Zoom to the **Column** selection table and number the order of the columns from left to right.
   0 – zero means that the column will not be included in the fields on the report.
4. Give the program a name. This is the program name that the **APG** generates and adds to the **Program** repository.
5. To save the output report as a file and letter, use the **COPY COMMAND** and send it to the printer.

To choose the desired style

You can decide whether the report prints as a table or with each record on a separate page. If you select **Generate**, you can design the output format at a later stage.

1. In the **APG** dialog box click the **Style** tab to give your report a three-dimensional or two-dimensional appearance.
   If you have a pre-designed GUI Output Form model you can select it from the

list of GUI Output Form models and apply it to the new report.

2. Select the **Caption** check box to display the program name as the report header.

Creating a Simple Printing Program

This procedure uses the Table APG to generate, but not execute, a Print program, and view the output.

This section includes the topics listed below:

- Defining the Task and I/O Properties
- Preparing a Report's Dataview
- Designing a Report's Appearance
- Using the Output Form Operation

Defining the Task and I/O Properties

To define the Task and I/O Properties:

1. Define a **Batch task** with the **Main** file.
2. To disable the **Start Execution** dialog box press CTRL+C to open the Task Control dialog box. Click the **Modes** tab and set the **Allow Options** field to **No**.
3. Press CTRL+I to open the **I/O Files** screen. Create an output file with **Media** set to **Graphic Printer**.
4. To enable the end-user to select the printer, set **PDlg** to **Yes**. To provide the end-user with the Print Preview option, press CTRL+P to open the **I/O Properties** dialog box and set **Print Preview** to **Yes**.

Preparing a Report's Dataview

To prepare a report's dataview:

1. Select the **Main table**, and define the fields to use from that table.
2. Go to the **Record Main** section and using the **Select** operation select the fields you want to use from the **Main table**.
3. If you need temporary fields, you can define **Virtual fields** and initiate them with any expression in the **Init** column.

Note: Only use the **Select** and **Link** operations in the **Record Main** section.

Designing a Report's Appearance

To design the report's appearance:

1. Press CTRL+F to open the Form repository.
2. Create a new form with **Interface Type** as **GUI Output**, with **Class** as **1** and **Area** as **Header**.
3. From the **Name** column zoom to open the **Form editor**.
4. Place a **Table** control on the form and assign the field to the table.
5. Add **Text** controls, such as **OWNER()**, **DATE()**, above the table. This appears on every page as a header. You can also add a footer to the table in the same way.

Using the Output Form Operation

The step below connects between the reading of the records and sending them to the printer using the designed format. You must send the information for every record to the printer.

1. In the **Record Suffix** handler add an **Output** operation.
2. The **Output** form sends **Form 2** to **I/O file 1** with **Page** as **Auto**. This means that Printing headers are automatically managed by eDeveloper.

eDeveloper treats the controls around the table as headers, and only prints them for the first record and for every new page. For the second record, eDeveloper only sends the specific row to the printer and not the entire page.

Defining a Standard Header and Footer

Sometimes you may want to print a standard header and footer, such as a company logo, for every printed page.

In eDeveloper, you can define blocks of printing to be fixed headers and footers for every page.

To define a permanent header and footer in your report:

1. Press CTRL+F to open the **Form** repository and create 2 new forms.
   For the first one set **Area** = Page Header.
   For the second one set **Area** = Page Footer.
   These forms do not replace the normal headers that you need to generate for the report. They are 2 forms that eDeveloper prints before all other forms on every page.
2. Press CTRL+I to open the **I/O file** repository and then press CTRL+P to open the **I/O properties** dialog. Assign your new forms to the **Page header form** and **Page footer forms** respectively.

eDeveloper automatically raises an internal event before printing Page header/footer forms. This lets you assign handlers to these events in order to calculate computed fields that are placed on these forms.

These events are called: Page Header and Page Footer.

Printing to the Same I/O File from Several Subtasks

There are some complex reports that need to be handled by more than one task. If the information for the report needs to be taken from several tables with a connection between them, then several subtasks need to be defined in order to handle the table processing. An example is an invoice, where you need to generate two tasks that will manage the printing. One to print the invoice header and the second to handle each invoice's line detail. Both tasks need to use the same printing definition.

A task behaves in the following way: Before executing the Task Prefix, the engine opens the I/O file, which is defined in the task's I/O files, and closes it after executing all the operations in the Task Suffix.

To use only one I/O file for all tasks, you need to find a global place for its definition, so that it opens once when the processing of the report begins and closes once on completion of the printing process.

Using the Same I/O Among Subtasks

To define an I/O file that can be used among several subtasks:

1. Define the **I/O** file in the parent task.
2. Zoom to one of the subtasks to view, in the **I/O** table, the files defined in the parent task.
   The title indicates that this is the I/O files table, belonging to the subtask that prints the Order/Invoice lines. The I/O file **Print Report** was defined in the parent task and can be used in the subtasks.

After defining an I/O file in the parent task so that all subtasks can use it, it is easy to redirect the output to that I/O file. To be able to do this use this I/O file number when using the Output Form operation.

Printing to the Same I/O from Several Programs

If you need to print a complex report such as a customer folder, you must write several programs that manage the printing of items such as orders, invoices, or payments. These programs must use the same printing and I/O definitions.

Before a program executes the Task Prefix, the engine:

- opens the I/O file, defined in the task's **I/O Files** repository and
- closes it after executing all the operations in the program's task suffix.

To only use one I/O file for all programs, define a global I/O file so that it:

- opens once when the processing of the report begins and

- closes upon completion of the printing processing. All programs should direct the output forms to that I/O file.

Using the Same I/O Among Programs

To use the same I/O with different programs:

1. **Defining a Global I/O file in the Top program of the Runtime Tree**
   To define one I/O file that can be used among several programs, you must define it in the upper program. When calling other programs, you need to define another I/O file in every program that referring to the upper I/O file.

2. **Referring to the Upper I/O File Within a Called Program**
   Unlike subtasks, where all I/O files from the parent task can be seen and used in the subtasks, when calling a program within a program you need to define an I/O file in the called program that will refer to the I/O file defined in the calling program.

   This link between two I/O files can be done using the I/O name to use in the I/O file properties. The I/O file in the called program must have an expression with the name of the I/O file that was defined in the calling program.

   When using the same I/O file name, the engine does not open a new I/O file, but directs the output to the same I/O file that is already opened.

Changing the Printing I/O Media

This topic discusses the Windows print dialog and media expression in I/O file properties.

This section includes the topics listed below:

- Windows Print Dialog
- Media Expression in the I/O File Properties

Windows Print Dialog

If your application does not internally manage a list of resources for each user, the easy (and quick) way to let the end user choose the printer is to display a list of printers before the report execution.

To choose the printer:

1. Set the **PDlg** to **Yes**, or any Boolean expression, in the **I/O file** properties. Before writing to that **I/O file**, the engine opens the Windows Printer dialog to choose the printer and set its properties such as copies, orientation, or the name of a file to create for later printing.
2. Select different priorities, if necessary, then click **OK**.

Media Expression in the I/O File Properties

You can use the media expression to define any valid operating system file name. You also have the option to include a server and path to the file name or use a logical name in the expression. In these cases, eDeveloper will read or write according to the specified criteria. The expression is evaluated in the task initialization stage, and cannot change dynamically. Instead of using a file name, you can use a reserved name such as: Console, LPT1:, LPT2:, LPTn:, or any printer name as defined in the Windows printer list. The engine redirects the output according to the media expression.

Changing the Default Printer from Within eDeveloper

This topic discusses the printer setup and printer dialog. It explains how you can provide the end-user with the capability to change printer properties, such as copies, orientation, and file name.

To setup the printer:

1. From the **File** menu click **Printer Setup**.
   By selecting this option before executing the report, you can access the list of printers defined in the operating system and change their properties.

This is the same action that can be performed by selecting the **Printers** option on the **Settings** menu.

2. Make any changes, if necessary, and then click **OK**.

The Printer Dialog

In the I/O file properties the **PDlg** property is only relevant when **Media=Graphic Printer**. The **Yes** option specifies that when the task opens the I/O file, a Windows print dialog box appears allowing the end-user to change the printer and its parameters. The **No** option specifies that the Windows print dialog box should not appear, and that the print parameters are transferred directly to the printer without end-user interaction. The Expression option allows dynamic selection of the **Yes/No** setting.

Allowing Print Preview

This topic explains enabling print preview in the I/O file properties and I/O file table before Runtime report execution. It explains how to permit the end-user to view the reports on the screen before sending them to the printer.

There are 3 ways to allow the end-user to view the report before sending it to the printer:

- In the **I/O Properties** dialog box set the **Print Preview** option to **Yes**.
- In the **I/O file media** set an expression **Console**.
- In the **Task control** dialog box set the **Allow options** property to **Yes** and use **CTRL+I** before starting the process of the report. This is done when the confirmation message box appears on the screen. In the **I/O file** repository, choose one of the other options.

Creating More Than One Report Set in a Program

This topic discusses Class > 0 and the **Auto Page in Output Form** operation, and explains how to define several report layouts in the same program, and still let eDeveloper handle the internal mechanism of automatic header printing.

GUI Forms (class = 0) are used for user interaction and Output Forms (class > 0) are used for report layouts.

If you need only one report format, you only need to create 3 forms (class = 1) for header, detail, and footer:

- If you use the **Output form** operation with **Page = Auto**, eDeveloper automatically prints the headers of the same class on every new page.
- If the page header and footer are defined in the **I/O file** properties, they are printed first. After that, it prints the headers of the same class.
- It is important to define the forms in the **Forms** repository in the same order as above.

If you need to define more than one layout for the report, and still have the automatic headers and footers printing mechanism, then you need to create another class (class = 2) with the second format.

Creating a Report in PDF Format

This topic discusses the printer driver and the PDF Writer and explains how to create reports in PDF format.

This section includes the topics listed below:

- Creating the Report Outside eDeveloper
- Creating the Report Inside eDeveloper

Creating the Report Outside eDeveloper

To create a report in PDF format:

1. First install a printer driver that supports PDF format.
   You can find this driver in the Adobe site under the support section.
2. Download and install the PDF writer driver.
   You now have another printer in the OS environment that supports PDF format.

Any printing sent to that driver can be saved as a PDF file and can be viewed by Acrobat Reader.

Creating the Report Inside eDeveloper

To create a report in PDF format:

1. In the **I/O Files** repository set the **PDlg** (Print Dialog) property to **Yes**
2. Enter an output file name in the expression column. This is the file name eDeveloper uses.

   eDeveloper sends the report to the PDF driver and creates a PDF file for preview. You can print this file or send it via e-mail.

Printing Different Length Multi-Line Texts

This topic describes how to deal with a Multi-Line Edit control and how to print its contents.

The Multi-Line Edit control gives the end-user the capability to store a lot of information in a small area (place) on the GUI screen.

When designing the reports, it is impossible to guess how much space is needed to print all the information (text) in such controls.

To create flexible controls on a flexible form that displays the entire text:

1. In the **Form** properties sheet set the expand form to **One page** or **Multi-page**. This determines whether a GUI or text-based form can be expanded to accommodate Multi-Line Edit controls.
2. Specify **One page** to enable the Multi-Line controls to be resized to contain all of the text in the control's frame. The form can increase its size up to a single page to make room for new lines in the Multi-Line Edit control.
3. Specify **Multi-page** on GUI forms to enable Multi-Line Edit controls to expand onto multiple pages using eDeveloper's header and footer mechanism.

4. On the form, enter the **Edit control** properties and in the input group of properties set the Multi-Line Edit to **Yes**.

Controlling the Number of Lines Displayed in a Table

You may be need to customize a report to fit the end-user's paper type. You can do this by controlling the number of lines to be printed on every page.

To design the report:

In this example to specify that every page should contain only 10 records:

1. In the **Table properties** of the form, set the **Fix Size table** property **to Yes**.
2. Set the table size according to the actual size that should appear on the paper.
3. On the last page, if there are less than 10 records, eDeveloper fills empty lines until reaching the desired table size. This ensures that content below the table is always printed at the same position on the page.

Note: You must set the table size to reflect the actual size on the form. If you do not do this, the report will contain one record per page. By default, eDeveloper draws a table with only one record.

Printing a Table Within a Subtask

This topic describes how to control the way eDeveloper prints tables within a subtask when it is called several times from the parent task.

To print a table within a subtask:

1. In the **[MAGIC_SPECIALS] INI** section use the **ClosePrintedTablesInSubtasks** setting.
2. If the **ClosePrintedTablesInSubtasks** does not exist or is set to **Y**, eDeveloper closes the printed table every time it enters the subtask.
3. If it is set to **N**, the table opens and the print-out resembles one long table.

Defining Application Menus 11

This chapter discusses how you can define the menus in your application. You can determine menu content, accessibility, size, and format. The different menu formats discussed in this chapter include pull-down and context menus.

This chapter covers the topics listed below:

- Defining a Pull-down Menu for an Application
- Changing Options Displayed in the Runtime Toolbar
- Assigning Help Screens
- Assigning Help Prompts
- Defining an Application Default Menu
- Defining an Entry Image on the Runtime Context Menu
- Defining a Context Menu for a Specific Program
- Executing a Program from the Menu with Arguments
- Creating Additional Context Menus
- Defining a Shortcut Key for Menu and Separator Entry Types
- Defining a Shortcut Key for Program, OS Command, and Event Entry Types

Defining a Pull-down Menu for an Application

eDeveloper generates a default pull-down main menu for each new application. Each application can have only one pull-down menu defined.

To define a menu:

1. From the **Navigator** pane, select the **Menu** repository.
2. Select the **Default Pull-down** menu and zoom to the **Menu Definition** list.
3. Add or remove entries.

Changing Options Displayed in the Runtime Toolbar

The supervisor can define rights for the logged-in user. If the user does not have the right to execute an entry, the entry is disabled in the pull-down menu.

You can also change the menu in runtime by using the MNUENABL and MNUSHOW functions.

This section includes the topics listed below:

- Defining Options Using the Menu Repository
- Defining Options Using the MNUENABL and MNUSHOW functions

Defining Options Using the Menu Repository

To define options for the Menu repository:

1. From the **Navigator** pane, select the **Rights** repository
2. Define rights in the **Rights** repository.
3. From the **Settings menu**, select the **User ID** option. From the **User ID** repository, assign the defined rights to the appropriate users in the **Rights** column.

4. From the **Navigator** pane, select the **Menu** repository.
5. Select the desired pull-down or context menu and zoom to the **Menu Definition** list. Press **CTRL+P**.
6. From the **Menu Properties** dialog box, click the **Properties** tab, and zoom to the **Rights** field.
7. Select one of the pre-defined rights, as defined in **Step 1**, as the right for this menu entry. Click **Select**.
8. Repeat steps **5-8** to define options for another menu entry.

Defining Options Using the MNUENABL and MNUSHOW functions

You can also change the menu entries a user sees in runtime by using the MNUENABL and MNUSHOW functions.

To enable or disable a menu entry:

1. Use the **MNUENABL** function to enable or disable a menu entry according to a specific condition that would be set in runtime, in addition to the assigned rights of the menu entry.
2. To hide or show a menu entry:
3. Use the **MNUSHOW** function to hide or show a menu entry according to a specific condition that would be set in runtime, in addition to the assigned rights of the menu entry.

Assigning Help Screens

You can assign help screens, either internal, Windows, or URL help, and help prompts to an entry in a menu.

To assign help screens for entries in a menu:

1. From the **Navigator** pane, define help screens in the **Help Screens** repository.
2. From the **Navigator pane**, select the **Menu** repository.
3. Select a pull-down or a context menu and zoom to the **Menu Definition** list.
4. Press **CTRL+P** for the specific entry to assign a help screen.
   Note: you cannot assign a help screen to only System, Program, or OS Command entry types.
5. From the **Menu Properties** dialog box, click the **Properties** tab, and zoom to the **Help** field.
6. Select one of the pre-defined help screens as the associated help screen for this entry. Click **Select**.
7. Repeat **Steps 4-7** to assign a help screen to another menu entry.

Assigning Help Prompts

You can assign help screens, either internal, Windows, or URL help, and help prompts to an entry in a menu.

To assign help prompts, or status bar messages, for entries in a menu:

1. From the **Navigator** pane, define help prompts in the **Help Screens** repository.
2. From the **Navigator** pane, select the **Menu** repository.
3. Select a pull-down or a context menu and zoom to the **Menu Definition** list.
4. Press **CTRL+P** to assign a help prompt to a selected entry.
   Note: You cannot assign a help prompt to a System, Program, or OS Command

entry type, but not tan an Menu entry type.

5. From the **Menu Properties** dialog box, click the **Properties** tab of the **Menu Properties** dialog box, and zoom to the **Prompt** field.
6. Select one of the pre-defined help prompts, as defined in **Step 1**, as the associated help prompt for this entry. Click **Select**.
7. Repeat **Steps 4-7** to assign a help prompt to another entry.

Defining an Application Default Menu

For every new application, eDeveloper generates a default context menu that is empty and you cannot delete it. You can have more than one context menu in your application, but only one context menu can be the default.

To define a default context menu:

1. From the **Navigator** pane, select the **Menu** repository.
2. Select Default Context menu and zoom to the Menu Definition list.
3. Add and remove entries from the **Default Runtime Context** menu.

Defining an Entry Image on the Runtime Context Menu

Context menus appear when the user clicks the right mouse button in Runtime mode. This topic discusses how to define an image for a context menu entry.

To define the image:

1. From the **Navigator** pane, select the **Menu** repository.
2. Select a context menu and zoom to the **Menu Definition** list.
3. Press **CTRL+P** to assign an entry image to a selected entry.
4. From the **Menu Properties** dialog box, click the **Toolbox tab**, and from the **Image For** combo box select **Menu**.

5. Select either a bitmap of your own for the image or an internal eDeveloper image:
   For your own bitmap, you need to specify the location of the BMP file in the **Tool Image** field.
   For an internal eDeveloper image, zoom to the **Tool Number** field and select one of the displayed images.
6. Click **OK** to return to the **Toolbox** tab, and click **OK** to approve the entry.

Defining a Context Menu for a Specific Program

Context menus appear when the user clicks the right mouse button in Runtime mode. This topic discusses how to assign a context menu to a program.

To define the context menu:

1. From the **Navigator** pane, select the **Menu** repository.
2. Create a new context menu, or use an existing context menu – not the default one.
3. From the **Navigator** pane, select the **Program** repository.
4. Zoom to a specific program you wish to assign a context menu to
   -OR-
   Press CTRL+P to open the **Task Properties** dialog box.
5. From the **Task Properties** dialog box, click the **Advanced** tab, and zoom to the **Attached Context** field.
6. Select one of the pre-defined context menus, as specified in step 2, as the associated context menu for this entry.
7. Click **OK** to return to the **Task Properties** dialog box, then click **OK** to approve the entry.

Executing a Program from the Menu with Arguments

This topic discusses how to add arguments to the program option of the menu so it will be passed on to the called program.

To add an argument:

1. From the **Navigator** pane, select the **Menu** repository.
2. Select a pull-down or a context menu and zoom to the **Menu Definition** list.
3. Press CTRL+P for the specific entry you wish to send arguments to.
   Note: You can send arguments only to a **Program Entry** type.
4. From the **Menu Properties** dialog box, click the **Properties** tab, and zoom into the **Arguments** field.
5. Add the arguments you wish to send to that program as global variables, defined in the main program.
6. Click **OK** to return to the **Menu Properties** dialog box, and click **OK** to return to the **Menu Definition** table.

Creating Additional Context Menus

In the Menu repository you can define additional context menu structures to have as many context menus as you want in an application.

To define a menu:

1. From the **Navigator** pane, select the **Menu** repository.
2. Create a new entry in the **Menu** repository or zoom to the **Menu Definition** list and create a new menu type entry.
3. Press CTRL+G to open the **Generate Menu Form** dialog box.
4. Select the desired program range by typing the program's numbers or by zooming to the **Program** list.
5. Click **OK** to return to the **Menu** repository.

Defining a Shortcut Key for Menu and Separator Entry Types

You can select a menu entry by pressing a shortcut key to open the menu entry.

To designate a shortcut key for one letter:

1. From the **Navigator** pane, select the **Menu** repository.
2. Create a new entry in the **Menu** repository or zoom to the **Menu Definition** list and create a new **Menu type** entry.
3. In the **Entry Text** column, add the character *&* before the letter that should select that menu entry.
   For example, if you specify *Ed&it* in the entry text, when the user presses the *i* key, this selects the Edit option on the menu.
4. Click **OK** to return to the **Menu** repository.

Defining a Shortcut Key for Program, OS Command, and Event Entry Types

You can select a menu entry by pressing a keyboard combination to open the menu entry.

To designate a shortcut key combination:

1. From the **Navigator** pane, select the **Menu** repository.
2. Select the **Default Pulldown Menu** entry and zoom to the **Menu Definition** list and create a new menu entry type.
3. In the **Acc Key** column of that entry zoom to the **Key Definition** box, and press the key combination that will be the shortcut key for that menu entry.
   In runtime, pressing this key combination, such as **CTRL+Z**, activates this menu entry anywhere in the application. Shortcut keys made in this manner will not override previously made shortcuts using the same combination.

Note: This procedure is for the following menu Entry Types:

- Program, OS Command, Event

You should not use it for Menu or Separator entry types.

Creating Application Helps 12

eDeveloper allows you to provide help for end-users at appropriate points in an application. You can design the help to be displayed when the user requests help from anywhere in the program. In addition, you can provide prompts and tooltips that are displayed automatically. When you define a column, type, form, control, or menu item, you can also specify an associated help.

This chapter covers the topics listed below:

- Creating Internal, Prompt and Tooltip Helps
- Designing WinHelp Help Topics

Creating Internal, Prompt and Tooltip Helps

To define an Internal, Prompt or Tooltip Help:

1. From the **Workspace** menu click **Help Screens** option to display the **Help Screens** repository.
2. Press **F4** to create a new line, and enter a name in the **Name** column.
3. Choose the help type from the Type column: Internal, Prompt, Windows, Tooltip, or URL.
4. From the **Name** column zoom to enter the help text.
5. In the **Properties** sheet of your form, column, model or control zoom from the **Help screen**, **Tooltip** or **Help prompt** property to select the appropriate help from the **Help List**.

Note: If a column is associated to a model, the internal, prompt and tooltip help parameters are copied from those defined for the model. eDeveloper keeps the column's inheritance of the model's help parameters, unless you change these values.

Designing WinHelp Help Topics

An external Help is a help file that is called by the Windows help utility, Winhelp.exe. The online Help for eDeveloper is an example of this. Windows help files are created in a word processor or specialized Windows help authoring tool.

You can set eDeveloper to call the Windows help facility to display a Windows Help file. Windows Help can be attached to types, columns, forms, and programs. All the Windows Helps for an application are stored in one Windows Help file.

To define or edit WinHelp screens:

1. From the **Workspace** menu click **Help Screens** option to display the **Help Screens** repository.
2. Press **F4** to create a new line, and enter a name in the **Name** column.
3. In the **Type** column select the **Windows** option.
4. Press **CTRL+P** to open the **Windows Help Properties** sheet. In the **Details** property, enter the name of the Windows Help file.
5. From the drop-down list select the Help command that you want to send to the WinHelp engine.
6. In the **Help Key** parameter, type the context number of the Help topic.
7. Press **F7** to test the help topic.

Using eDeveloper Toolkit Utilities

13

Magic includes a variety of utilities designed to help you to develop, modify, maintain, and move your application.

This chapter covers the topics listed below:

- Porting Your eDeveloper Application
- Creating an eDeveloper Application Documentation File
- Cross-Referencing an Object
- Organizing Your Application

Porting Your eDeveloper Application

The Magic Flat File (MFF) lets you deploy a database-independent eDeveloper application file. The application, created as a binary file, can be stored anywhere on the user's computer system and is used for deployment only.

This section includes the topics listed below:

- In the Development Environment
- In the Runtime Environment

In the Development Environment

To create a Magic application flat file in the Development environment:

1. Open your application.
2. From the **File** menu click **Save as MFF**.
3. Name the Magic Flat File (MFF) and press Save.
   This creates an MFF file for your application. Move this Magic Flat file to the directory where you'll run your application.

In the Runtime Environment

To create a Magic application flat file in the Runtime environment:

1. From the **Settings** menu, click **Applications** option and select your application.
2. Press **CTRL+P** to open the **Application Properties** dialog box.
3. Check the **Flat MCF deployment** check box.
4. Click **OK**.
   eDeveloper runs the MFF file instead of the MCF file for this specific application.

Creating an eDeveloper Application Documentation File

The eDeveloper Documentation Template utility is used to create hard copies of the eDeveloper elements that are associated with the various eDeveloper structures for a particular application, such as tables, repositories, dialog boxes and end user forms. These hard copies serve as developer documentation.

You can also use the default documentation template file, DOC_ STD. ENG, or DOC_ EXT. ENG as a guideline. These files, which create documentation for all data items, are part 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.

Running the Export Utility in Documentation Mode

To run the export utility in Document mode:

1. Open your application. From the **Settings** menu click **Environment** and click the **External** tab.
2. Place the cursor on the **Documentation Template File** property and enter the name of your documentation template file.
3. Press **SHIFT+F10** to open the **Export/Import** dialog box.
4. From the **Operation** list select **Export Document**.
5. From the **Type** list select the eDeveloper component you want to document (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 -9**, each time selecting a different eDeveloper component for the **Type** list.
6. 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** section's **From** and **To** fields to indicate the range of occurrences you want to document.

7. In the **File Name** field type an output file name.
   The documentation generator chooses 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.
8. Click **OK** to confirm your export settings.
   If the documentation template has a syntax error in the section you have specified, you receive an error message, and the process terminates.
9. If you want to create documentation for other components, return to the Type combo box and select the next type.
   You can interrupt the documentation generator at any point by clicking **Exit** or pressing **ESC**.

Cross-Referencing an Object

The eDeveloper 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 eDeveloper cross-reference utility lets you find information about the following objects:

Modal	Table
Model	Index
Program	Help screen
Right	Menu
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.

This section includes the topics listed below:

- Selecting Entries to Cross-Reference
- Deleting or Searching for a Cross-Reference
- Saving or Printing Cross-Referenced Information
- Changing the Maximum Number of Cross-Referenced Results

Selecting Entries to Cross-Reference

To select entries to cross-reference in a repository:

1. In an eDeveloper repository select an entry and press **CTRL+X**.
2. In the **X-Ref** dialog box check the boxes for the repositories in which you want to cross-reference your object.
3. Click **OK**.
   The results are displayed in the cross-reference tab of the Navigation Pane.
4. Click the result entry to display the corresponding repository in the **Workspace** pane.

Deleting or Searching for a Cross-Reference

To delete a cross-reference item:

On the **Navigator** pane click **X-Ref** and select a cross-reference item. Press **F3** to delete the cross-reference.

To search for a cross-reference item:

1. In an eDeveloper repository select an entry and press **CTRL+X**.
2. In the **X-Ref** dialog box check the boxes for the repositories in which you want to cross-reference your object.

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 or Printing Cross-Referenced Information

To save cross-reference results:

From the **File** menu click **Cross-Ref Result** and then click **Save Result**.

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. Click the **Preferences** tab.
3. In the **Maximum number of X-ref results** property enter a figure for the desired maximum number of cross-reference results.

Organizing Your Application

eDeveloper has features that allow you to:

- define **Folders** for an application.
- use **Bookmarks** for direct access to a desired object.
- add **Comments** to document your application.

- create and use **Components** in other applications.
  For more information, see Chapter 14, Using eDeveloper Components.

This section includes the topics listed below:

- Using Comments
- Bookmarking a Location
- Using Comments

Creating eDeveloper Folders

To create eDeveloper folders:

1. From the **Navigator** pane select either the table, model, program, rights, help screens, or components repositories.
2. Press **F4** to create a new folder.
3. Give the new folder a name, for example **Reports**.
4. Repeat **steps 2-3** to create other folders.
5. Move to the repository and select the desired folder for each object.

Bookmarking a Location

You can easily bookmark locations in your application for later access. The bookmarks are stored within the application file and are available through the navigator screen.

At any point during application development, you can save a bookmark of the position you are at.

To bookmark a location:

1. Select the desired object
2. From the **Options** menu, select **Bookmark** or press **CTRL+B** at any point within

the application.
The **Bookmark** box opens.

3. In the **Bookmark** box enter a name for your bookmark and click **OK**.

To return to a location using a bookmark:

1. In the **Navigator** pane (ALT+F1), open the pull-down menu.
2. Select the **Bookmark** item to see the list of bookmarks.
3. Select the required **Bookmark** and zoom to the entry.

Note: Bookmarks are shared by all users. Therefore it is advisable that you give each bookmark a descriptive name, so that any user will be able to understand to where the bookmark is pointing.

Using Comments

Comments enable you to internally document your application.

To add comments to an application object:

1. Select the item, such as a task or table.
2. Press F10 to open the **Comments For** text box.
3. Enter your comment.

Using eDeveloper Components 14

In eDeveloper, you can define application objects as components and then use them in other parts of your application or other separate applications.

A component is an eDeveloper application file with an interface. Using the interface, another eDeveloper application can call an object within the first application, and execute or use the object as if it was part of the host application. The file you create is called a Magic Component Interface (MCI) file.

You can create components for the objects listed below:

Models	Rights
Tables	Events
Programs	Application properties
Helps	Environment settings

This chapter covers the topics listed below:

- Creating a Component
- Loading a Component
- Integrating Components into Your Application
- Sharing an Event Among Applications
- Maintaining the Loaded Component Application

Creating a Component

To create a component:

1. Open the application you want to use to build your Component, and from the **Options** menu choose **Component Builder**.
   This opens the Component Builder, which is divided into an upper and lower table.
2. In the upper table press **F4** to create a new entry, and give your component a name.
3. Zoom from the column for the object you want to select, to create a line in the lower table.
4. Click **Add Items** to select the objects you want to build into your Component and then click **OK**.
   A selection window opens showing the objects available for selection. For example, if you selected the **Tables** tab, the window displays the tables available for selection.

   Note: You will only be able to select those objects that have been assigned with a public name.

To specify a path for the MCI file, and to complete its build:

1. After you finish entering the objects in all the columns, from the **Component Builder** menu click **Build Interface File**.
2. Specify the designated path and name for the **MCI** file. Zoom to browse your disk drives.
3. Click **OK** to create the MCI file.
4. From the **File** menu click **Close Component Builder**.

Loading a Component

You can use components for specific application operations that can be used by other applications. By exporting components you can share the resources you create in one application among a number of other applications. You share the objects by loading the component into your application.

To load a component:

1. Open the application in which you want to use the component.
2. From the **Navigator** pane select the **Components** repository.
3. Press **F4** to create a new line.
4. Press **F5** to open the **Open file** dialog box.
5. Select the required component interface file and click **Open**.
   This loads the component and you can now use it in your application.

Integrating Components into Your Application

When you open a selection list in an application with a component, the **View** list at the top of the window enables you to define the objects for selection, as follows:

- **All** – All the objects are available in the selection window from the current application and from components. The component object's prefix is the component name from the component repository.
- **Internal** – Only objects from the current application are available.
- **Component name** – All the components names from the Component repository appear. You can select to view one object from within a specific component.

Selecting a Component for Integration

The steps below illustrate the procedure for selecting a table component.

1. Open the **Program** repository and press F4.
2. Press CTRL+G to activate the Automatic Program Generator (APG).
3. Zoom from the **Main Table** field to open a selection window with both the current application's and the component's tables. The component table appears in red.
   You can use the **View** list to limit the tables available for selection.
4. Select the required table in order to create a program based on a component table, as opposed to a table from the current application's Table repository.

   Note: In the Component Builder you can only select objetcs that have been given a public name.

Sharing an Event Among Applications

An application Event Handler you define as a Global Event in the Main Program, can be caught in a component when the event is fired in another component, or in the main application itself, the host application.

A component, that includes an Event Handler in the component main program that you define as a Sub-Tree, can also be caught in the component when the program which fired the event is also part of the component.

The search path for the Event Handler is always as follows:

- The starting point is the task where the event was fired up in the Runtime tree, to the Main Program of the host application.
- After this climb, the event handler is searched in the loaded components and Main Program, but only for Handlers defined as Global.

Host application event handlers defined as Global, will be treated as subtask handlers. The component application's event handlers, which are defined as subtask, and fired from a program in the same component application, will be treated as Global handlers.

Maintaining the Loaded Component Application

When you create a component, you should include information about the help, colors and fonts files that are used in that specific component. If you do not include this information when you create the component, the host application's environment will take effect.

In Runtime, once the component is loaded, some of the component application's environment properties, such as Database, DBMS, Server and Services properties and Logical Names, are added to the correct Magic.ini sections in the of the host application.

Note: When deleting a component from the host application, the added properties are not removed automatically from the Magic.ini file.

You can set special environment values when you create a new component in the Component Interface Builder. These values are stored within the MCI file. When loading the component into the host application these values are entered into the host environment.

Adding Settings to a Component

To add settings to a component:

1. From the **Options** menu click **Component Builder** to open the Component Builder.
2. From the **Environment** column zoom to open the **Environment** selection table.
3. Click a tab and then click **Add Items** to open the tab's selection list.
4. When you have made your selection click **OK** to add the settings to your component.
5. From the **Component Builder** menu click **Build Interface File** to save your component as an **MCI**.

Partitioning eDeveloper 15

eDeveloper 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.

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 remote service, and send task requests for execution to a server. In more advanced cases, the application logic can be distributed among several servers.

This chapter covers the topics listed below:

- Setting Up an eDeveloper Partitioned Application
- Retrieving Broker Information From the Command Line
- Running a Remote Program From the Command Line

Setting Up an eDeveloper Partitioned Application

You can choose which application components you want to run on the server side (remote engine) and which components will run on the client side (local engine).

Background tasks should be selected as remote executed tasks.

Setting up an eDeveloper Partitioned Application requires the following:

1. Partitioning the eDeveloper Application
2. Knowing How a Partitioned Application Works
3. Setting the eDeveloper Application
4. Setting Up the Server
5. Setting Up the Client
6. Using eDeveloper Partitioning

Partitioning the eDeveloper Application

eDeveloper supports partitioning of the application logic among one client and one or more servers. Partitioning the application means that part of the application runs on the client machine and other parts run on other server machines. In addition to the eDeveloper client installation, eDeveloper Enterprise installation should be performed on each machine running an eDeveloper server. Only batch processes can be run on the server side.

Knowing How a Partitioned Application Works

An eDeveloper engine should be started and connected to a messaging server. The eDeveloper client issues a Call Remote for a service and a specific program (public name). The request is sent to the eDeveloper server which processes the request.

There are two request types:

- Synchronous requests - The eDeveloper client waits until the server finishes processing the request.
- Asynchronous requests - These are sent from the client, and the client immediately continues processing without waiting for the server to finish its processing.

For each Call Remote operation, the Wait flag determines the type of request (**Yes** = Synchronous, **No** = Asynchronous). A messaging server is also required to send requests from the eDeveloper client to the eDeveloper server. The Magic Broker is one of the messaging servers that eDeveloper works with.

The servers and the client can share the same MCF file or use separate MCF files.

Setting the eDeveloper Application

Set the public name for eDeveloper programs that will be called from another eDeveloper client or server.

Setting Up the Server

To set up the server:

1. Set up the Magic Server to connect to a messaging server.
2. Define a messaging server in the **Servers** list.
3. Set **Activate as Enterprise Server** to **Yes** and set the **Messaging Server** to the appropriate Server in the Partitioning tab.
4. Two optional settings:
   In the **System** tab set a **Start Application**, you can command the Magic server to open an MCF file in advance.
   In the **Partitioning** tab if you set the **Enterprise Server can change application** property to **No**, you can disable the replacement of an already loaded MCF.

When you use the Magic Broker as the messaging server, and set a password for accessing the Broker (in the Mgrb.ini file), the same password should be set in the password field in

the properties of the Magic Server you defined. It is also recommended to use a secret name for this field.

Setting Up the Client

To set up the client:

1. Set up a Magic Server.
2. Enter a name (descriptive name).
3. Set the **Server** property as the messaging server defined in the **Servers** list.
4. In the **Remote Application** property, enter the remote application name.
5. In the **Call Remote** properties window choose the service, the public name of the called program, and the return codes.

Using eDeveloper Partitioning

Before the eDeveloper client can issue Call Remote requests, the Magic server engines must be started. Each one should be connected to a messaging server.

The eDeveloper client can now issue requests.

Retrieving Broker Information From the Command Line

You use the MGRQCMDL utility with the different tags to learn about the state of the broker and the applications attached to the broker. The Magic Command Line Requester (**Mgrqcmdl.exe** for Windows, **Mgrqcmdl** for Unix) can be used to query the Magic Broker. It has a set of parameters that enable querying and sending commands to the Magic Broker.

- To list the Magic Engines connected to a Magic Broker:

  ```
  mgrqcmdl -query=rt
  ```

  The output contains a list of Magic Engines that are currently connected to a Magic Broker.

- To list the applications that the Magic Engines serve:

  ```
  mgrqcmdl -query=app
  ```

- To list the requests a Magic Broker received:

  ```
  mgrqcmdl -query=log
  ```

  The requests are listed in descending order (starting with the last request that the Magic Broker received).

- To list a range of request numbers:

  ```
  Use -query=log=100-90.
  ```

- To limit the list to a single application:

  ```
  Use -query(app_name).
  ```

- To list the requests waiting in queue:

  ```
  mgrqcmdl -query=queue
  ```

Running a Remote Program From the Command Line

You use the MGRQCMDL utility with the tags to execute a program in a remote eDeveloper Application.

The Magic Command Line Requester (mgrqcmdl.exe on Windows, mgrqcmdl on Unix) can be used to call remote programs on Magic Engines that are connected to a Magic Broker.

To run a remote program:

1. The minimum information for calling a remote program is:
   an application name: (-appname=my_appl)
   and a program's public name: (-prgname=my_remote_prg)

   ```
   mgrqcmdl –appname=my_appl –prgname=my_remote_prg
   ```

2. To specify a username and password, use the parameters:

   ```
   username=my_user and –password=my_pass
   ```

3. When the output of the program is a Requester, you redirect the output to a file using the parameter:

   ```
   -filename=request.out
   ```

4. You supply additional arguments using:

   ```
   -arguments=… and –variables=…
   ```

5. To set the priority of the request (between 0 and 9) use:

   ```
   -priority=5
   ```

Connecting to External Applications

16

eDeveloper can receive data from other applications using Dynamic Data Exchange (DDE), OLE Automation, and Call Operations for a DLL file or a 3rd Generation Language.

This chapter covers the topics listed below:

- Calling eDeveloper from an External Application
- Calling an External Application Using the Exit Operation

Calling eDeveloper from an External Application

You can connect from non-eDeveloper clients to an eDeveloper server. eDeveloper supports connections using standard methods such as Enterprise Java Beans.

Calling an External Application Using the Exit Operation

The Exit operation lets you run external programs and script files from eDeveloper. You can perform the Exit operation using an expression and eDeveloper then sends a command to the operating system. The command can include any parameters that are used when calling an external program from a command line, such as cmd on Windows NT.

Examples of expressions are:

- notepad list.txt
- external.bat
- output.log

This section includes the topics listed below:

- Using the Exit Operation in a Client/Server Environment
- The Wait Property
- The Show Property
- The Ret Property
- Troubleshooting

Using the Exit Operation in a Client/Server Environment

When using eDeveloper Client/Server architecture, you can start running any program on the server, the machine where the data server runs, by specifying the server name, in parentheses, in the expression. For example:

```
'(unixsrvr) prog1'
```

The Wait Property

The Wait property controls whether an eDeveloper program waits until the external program is completed.

- When you set the **Wait** property to **No**, the eDeveloper program does not wait for the called program to finish before continuing.
- When you set the **Wait** property to **Yes**, the eDeveloper program waits for the called program to finish before the eDeveloper program continues.

The Show Property

The Show property controls the initial behavior of GUI external programs that are called.

The valid values are: **Normal**, **Hide**, **Minimize**, and **Maximize**.

The Ret Property

The Ret property, which is optional, lets you specify a numeric variable that will receive a return code. If the returned value is greater or equal to zero, the external program was successfully executed by eDeveloper, and the return code should be interpreted according to the specifications of the external program. If the value returned is less than zero, the command failed, which usually happens because a wrong program or path name was specified.

Troubleshooting

If you have problems starting a program using the Exit operation, try issuing the same command from the command line interface.

- For Windows use cmd or DOS window
- For UNIX use the terminal window (like xterm)

Improving Performance 17

There are some procedures you can use to improve eDeveloper performance.

This chapter covers the topics listed below:

- Influencing the DBMS Optimizer
- Using RDBMS Features
- Repeatedly Calling a Task
- Accessing a Heavily Used Table

Influencing the DBMS Optimizer

eDeveloper allows for influencing the DBMS optimizer using DBMS hints. eDeveloper supports sending hints to the DBMS in order to change or influence the optimizer's behavior or priorities.

This section includes the topics listed below:

- Prioritizing the Hints
- Examples of Hints

Prioritizing the Hints

eDeveloper can send hints to the database at the following levels:

- Index entry level
- Table entry level
- Database entry level

The priority is: index, table, database. This means that if there is a hint in both index and table level, the index hint is sent to the DBMS. For example, if you specify FORCE_INDEX hint:

In the Properties dialog of the	Effect
Index	Applies only to the index
Table	Applies to all indexes in the table
Database	Applies to all indexes in all tables in the database

Hints are a method of changing the DBMS optimizer's behavior, and must be used with special care and caution since the effect can be serious.

Hints can be of the following nature:

- The optimization approach for a SQL statement
- The goal of the cost-based approach for a SQL statement
- The access path for a table accessed by the statement
- The join order for a join statement
- A join operation in a join statement
- The locks to be set for a table being accessed
- The isolation level to be used

Examples of Hints

Database

- Oracle:
  /*+ FIRST_ROWS */
- SQLserver:
  FASTFIRSTROW
  SERIALIZABLE

Table

- Oracle:
  /*+ ROWID */
- SQLserver:
  HOLDLOCK
  LOOP
  MERGE
  FORCE ORDER

Index

- Oracle:
  /*+ (INDEX table index_name)
- SQLServer:
  INDEX = index_name

You can change the session's parameters constantly using a Direct SQL command, such as (for Oracle):

```
ALTER SESSION SET OPTIMIZER_GOAL = FIRST_ROWS
```

Using RDBMS Features

This topic discusses the use of specific DBMS features in order to enhance performance and those eDeveloper features that affect the DBMS.

The following DBMS features can be used within eDeveloper programs:

- Direct SQL statements
- Using views
- DBMS Stored Procedures
- RDBMS sort
- Sequence / Identity

You should examine thoroughly the following features should be thoroughly examined:

- Check existence flag
- Non-unique keys
- Range expression
- Mixed segments in eDeveloper indexes

Improving Performance Using DBMS Features

In order to improve performance, you should consider implementing the following list of eDeveloper-supported DBMS features:

- Direct SQL
- Database views
- Database stored procedures
- RDBMS sort
- Sequence

Direct SQL

- DDL statements such as
  INSERT into TABLE1 select * from TABLE2
- Aggregate functions such as
  SELECT sum(FIELD1), AVG(FIELD2) from TABLE1
- Massive UPDATE/DELETE such as
  DELETE from TABLE1 where …

Database Views

Views are a means of defining a range of records over one or more joined tables. Since a view is only a definition, it does not take up any database space. The view can include aggregate functions, computed fields, and joined tables.

Database Stored Procedures

Stored procedures are procedural SQL code that can be executed by eDeveloper. A stored procedure can include SQL code such as cursor handling, DMLs, and, DDLs.

RDBMS Sort

The database can sort data much faster than eDeveloper. However, eDeveloper requests a database sort in the following cases:

- When an eDeveloper index is used.
- When an eDeveloper sort is defined over fields from the main, or link Joined tables.

In other cases eDeveloper will execute the sort internally.

Sequence / Identity

In cases where a counter file has to be created, databases supply built-in counters.

eDeveloper supports MSSQLserver by specifying IDENTITY in the TYPE property of a column in the table repository.

eDeveloper supports Oracle SEQUENCE by Direct SQL commands as *SELECT sequence_name.NEXTVAL from DUAL* and *SELECT sequence_name.CURRVAL from DUAL*.

Gaining Performance

In order to improve performance, you must examine the following list of eDeveloper/ DBMS features, and check their effect on the application.

- **Check Existence: You** can request eDeveloper to check for the existence of every table that you open. This action takes time. This might be an unnecessary step since these tables might be known to exist.
- **Non-unique eDeveloper indexes:** When eDeveloper uses a non unique index as its main/linked table index, it adds the table's position to the end of the ORDER BY clause sent to the DBMS. This rule is valid for main tables in online tasks and for all linked tables.
  An ORDER BY clause including the segments from the table's position might cause performance problems, especially if there is no corresponding DBMS index to support the query at hand, which is usually the case.
- **Range expression:** eDeveloper evaluates this expression for every record retrieved from the DBMS in order to determine whether it is considered a part of the dataview. This might cause performance problems since a record has to reach the client in order to be determined as part of the dataview. Use the eDeveloper SQL WHERE clause or the DB SQL WHERE clause to solve this problem. The client asks the DBMS to return only the records that belong to the dataview for processing.

Repeatedly Calling a Task

When a task is repeatedly called, you can define it as resident. This means that the eDeveloper code for this task is loaded into memory and can be re-called for use. This also affects the cursors being used by the task, and on the linked data being accessed by the task.

To make a resident task:

There are two conditions that have to be met in order for a task to be resident:

1. In the **Task Properties** dialog box click the **Advanced** tab and set the **Resident Task** property to **Yes**.
2. Open the files being accessed by this task in the calling task or in any task above it in the runtime tree.

Once these conditions are met, the task becomes a resident task. This means that:

- The code generated by this task is loaded into memory and is released only at a later stage. A non-resident task is always retrieved from the application file itself.
- The cursors opened by the resident tasks are prepared once, upon first entering the task. Once re-entering the task, the cursors are closed and re-opened. These cursors are freed once the tables accessed by the resident task are closed. You can determine the stage at which the cursors are free by closing the tables accessed by this task at different stages in the runtime tree.
- The cache, accumulated by the resident task for linked tables, stays available for the next time this task will be called. The cache is cleared once the linked table accessed by the resident task is closed.

This behavior enhances performance since it requires less I/O for code access, cursor preparation, and data access.

Accessing a Heavily Used Table

eDeveloper allows the developer to define a table entry as resident. Preloading a table means that its data will be read and stored in the client's memory, and all access will be through the loaded data in memory.

The programmer and the user can decide whether resident tables will be pre-loaded. The data can be pre-loaded upon opening the application, or upon opening the table, depending on the parameters specified in the DB Tables list of the running tasks.

To pre-load a table's data:

1. Open the **Table Repository** and press CTRL+P to open the **Table Properties** dialog box.
2. Click the **Advanced** tab and set the **Resident** property to either **Immediate** or **On demand**.

When using the toolkit for debugging programs, tables are pre-loaded once the toolkit switches to runtime mode. They are released once you switch back to toolkit mode.

To eliminate the loading of the tables to memory when going to runtime:

In the **Magic.ini** file's **[MAGIC_ENV]** section set the **LoadResidentTables** setting to **No**.
This can also be used by the user to eliminate the need to load resident tables.

Deploying eDeveloper 18

eDeveloper deployment involves several aspects related to the configuration environment. You should properly configure your deployment environment in order to optimize performance.

This chapter covers the topics listed below:

- Creating and Using a Magic Flat File
- Setting Up a Multi-Threaded Environment
- Managing a Multi-Threaded Environment
- Setting a Single Context Environment

Creating and Using a Magic Flat File

The Magic Flat File (MFF) lets you deploy a database-independent Magic application file (MCF). The eDeveloper application is stored as a compressed binary file. You can only modify the MCF on which the MFF is based. An MFF file cannot be modified and it can only be used for deployment purposes only. eDeveloper accesses the MFF directly without the involvement of any database gateway.

This section includes the topics listed below:

- Creating a Magic Flat File
- Running an eDeveloper Application Using a Magic Flat File

Creating a Magic Flat File

To create an MFF:

1. In Toolkit mode open an eDeveloper application.
2. From the **File** menu click **Save As MFF** to open the **Save As** dialog box.
3. Enter the name and location of the **MFF** file.

Running an eDeveloper Application Using a Magic Flat File

There are two ways to setup an application to run as a Magic MFF. The first method involves setting the Magic.ini file to run the application as a Magic Flat file.

To set the Magic.ini file:

1. From the **Settings** menu click **Applications** and park on your application.
2. Press **CTRL+P** to open the **Application Properties** dialog box.
3. Check the **Flat MFF deployment** box.
4. Click **OK**.
   Note: An MFF file can only be run in eDeveloper Runtime mode. If the application's file name is not specified, the application's prefix is used.

You can also run an MFF in Runtime mode as follows:

Call the MFF file from the command line prompt using the following parameter:

```
/MFF=<flat MCF file name>
```

Note: Make sure that for each MFF file, you have a corresponding MCF file saved in your system. The eDeveloper Toolkit mode cannot read MFF files. eDeveloper components can also be stored as MFF files, but you need to create a separate MFF file for each application that will be used as a component.

Setting Up a Multi-Threaded Environment

The eDeveloper engine is a multi-threaded engine that runs in the background. One application server can serve concurrent users by using multiple threads. For each request, eDeveloper creates a new thread whose sole purpose is to serve the request. The thread remains only as long as eDeveloper operates on the browser side when the request arrives from an eDeveloper interactive web application client.

To set up a Multi-Threaded environment:

1. From the **Settings** menu, click **Environment** option to open the **Environment** dialog box.
2. Click the **System** tab, and set the **Application startup mode** property to **Background**.
3. Restart **eDeveloper**.

Limiting the Number of Concurrent Threads

You can limit the maximum number of threads by defining the requisite environment setting to a non-zero value. If the value of this property is zero, there will be no user limit.

To define the number of concurrent threads:

1. From the **Settings** menu click **Environment** to open the **Environment** dialog box.

2. Click the **Enterpise Server** tab, and in the **Maximum number of concurrent requests** property enter a value other than zero.

The maximum number of concurrent threads is further limited by two factors:

- Your eDeveloper license: The number of users set in the license cannot be exceeded. This means that the total number of threads cannot exceed this value.
- The operating system limitation: This is due to a possible limited amount of resources.

Sending Requests to Multiple eDeveloper Applications

To serve requests by two or more separate eDeveloper applications, you must start an eDeveloper engine separately for each eDeveloper application.

Note: All threads of a single eDeveloper Enterpise Server serve the same application.

Managing a Multi-Threaded Environment

The eDeveloper Enterpise Server is a multi-threaded engine and can serve concurrent requests through separate threads. This topic explains how to monitor and manage an eDeveloper multi-threaded environment.

Monitoring and managing an eDeveloper multi-threaded environment involves:

- Monitoring eDeveloper threads
- Starting an eDeveloper application server
- Shutting down an eDeveloper application server

This section includes the topics listed below:

- Monitoring eDeveloper Threads
- Starting or Closing an eDeveloper Enterpise Server

Monitoring eDeveloper Threads

To monitor eDeveloper threads:

1. Open the **eDeveloper Broker** menu.
2. Select **Display AppServers**.
   A line is displayed for each Enterpise Server or Magic engine. The number in parenthesis displays the number of additional threads that eDeveloper can start. This number depends on the **Maximum number of concurrent requests** setting in the Environment settings and the eDeveloper license.

   Only a maximum number of 10 requests can be served concurrently and other requests will wait in the Magic Broker queue.

   For example, if your license allows 10 concurrent users, and you did not limit the number of threads using *Maximum number of concurrent requests*, then when you start an eDeveloper Enterpise Server, you see the number 10 in parenthesis. This number goes down to 0 when loading the Enterpise Server with many requests.

You can also monitor the number of threads used by eDeveloper using the Operating System tools. For example: On Windows NT/2000 use the Task Manager. The total number of displayed threads is larger than the number of threads eDeveloper starts for serving requests. This is because eDeveloper reserves several threads for internal use.

The eDeveloper **RqRtInf** function can also be used to receive information on the Enterpise Server including the:

- current number of busy-work threads
- maximum number of work-threads allowed
- highest number of busy work-threads since the Enterpise Server's startup

Starting or Closing an eDeveloper Enterprise Server

To start an eDeveloper Enterprise Server:

- Use the eDeveloper **RqExe** function.

To close an eDeveloper Enterprise Server:

- Use the eDeveloper **RqRtTrm** function. A Logical parameter allows you to control whether the Enterprise Server will stop immediately, or only after completely serving the executed requests.

Setting a Single Context Environment

This topic discusses how to set up the eDeveloper environment of a single context. The eDeveloper environment of a single context involves the following:

- Runtime context
- Magic.ini and INIPut function
- Resources shared by threads
- Using external programs (UDF/UDP)
- CTX functions

Runtime Context

The eDeveloper Enterprise Server manages:

- contexts, which are internal descriptions of the exact location of clients in the task.
- manipulated data within the task's transaction.

Each runtime context has a separate memory area. The context includes: Magic.ini settings, a Security file (usr_std), and a Main Program (virtual variables).

The Magic.ini and the INIPut Function

For each Runtime context, the Magic.ini file is stored 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 force write flag:

```
INIPut (<assignment>,<force write>) (default - N)
```

When the Magic.ini file is resident, modifications by the INIPUT function are done only in memory.

Resources Shared by Threads

DBMS connections and external devices such as printers and I/O files, are shared by threads. There is no DBMS connection for each context, and eDeveloper tries to reuse DBMS connections when possible. Access to external devices like printers and files are managed by the Operating System.

Using External Programs (UDF/UDP)

External programs such as UDF/UDP, should be thread-safe. Remember that they can be run by several threads concurrently.

CTX Functions

Use the functions below for querying the eDeveloper Context Manager:

- CtxNum (number of active contexts)
- CtxProg (number of the top level program of the context)
- CtxStat (context status)
- CtxSize (context memory size)
- CtxLstUse (number of seconds since the last activation of a context)

Building a Batch Task 19

Batch tasks are normally executed without interactive user contact. This means that the task logic is executed automatically. This chapter discusses the different uses of a batch task and the modifications you can make to certain batch task settings and the corresponding results of those changes.

This chapter covers the topics listed below:

- Creating a Simple Batch Program
- Manipulating the Execution of a Batch Task
- Batch Task Event Handling
- Defining an Endless Executed Program
- Defining a Chunk of Records from a Data Table
- Creating an Import/Export Program

Creating a Simple Batch Program

A Batch program is usually used when you want to perform automatic procedures, such as generating reports and global table updates.

Batch tasks are executed without interactive user contact and the task logic is executed automatically. The task logic is conditioned according to the expressions defined in the different task operations and properties.

The Record Main of a batch task cannot hold the task logic, except for the Link condition which determines whether the link will be executed or not.

The Record Main level is where you should define the task's dataview, including the range of the dataview and, if you are in Create mode, the initialization of variables.

To prevent a user interfering with the process:

1. In the **Program** repository zoom from the relevant program to open the **Task Execution** repository.
2. Press CTRL+P to open the **Task Properties** dialog box. Click the **Properties** tab and set the **Allow Events** property to **No**. Click **OK**.
3. From the **Task** menu choose **Task Control** to open the **Task Control** dialog box. Click the **Behavior** tab and set the **Record event internal** property to **zero**.

Manipulating the Execution of a Batch Task

Batch tasks usually display a pop-up window at the beginning of the execution, where the user should confirm the execution of the current process. The pop-up window is enabled by the 'Allow Options' property in the Task Control dialog box.

Batch tasks can be terminated in one of the following ways:

- Loops on all the records of the dataview and then ends.
- The End Task condition is evaluated to True.
- The user presses the **ESC** key.
- Defining Events that will eventually terminate a task execution.

This section includes the topics listed below:

- Using the Confirm Execution Pop-Up Window
- End Task Condition
- Allow Events

Using the Confirm Execution Pop-Up Window

Before every batch task you are asked to confirm the execution of that process. The pop-up window is displayed according to the value set in the Task Control's **Allow Options** property.

1. From the **Program** repository zoom from the relevant program to open the **Task Execution** repository.
2. From the **Task** menu choose **Task Control** to open the **Task Control** dialog box.
3. Set the **Allow Options** property.

The **Allow Options** property also lists the following operations, which can be defined in the pre-execution phase:

- Range of Records
- View by Key
- Sort Records
- Redirect Files

Each of these operations can be disabled in the **Task Control**, or you can set the **Allow Options** property to **NO** and these options will not be allowed automatically.

End Task Condition

The End Task Condition is evaluated on every loop the Batch task performs on the Record level.

The End Task Condition is unnecessary when a Main table is defined and the procedure executed should be performed on all the records in the dataview.

Where no Main table is selected, the evaluation of the End Task Condition to True prevents the task from entering an endless loop.

1. From the **Program** repository zoom from the relevant program to open the **Task Execution** repository.
2. Press CTRL+P to open the **Control Properties** dialog box.
3. Set the **End Task Condition** property to **Yes**.
4. Set the **Evaluate Condition** property to one of the following:
   - Before Entering Record (default)
   - After Updating Record
   - Immediately when condition is changed

Allow Events

Batch tasks automatically perform the logic defined in them. Usually the user does not interfere in the task process, which continues until it is terminated automatically.

Although the process is executed automatically, you can also abort continuation of the process as follows:

1. From the **Program** repository zoom from the relevant program to open the **Task Execution** repository.
2. Press CTRL+P to open the **Control Properties** dialog box.
3. Set the **Allow Events** property to **Yes**. The user will be able to press the **Esc** key at any point and halt the process.

When **Allow Events** is evaluated to **False**, none of the triggered events, Internal, System, User, Timer, Expression, will be polled from the stack of the events and none of the event handlers will be executed.

Batch Task Event Handling

Batch task Logic can be defined in the Task Prefix, Task Suffix, Record Suffix and the different Event Handlers.

All the Event types can be triggered in a batch task. There is a polling mechanism for the triggered events which is dependent on the Task and Environment parameters.

The parameters are as follows:

- Batch event interval
- Record Event Interval
- Allow Events

The events can be triggered by the user or by the Raise Event operation, both of which manipulate the logic.

This section includes the topics listed below:

- Defining the Event Handler
- Defining the Handler

Defining the Event Handler

The process of defining an event for either a batch or an online task is the same.

Define the Event trigger as follows:

1. In the **Task** repository create a new line, either after the **Task** or **Record Suffix**, and select **Handler** as the **Level**.
2. From the **Event** column zoom to open the **Event** dialog box.
3. Choose one of these **Event Types**:
   - System - a keyboard stroke
   - Internal - an eDeveloper action from the Actions list
   - User - a logic trigger created by user needs

- Timer - a timer interval trigger
- Expression - evaluation of an expression to True
- Error - an error retrieved from the Database

Defining the Handler

You can fine-tune a Handler in the following ways:

- Specify the Handler to a control
- Define the handler's scope
- Determine if the event is propagated up in the runtime tree
- Condition caching of the trigger.

The Handler includes the different operations defined for the event and manipulates the program logic and results.

You can trigger events in one of the following ways:

- User keyboard strokes, which trigger **System** type events (such as **CTRL+Y** or **F4**).
- The **Raise Event** operation, which can trigger **System**, **Internal** and **User** type events. If required, the **Raise Event** operation enables synchronization by setting the operation property **Wait**, to **Yes**. Another alternative to using the **Raise Event** operation is passing arguments to the **Event Handler**, which caches the event.

Allow Events

Define this property in the **Task Properties** dialog box.

- It is a **Yes**, **No** or **Expression** property that is evaluated once, when the task is opened.
- When you evaluate this property to **False**, none of the triggered events, Internal, System, User, Timer, Expression, will be polled from the stack of events and none of the event handlers will be executed.

Batch Event Interval

To define this property:

1. From the **Settings** menu click **Environment**.
2. Click the **System** tab.

The interval is in milliseconds and determines the interval of polling events which were triggered from the stack of events.

Record Event Interval

You define this property in the **Task Control** dialog box.

The interval is in milliseconds and determines the interval of polling events triggered from the stack of events.

When the **Record Event Interval** and the **Batch Event Interval** properties both have a value greater than zero, the addition of both values is the polling interval of events from the stack of events.

This property can be set either to a simple number or to an evaluation of an expression. In both cases, it is evaluated once, when the task is opened.

Defining an Endless Executed Program

Batch tasks can be terminated in one of the following ways:

- Looping on all the records of the dataview and then stopping.
- Evaluating the End task condition to **True**.
- When the user presses **ESC**.
- Defining Events that eventually terminate task execution.

You should not use any of the above if you want an endlessly looping Batch task. If you want an endlessly looping batch task ensure that:

- No **Main table** is defined for the **Batch task** in the **Task Properties** dialog box.

- The **End task** condition property is evaluated to **False** and that it never terminates the task execution.
- The **Allow Events** property should always be evaluated to **False**, which prevents the user triggering an event that closes the executed task.

Endless looping is also relevant for ISAM databases, when updating one of the segments of the Fetching index. Increasing that segment will always cause looping since there is always a following record that is fetched.

This section includes the topics listed below:

- Batch Task without a Main Table
- End Task Condition
- Allow Events
- Updating the Fetching Index

Batch Task without a Main Table

This is a batch program with no **Main table** selected in the **Task Properties** dialog box.

A virtual record is always created and as there is no range of fetched records, it loops endlessly.

End Task Condition

On every loop the Batch task performs on the Record level, the **End task Condition** is evaluated.

- If you define a Main table and the procedure executed should be performed on all the records in the dataview, the End task condition is unnecessary.
- If you do not define a Main table, evaluation of the End Task Condition to True will prevent the task from entering an endless loop.

To create an endless looping task, this property should always be evaluated to **False**.

Allow Events

- If you evaluate this property to True, the user will be able to press Esc at any point and halt the process.
- If you evaluate this property to False, none of the triggered events, Internal, System, User, Timer, Expression, will be polled from the stack of events and none of the event handlers will be executed.

Updating the Fetching Index

This is only relevant for ISAM databases.

When you define a Main table for a program and fetch the dataview, there is an absolute number of records. This prevents the task from entering an endless loop where the records of that Main table are processed.

If one of the segments of the fetching index is increased, so that it is re-fetched from the database with the new greater value, it then enters an endless loop.

Defining a Chunk of Records from a Data Table

To automatically delete a chunk of records you should define a batch program according to one of the following options:

- Define a Batch program with **Init.** status = **Delete**.
- In the **Task Control** dialog box setting the **Force record delete** property to **True**.
- Defining a **Batch** program with a **Direct SQL** statement where a chunk of records are dropped from the database.

The definition of the dataview to be deleted can be one of the following:

- Range FROM/TO of records in the Record Main.
- Range expression in the task's Range/Locate option.
- Defining a SQL Where clause in the task's Range/Locate option.

This section includes the topics listed below:

- Init. Status = Delete
- Force Record Delete = True
- Direct SQL Statements

Init. Status = Delete

If you set **Init. Status=Delete**, The fetched records are not processed and are removed from the database. The records deleted are only the ones which are fetched from the Main Table. Linked records are not deleted.

Force Record Delete = True

To allow records to be conditionally deleted:

1. From the **Task** menu choose **Task Control**.
2. Click the **Behavior** tab and set the **Force record delete** property to be evaluated to True.

   This is a **Yes**, **No** or **Expression** property and is evaluated for each record fetched and processed. This is similar to **Init. status=Delete**, in that the records deleted are only the ones which are fetched from the main table and linked records won't be deleted.

The difference between deleting records when **Init. status=Delete** and this method is the process that each fetched record passes. **Status=Delete** removes the fetched records from the database without first updating any required information. This means that for each record in the dataview, there is one record loop (Record Prefix/Suffix). **Force record delete** loops twice for each record. The first loop is for modifications, if there are any. The second is the actual delete.

Status=Delete is generally faster, but in order to delete a chunk of records you need to have an index that can fetch the requested chunk of records efficiently.

However, if you want to process a chunk of records and only want to delete a selection after the process, then **Force record delete** is the better.

Note: This property can also be defined for Online tasks.

Direct SQL Statements

A Batch task defined without a Main table can use Direct SQL.

The Direct SQL statement should be according to the selected database. Terminology may be differ from one database to another.

The SQL statement can also have a **Where** section where the range of deleted records is set. This enables you to delete a chunk of records. It can be defined according to an index or not, but this is something that the database determines.

Creating an Import/Export Program

You can use an Import/Export program to import data from a text file into a Table in your application, or to export data records from a Table to a text file.

To create an Import/Export program:

1. In the **Table** repository select the Table for which you want to create an Import/Export program.
2. From the **Options** menu choose Generate Program or press **CTRL+G**.
3. Set the **Mode** property to **Generate**.
4. From the **Option** list select either Import or Export depending on the type of program you wish to create.
   If you select **Export**, text output will be generated from the data currently held in the Table.
   If you select **Import**, data from a text file will be read into a DB Table.
5. From the **Column** field zoom to choose the column to be selected by the Import/Export program:
   Note: Only the selected columns will be processed by the program and the other

columns will be skipped. By default eDeveloper selects all columns.

6. In the **Program name** field enter a name for the program as you wish it to appear in the Program repository.
7. In the **File name** field enter a name for the text file and zoom to set its location. You can also use a Logical Name to specify the file location.

Defining an I/O Form's Style

To define a Style for the Input/Output form:

1. In the **Program Generator** dialog box, click the **Style** tab.
2. In the **Display** field select a display type from the **Display** list. Export/Import programs usually use **Line** format.
3. In the **Style** field select the desired Style type.
4. From the **Use Model** field zoom to open the **Model List** and choose the Model of a text-based form that the program should use:
5. Click **OK** to generate the new program, which will be entered into the **Program** repository.

Integrating With the J2EE Environment

20

eDeveloper allows you to generate Enterprise Java Beans (EJBs). Clients operating within a J2EE environment are able to view and activate programs as EJB methods.

This chapter includes the topics listed below:

- J2EE Server Installation
- Enabling eDeveloper with EJB Support
- Generating EJBs Using eDeveloper
- EJB Deployment Using eDeveloper
- Setting Up a Java Environment
- Learning the Content of a Java Class
- Creating a New Instance of a Java Class
- Reading Values of Java Variables
- Calling a Java Method

J2EE Server Installation

The environment settings described in this section enable the eDeveloper Component Builder to generate EJBs. If the test clients are activated on another host, such as a deployment site, they will also require these variables.

For ease of use each EJB's .jar is copied into the different deployment folders, such as Weblogic and Websphere. In order to deploy or test an EJB on any host, its folder must be copied as is to the target host. At deployment the standard settings of each J2EE server are satisfactory.

This topic describes how to install these J2EE servers:

- WebSphere
- WebLogic 6
- WebLogic 5.1
- Sun Ref
- jBoss 2.4.4

WebSphere

Setup.exe - For silent mode use: **setup path\setup.iss -s**

1. Install WebSphere, for example into **c:\was\appserver**.
2. Set the **Environment variables**
   From the **Control Panel** choose **System** and then **Environment**.

   MG_J2EE_HOME **c:\WebSphere\appserver**
   This is set automatically depending on the installation.

   MG_JAVA_HOME **%MG_J2EE_HOME%\Java**

BEA WebLogic 6

1. Install WebLogic 6. For example: into **c:\bea**.
2. Set the Environment variables:
   From the **System** menu choose **Environment**.
3. Then set the following:

MG_J2EE_HOME	**c:\bea\wlserver6.1** Depending on the installation
MG_JAVA_HOME	**c:\bea\jdk131**
MG_CLASSPATH	Prefix with: **%MG_JAVA_HOME%**\jre\lib\ext**mgejbgnrc.jar;%MG_J2EE_HOME%**\lib**weblogic.jar;**

BEA WebLogic 5.1

1. Download & Install **JAVA** 1.3.0 or higher, such as from **Sun Microsystems**.
   For example: into **c:\jdk1.3**.
2. Install **WebLogic 5.1**.
   For example: into **c:\weblogic**.

3. Set the Environment variables:
   From the **System** menu choose **Environment**.
   This also applies to c:\weblogic**SetEnv.cmd**.

MG_J2EE_HOME	**c:\weblogic** Depending on the installation.
MG_JAVA_HOME	**c:\jdk1.3** Depending on the installation.
MG_CLASSPATH	**%MG_JAVA_HOME%**\jre\lib**ext\mgejbgnrc.jar;%MG_J2EE_HOME%**\lib**weblogicaux.jar;**

Sun Reference Implementation

1. Download & Install **JAVA** version 1.3.0 or higher, such as from **Sun Microsystems**, for example into: **c:\jdk1.3**.
2. Download & Install **J2EE** version 1.2.1 or higher, for example into **c:\j2sdkee1.3**.
3. Set the Environment variables:
   From the **System** menu choose **Environment**.

MG_J2EE_HOME	**c:\j2sdkee1.3** - Depending on the installation.
MG_JAVA_HOME	**c:\jdk1.3** - Depending on the installation.

jBoss 2.4.4

1. Download & Install **Java** version 1.3.0 or higher, such as from **Sun Microsystems**.
   For example: into **c:\jdk1.3**.
   Set this to start the jBoss server: PATH= c\jdk1.3\bin;%PATH%
2. Download & Install **jBoss 2.4.4** from the jBoss Web-site.
   For example: into **c:\jboss-2.4.4**.

3. Set the Environment variables:
   From the **System** menu choose **Environment**.

MG_J2EE_HOME	**c:\jBoss-2.4.4**
MG_JAVA_HOME	**c:\jdk1.3**
MG_CLASSPATH	Prefix with %MG_J2EE_HOME%\lib\ext\mgejbgnrc.jar%MG_J2EE_HOME%\lib\ext\jboss-j2ee.jar;

If you have installed the enterprise server after installing eDeveloper, you must copy the MGEJBGNRC.jar file from the Support directory under the eDeveloper root and copy the file to the %MG_J2EE_HOME%\Lib\Ext directory.

Enabling eDeveloper with EJB Support

To be able to generate EJBs using eDeveloper, you have to set certain properties within eDeveloper. The following steps describe the changes you should make.

1. In the **Script** directory edit the **Mgreq.ini** file by removing the semi-colon in front of `MGSRVRO5`. If you enter a gateway value of 5, the server will only be dedicated for J2EE requests. If you enter a gateway value of 1 and the eDeveloper engine is set for background mode the server is enabled for IAS, J2EE and Web Service requests.
2. Open eDeveloper (V9.2 or later) as an **Online AppServer**.
3. Once you have set eDeveloper to work with EJB generation, you can begin to create your EJBs.

When you load eDeveloper as a **background** AppServer, it is **not** mandatory to set **Gateway=5**. You can leave **Gateway=1**, as with the broker, and uncomment the following line (add a semi-colon (;) before the line) in the **mgreq.ini** file:

```
[MAGIC_MESSAGING_GATEWAYS]
MGSRVR05= , , , ,MaxThreads=10%
```

In this case 10% of the license is allocated to J2EE, while the remaining 90% is free for internal messaging. This means that the appserver can simultaneously serve both types of requests.

Generating EJBs Using eDeveloper

You can create EJBs using eDeveloper's Component Builder utility. The following procedure describes the process:

1. In your eDeveloper application, from the **Options** menu choose **Component Builder**.
2. Choose a name for the component, and then in the **Class** column select **EJB** from the **Class List.**
3. Click **Add Items** and select the programs you want to use as the EJB's methods.
4. Zoom from the **Arguments/Returned Value** column to change the Java type with which each parameter or returned value will be defined in the EJB.
5. From the **Component Builder** menu choose **Build JAR file** to generate the EJB.

EJB Deployment Using eDeveloper

eDeveloper's Component Builder generates the **EJB .jar** files. The files are saved in an EJB folder in the main eDeveloper directory.

In order to facilitate testing, the Component Builder generates a **test application** called **<Bean Name>_test_app.ear**. This application is not an integral part of the generated EJB, but rather wraps the application up with a simple client that you can activate later, and modify if required, in order to test the EJB.

The following are the steps you need to take to be able to deploy an EJB that you have generated using eDeveloper:

- Setting Up URL Resources for a J2EE Server

- Starting the J2EE Server
- Starting the Deployment Tool
- Deploying the EJB
- Running the Client
- Stopping the J2EE Server
- Advanced Configuration of eDeveloper AppServers

Each topic is sub-divided according to these four J2EE server types:

- WebSphere
- WebLogic 5.1
- WebLogic 6
- Sun Ref Implementation

Setting Up URL Resources for a J2EE Server

WebSphere

1. From **Start** choose **Programs**, then choose **IBM WebSphere**. Then choose **Enterpise Server V4.0** and then the **Administrator's Console.**
2. In the left-hand frame select the following:
   WebSphere Administrative Domain -> Resources -> URL Providers -> Default URL Provider -> URLS
3. In the right-hand frame click the **New** button and enter the following data:
   ```
   Name = MagicURL
   JNDI Name = url/MagicURL
   Spec = http://localhost:1500
   ```
4. To save your changes click **OK** and then click the **Save** link.

WebLogic 5.1

Open: %MG_J2EE_HOME%**weblogic.properties** and add **URLResource(s)**.

For example:
weblogic.httpd.URLResource.url.MagicURL = http://localhost:1500

WebLogic 6

1. **Open:** %MG_J2EE_HOME%\config\mydomain**config.xml** and add one or more **URL Resources**.
2. For example:
   <WebServer**URLResource="url.MagicURL=http://localhost:1500 url.MagicURL2=http://localhost:1501"**DefaultWebApp = "DefaultWebApp_myserver"
   LogFileName = "./config/mydomain/logs/access.log"LoggingEnabled="true"
   Name="myserver"/>

Sun Ref

You do not need to set up URL resources for the **Sun Ref** server.

Starting the J2EE Server

WebSphere

Start > **Programs** > **IBM WebSphere** > First Steps > Start **Enterpise Server**.

WebLogic 6

Start **Programs** > **BEA WebLogic** > **WebLogic Server 6.0** > Start **Default Server**.

WebLogic 5.1

cd %MG_J2EE_HOME%
SetEnv.cmd
StartWebLogic.cmd

Sun Ref

cd %MG_J2EE_HOME%\bin
j2ee - verbose

jBoss 2.4.4

%MG_J2EE_HOME%\bin
run.bat

Starting the Deployment Tool

WebSphere

Start > **Programs** > **IBM WebSphere** > **Enterpise Server V4.0** > **Application Assembly Tool (AAT)**.

WebLogic 5.1

Start > **Programs** > **WebLogic 5.1.0** > **Ejb Deployer**.

WebLogic 6

1. Start **Programs** > **BEA** > **WebLogic** > **WebLogic Server 6.0** > **Start Default Console**.
2. In the left-hand panel**:** mydomain > **Deployments > EJB**.

Sun Ref

Deploytool

Deploying the EJB

When you deploy the EJB into the J2EE server, you make it available for execution.

The following describes the procedures for the different servers.

WebSphere

In the **Application Assembly Tool (AAT)**:

1. Click **Existing** and then **Browse** to open the following file:
   \ejb\<Bean_Name>\sun\<EJB_Name>_test_app.ear
2. From the **File** menu click **Generate Code for Deployment**.
3. Specify the **Deployed Module Location** as follows:

\ejb\<Bean_Name>\websphere\Deployed_<Bean_Name>_test_app.ear

4. Click the **Generate Now** button.

In the **Admin. Console**:

1. In the left-hand frame select:
   WebSphere Administrative Domain, then **Nodes**, the **<Server Name>** and then select **Enterprise Applications**.
2. Click the **Install** button, then click the **Browse** button and select the file generated by the AAT:
   \ejb\<Bean Name>\websphere**Deployed_<bean name>_test_app.ear**
3. Click **Next,** then click **Next** again and then again click **Next**.
4. Clear the **Redeploy Option** and click **Next**. Then click **Finish**.
5. Save your changes by clicking the **Save** link.
6. In the left-hand frame for **Default server** verify that **Execution State** is **Start** and **Module visibility** is **Application**.
7. Click **OK** and then restart **Websphere 4 server**.

WebLogic 5.1

From the **ejb\<bean_name>\weblogic** directory:

1. Use **WLrebuild5.cmd** once for each EJB, in the development site.
2. From the **Ejb Deployer**
   File > Open > (**<bean name>_deployable.jar**), **Tools > Deploy**

3. Validate the server
   Servers > Add

WebLogic 6

In the Default Console choose to install a new EJB.

1. To install a new EJB browse to **..\ejb\<bean name>\sun\<EJB_name>.jar**.
2. Then click **Upload**.

Sun Ref

1. From the **File** menu choose **Open Application**.
2. From the **Tools** menu choose **Deploy Application** and select the **Return Client Jar** box.

Running the Client

The .cmd file uses the **client JAR** generated in the previous stage.

WebSphere	ejb\<bean_name**>\websphere**\ClientTest**.**cmd
WebLogic 6	ejb\<bean_name**>\weblogic**\ClientTest**6.**cmd
WebLogic 5.1	ejb\<bean_name**>\weblogic**\ClientTest**5.**cmd
Sun Ref	ejb\<bean_name>**\sun**\ClientTest**.**cmd

Stopping the J2EE Server

WebSphere

First Steps > Stop the Enterpise Server.

WebLogic 5.1

You can stop the server from windows services.

WebLogic 6

1. **WebLogic Server 6.0** > **StartDefaultConsole**.
2. In the **Administration Console** domain tree of the left-hand pane, select the server.
3. On the **Monitoring General tab** page, click the **Shutdown this server** link.

Sun Ref

J2ee – **stop**.

Advanced Configuration of eDeveloper AppServers

Where an EJB is configured to use more than one AppServer, such as **MagicAppservers = mgurl1,mgurl2**, then:

1. If all AppServers use the same **Magic.ini** in [MAGIC_COMMS]TCP/IP, define a continuous range (e.g. 1500-1501)
2. Otherwise, for different **Magic.ini** files: in [MAGIC_COMMS]TCP/IP of each Magic.ini, define one port (e.g. 1500).

Setting Up a Java Environment

This topic will show you how to set up the Java environment. This setup is required in order to be able to call Java classes from eDeveloper.

Required Java Software

To run Sun Microsystems Inc.'s Java™ code on your PC or server, you should install the Java Runtime Environment (JRE). As opposed to Java code, which you can write, compile once (to byte-code) and run on almost any platform, the JRE software is operating-system specific.

The Java Software Development Kit (SDK) should be used primarily by Java developers and includes the tools required to compile Java code, several Java utilities, and the JRE.

System-wide Settings

If you install several Java JRE/SDK versions, you should set the JAVA_HOME environment variable. This variable should be set in the main directory of the JRE/SDK.

For example:

JAVA_HOME = C:\Java\j2re141

If you are using the eDeveloper Java functions, you should also add the following entries to the PATH environment settings:

For JRE: %JRE_HOME%\bin\client

For SDK: %SDK_HOME%\jre\lib\client

If you are using eDeveloper on a non-Windows operating system, please follow the instructions in the readme file that accompanies your eDeveloper installation.

Setting the Java Classpath

The Java code that you want to execute can reside on several directories and in several files.

Usually, sveral Java classes are packaged in a JAR (Java archive) file or in a ZIP file.

The Java runtime environment locates the Java classes using the classpath. The Java classpath should include a list of directories and/or JAR/ZIP files.

You can define a system-wide classpath using the environment variable CLASSPATH. The CLASSPATH value should appear as follows:

CLASSPATH = C:\Java_classes\;C:\my_jars\service1.jar

You can also define a specific classpath for the eDeveloper environment by setting the CLASSPATH entry in the MAGIC_JAVA section of the eDeveloper INI file.

The separator between the entries of the CLASSPATH should be a semicolon (;) on a Windows platform and a colon (:) on Unix and iSeries platforms. For more detailed information, refer to to the Java 2 SDK documentation, Tools and Utilities, Setting the Classpath.

Setting JVM Arguments

When running Java code, you may set JVM (Java Virtual Machine) arguments to provide additional setup information for the Java runtime environment.

When running Java code, you may pass this information using the –D flag of the java command.

To pass JVM arguments to the eDeveloper environment, you should add the arguments (prefixed by the –D) to the JVM_ARGS entry in the MAGIC_JAVA section of the eDeveloper INI file.

For example:

```
JVM_ARGS= -Djava.compiler=NONE -Djms.properties=C:\j2sdkee1.3.1\con-
fig\jms_client.properties
```

Learning About the Content of a Java Class

This topic will show you how to learn about the content of a Java class.

What is a Java Class?

In Java, an object has variables and methods. For example, a rectangle is an object that has length and width variables as well as a calculateArea() method. A Java class (rectangle) is a blueprint or template of all objects of a certain type. A Java class includes variable members (fields) and methods (functions that can be accessed using the eDeveloper Java functions (JCreate, JGet, JCall, etc.).

The Java developer can choose to define static variables and/or methods. The static variables are shared by all the Java objects that refer to this class. To access the static variables and methods you should prefix the variable/method with the class name. For example, separator is a static variable of class java.io.File. To access the separator value use: java.io.File.separator (or File.separator).

Using the JExplore Functions

The eDeveloper JExplore function queries a class definition and generates an XML output that describes the structure of the class.

The EJBExplore fucntion has one alpha input parameter. This parameter should be the full name of a Java class, such as “java.lang.StringBuffer”. As the XML output can be quite big (more than the maximum size of an alpha field), it is recommended to define a BLOB variable with the “GUI display” property set to “Rich Edit” and update it with the JExplore function.

Using the Javap Utility

The javap utility (part of the Java 2 SDK) can also be used to query a class definition. The syntax is: javap –s <class name>

For example:

```
javap –s java.lang.StrigBuffer
```

Examples

The following is the output of the function: JExplore(‘java.lang.StringBuffer’) [partial output]

```
<?xml version="1.0" encoding="UTF-8"?>
<ClassMetaData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Classes>
```

```
<Class name="java/lang/StringBuffer" abstract="N" interface="N">
  <Dependencies>
    <Extends>
      <ClassName>java.lang.Object</ClassName>
    </Extends>
    <Implements>
      <ClassName>java.io.Serializable</ClassName>
      <ClassName>java.lang.CharSequence</ClassName>
    </Implements>
  </Dependencies>
  <Structure>
    <Constructors>
      <Constructor idx="1">
        <Signature>(I)V</Signature>
        <Parameters>
          <Parameter idx="1">
            <name></name>
            <type>int</type>
          </Parameter>
        </Parameters>
        <Throws>
        </Throws>
      </Constructor>
      <Constructor idx="2">
        <Signature>(Ljava/lang/String;)V</Signature>
        <Parameters>
          <Parameter idx="1">
            <name></name>
            <type>java.lang.String</type>
          </Parameter>
        </Parameters>
        <Throws>
        </Throws>
      </Constructor>
      <Constructor idx="3">
        <Signature>()V</Signature>
        <Parameters>
        </Parameters>
```

```
      <Throws>
      </Throws>
    </Constructor>
  </Constructors>
  <InstanceMethods>
    <Method idx="1">
      <name>toString</name>
      <Signature>()Ljava/lang/String;</Signature>
      <Parameters>
      </Parameters>
      <Throws>
      </Throws>
    </Method>
    <Method idx="2">
      <name>append</name>
      <Signature>(C)Ljava/lang/StringBuffer;</Signature>
      <Parameters>
        <Parameter idx="1">
          <name></name>
          <type>char</type>
        </Parameter>
      </Parameters>
      <Throws>
      </Throws>
    </Method>
  …
  </InstanceMethods>
  <StaticMethods>
  </StaticMethods>
  <InstanceVariables>
  </InstanceVariables>
  <StaticVariables>
    <Variable idx="1" final="Y">
      <name>serialVersionUID</name>
      <Signature>J</Signature>
    </Variable>
  </StaticVariables>
</Structure>
```

```
    </Class>
  </Classes>
</ClassMetaData>
```

The following is the output of the command: javap -s java.lang.StringBuffer [partial output]:

```
Compiled from: StringBuffer.java.

public final class java/lang/StringBuffer extends java.lang.Object implements ja

va.io.Serializable, java.lang.CharSequence {

  static final long serialVersionUID;
    /*  J  */
  public java/lang/StringBuffer();
    /*  ()V  */
  public java/lang/StringBuffer(int);
    /*  (I)V  */
  public java/lang/StringBuffer(java.lang.String);
    /*  (Ljava/lang/String;)V  */
  public synchronized int length();
    /*  ()I  */
  public synchronized int capacity();
    /*  ()I  */
  …
```

Notes:

You will be able to query only Java classes that can be located using your Java classpath settings. For more information refer to the Setting Up the Java Environment topic.

The schema of the XML generated by the JExplore function can be found in the <Magic Dir>\support directory (JExplore.xsd).

Creating a New Instance of a Java Class

To use a Java class, you should first create an instance of a Java class.

The JCreate function creates an instance of a Java class (a Java object) and returns a handle to this instance. The handle should be stored in a BLOB variable.

In Java you create class objects using constructors. Each class should have at least one constructor. You can differentiate between constructors by their signature.

For example, let's examine the constructors of the java.lang.StringBuffer class.

The output of *javap –s java.lang.StringBuffer* looks as follows:

```
…
 public java.lang.StringBuffer();
    /*   ()V   */
 public java.lang.StringBuffer(int);
    /*   (I)V   */
 public java.lang.StringBuffer(java.lang.String);
    /*   (Ljava/lang/String;)V   */
…
```

The java.lang.StringBuffer has 3 constructors (java.lang.StringBuffer methods):

1. Without any input (signature is "()V")
2. With one input parameter of type int (signature is "(I)V")
3. With one input parameter of type String (signature is "(Ljava/lang/String;)V")

Java constructors do not have a return value, they only instantiate a Java object. The V denotes that these methods do not return a value (void).

The JCreate function has several arguments. The first is the class name and the second is the constructor signature. Additional arguments should be passed depending on the input parameters of the constructor.

For example, to create an empty String Buffer object use:

```
JCreate('java.lang.StringBuffer','()V')
```

To create the same object and initialize it with a String object use:

```
JCreate('java.lang.StringBuffer','(Ljava/lang/
String;)V','My String')
```

If an error occurs during execution of the eDeveloper Java functions, the JExceptionOccured function returns logical true and the JExceptionText('False'LOG) returns the Java exception short description.

If you set a wrong signature, eDeveloper will not be able to locate the correct method and will display the error:

```
java.lang.NoSuchMethodError: <init>
```

If you do not supply the correct number of input variables, the following exception is returned:

```
java.lang.Exception: Incompatible signature and number of
arguments!
```

It is recommended to set the GUI display property of the BLOB variable that stores the reference to the Java object to "Rich Edit". If you display this variable on a GUI form, you should see the Java class and an instance id (after updating the variable with JCreate output).

The following is a description of signature terminology (as copied from the Java JNI documentation).

Java VM Type Signatures

Signature	Java Programming Language Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L fully-qualified-class;	fully-qualified-class
[*type*	*type*`[]`
(arg-types) ret-type	method type

For example, the `Prompt.getLine method has the signature:`

```
(Ljava/lang/String;)Ljava/lang/String;
```

`Prompt.getLine` takes one parameter, a Java String object, and the method type is also `String.`

The `Callbacks.main method has the signature:`

```
([Ljava/lang/String;)V
```

The signature indicates that the `Callbacks.main` method takes one parameter, a Java `String` object, and the method type is void.

Array types are indicated by a leading square bracket ([) followed by the type of the array elements.

Notes:

You will be able to create instances only for Java classes that can be located using your Java classpath settings. For more information refer to the Setting Up a Java Environment topic.

eDeveloper performs several automatic conversions. For example: Alpha strings are converted to Java String objects, and Numeric fields are automatically converted to the appropriate Java primitive type (such as int, long, double TBC).

Reading Values of Java Variables

This topic will show you how to read values of static and non-static Java variables.

Reading Values of Java Object Variable Members

In Java, a class can include the definition of variable members. Each Java object has its own variable members. It is also possible to define variables that are shared between all objects of a specific Java class. These shared members are called static variable members.

You can read the value of a variable member (non-static) using the JGet function.

To read a variable member value, you should have a reference to a Java object and you should decide which variable you want to call and find its signature.

For example, let's examine non-static variable members of the class java.util.Vector (a class for storing vectors of objects).

The output of *javap –s java.util.Vector* looks as follows:

```
…
  protected int elementCount;
    /*  I  */
…
  public synchronized boolean add(java.lang.Object);
```

```
        /*   (Ljava/lang/Object;)Z   */
    …
```

The elementCount member is a member variable of type int and the method add() adds a Java object to the Vector.

The JGet function has several arguments. The first is a reference to a Java object for which you would like to read a variable value. The second one is the variable name and the third is the variable signature.

To store the value of the variable, you should update a virtual with the JGet expression. The type of the virtual depends on the Java return value from the method. If the function returns a Java object, you should use a virtual of type BLOB.

As an example we will create a reference to a Vector Java object, add a String object to the vector and read the number of elements in the vector (elementCount variable member).

1. Create a new online program and define the following virtuals:
   - Name = 'J_Vector object', type = BLOB
   - Name = 'J_String object', type = BLOB
   - Name = 'No of elements', type = Numeric, picture = '5'
   - Name = 'Submit button' (for a push button)
2. Init the 'J_Vector object' virtual with the expression:

   ```
   JCreate ('java.util.Vector','()V')
   ```

3. Init the 'J_String object' virtual with the expression:

   ```
   JCreate ('java.lang.String','(Ljava/lang/String;)V','ABC-
   DEFG')
   ```

4. Define a handler for the button that will execute the following:

   a. Evaluate the expression: JCall (A,'addElement',' (Ljava/lang/Object;)V',B)

   b. Update the 'No of elements' variable with JGet (A,'elementCount','I')

Explanation:

1. The ‘J_Vector object’ virtual is initialized with an empty Java Vector object.
2. The ‘J_String object’ virtual is initialized with a Java String object with the value ‘ABCDEFG’ .
3. The ‘No of elements’ virtual is updated with the number of elements in the Vector object (value = 1).

Like with the JCreate function, if an error occurs during execution of the eDeveloper Java functions, the JExceptionOccured function returns logical true and the JExceptionText(‘False’LOG) returns the Java exception short description.

If you set a wrong signature, eDeveloper will not be able to locate the correct method and will display the error:

```
java.lang.NoSuchMethodError: <method name>
```

If you do not supply the correct number of input variables the following exception is returned:

```
java.lang.Exception: Incompatible signature and number of
arguments!
```

Reading Values of Java Class Variable Members (static variables)

Reading values of a static variable member is similar to reading values of non-static variable members.

You should use the JGetStatic function (instead of JGet).

For example, let’s examine static variable members of the java.lang.Integer class (class for storing numeric integer values).

The output of *javap –s java.lang.Integer* looks as follows:

```
…
   public static final int MAX_VALUE;
      /*  I  */
…
```

MAX_VALUE is a static member of the class java.lang.Integer.

The JGetStatic function has several arguments. The first is a concatenation of the class and variable name (separated by a dot character). The second one is the variable signature.

For example: JGetStatic('java.lang.Integer.MAX_VALUE','I')

Notes:

Java does not permit access to virtual members that are defined as private members.

eDeveloper performs several automatic conversions. For example: Alpha strings are converted to Java String objects, Numeric fields are automatically converted to the appropriate Java primitive type (such as int, long, double TBC).

Calling a Java Method

This topic explains how to call static and non-static Java methods.

Calling Java Object Methods (non-static)

Java code is executed by calling Java class methods of a Java object. Java class methods are like eDeveloper functions. They can have one or more input parameters and a single return value.

The Java programmer can choose to define static methods, which are shared by all Java objects of the same class.

To call Java methods, first you create a Java object (or retrieve a reference to an existing one). Then, you decide which method you want to call and find its signature.

For example, let's examine several methods of the java.lang.StringBuffer class. The output of *javap –s java.lang.StringBuffer* looks like the following:

```
…
   public synchronized int length();
      /*   ()I   */
...
   public synchronized java.lang.StringBuffer insert(int, java.lang.String);
      /*   (ILjava/lang/String;)Ljava/lang/StringBuffer;   */
…
```

The method length() does not have input parameters and returns a Java int primitive type.

The method insert() with the signature: "(ILjava/lang/String;)Ljava/lang/StringBuffer;" has two input parameters, an int and a String Java object and returns a StringBuffer Java object.

The JCall function has several arguments. The first is a reference to a Java object on which you want to execute the method. The second one is the method name. The third is the method signature. Additional arguments should be passed depending on the input parameters of the constructor.

To store the result of the method call, you should update a virtual with the JCall expression. The type of the virtual depends on the Java return value from the method. If the methods return a Java object, you should use a virtual of type BLOB.

As an example we will create a reference to a StringBuffer Java object (initialized with the string 'ABCDEFG'), query its length, and call the insert method in order to get a new StringBuffer object with the value 'ABC123DEFG' .

1. Create a new online program and define the following virtuals:
   - Name = 'StringBuffer object', type = BLOB
   - Name = 'String length', type = Numeric, picture = '5'

- Name = ‘Another StringBuffer’, type = BLOB
- Name = ‘Submit button’ (for a push button)

2. Init the virtual ‘StringBuffer object’ with the expression:

```
JCreate ('java.lang.StringBuffer','(Ljava/lang/String;)V','ABCDEFG')
```

3. Define a handler on the button that will execute.

   a. Update the ‘String length’ virtual with JCall(A,’length’,’()I’)

   b. Update the ‘Another StringBuffer’ virtual with JCall (C,'insert',' (ILjava/lang/String;)Ljava/lang/StringBuffer;',3,'123')

   c. Using a Verify operation, display the value of the ‘Another StringBuffer’ virtual.

Explanation:

1. The ‘StringBuffer object’ virtual is initialized with a StringBuffer object that has the string ‘ABCDEFG’
2. The length of the alpha string is retrieved (value is 7)
3. A new StringBuffer is created by inserting the alpha string ‘123’ from position 3 of the String Buffer ‘ABCDEFG’. The result should be ‘ABC123DEFG’

Like with the JCreate function, if an error occurs during execution of the eDeveloper Java functions, the function JExceptionOccured returns logical true and the JExceptionText(‘False’LOG) returns the Java exception short description.

If you set a wrong signature, eDeveloper will not be able to locate the correct method and will display the error:

```
java.lang.NoSuchMethodError: <method name>
```

If you do not supply the correct number of input variables the following exception is returned:

```
java.lang.Exception: Incompatible signature and number of
arguments!
```

Calling Java Class Methods (static)

Calling static methods is similar to non-static methods. However, you should use the JCallStatic function instead of the JCall function.

The JCallStatic function has several arguments. The first is a concatenation of the class name and the static method name (separated by a dot character). The second one is the static method signature. Additional arguments should be passed depending on the input parameters of the constructor.

For example, the static method toHexString of class java.lang.Integer has the signature:

```
public static java.lang.String toHexString(int);
  /*   (I)Ljava/lang/String;   */
```

We will call this method from eDeveloper by evaluating the expression:

```
JCallStatic('java.lang.Integer.toHexString',' (I)Ljava/lang/String;',16)
```

(the result should be the alpha '10')

Note:

eDeveloper performs several automatic conversions. For example: Alpha strings are converted to Java String objects, and Numeric fields are automatically converted to the appropriate Java primitive type (such as int and long).

Using COM Support

21

eDeveloper lets you easily handle any registered COM object in your eDeveloper application. You may choose to use and handle an ActiveX control as an additional control in the GUI Display form or as a general OLE object to be handled by OLE automation.

This chapter covers the topics listed below:

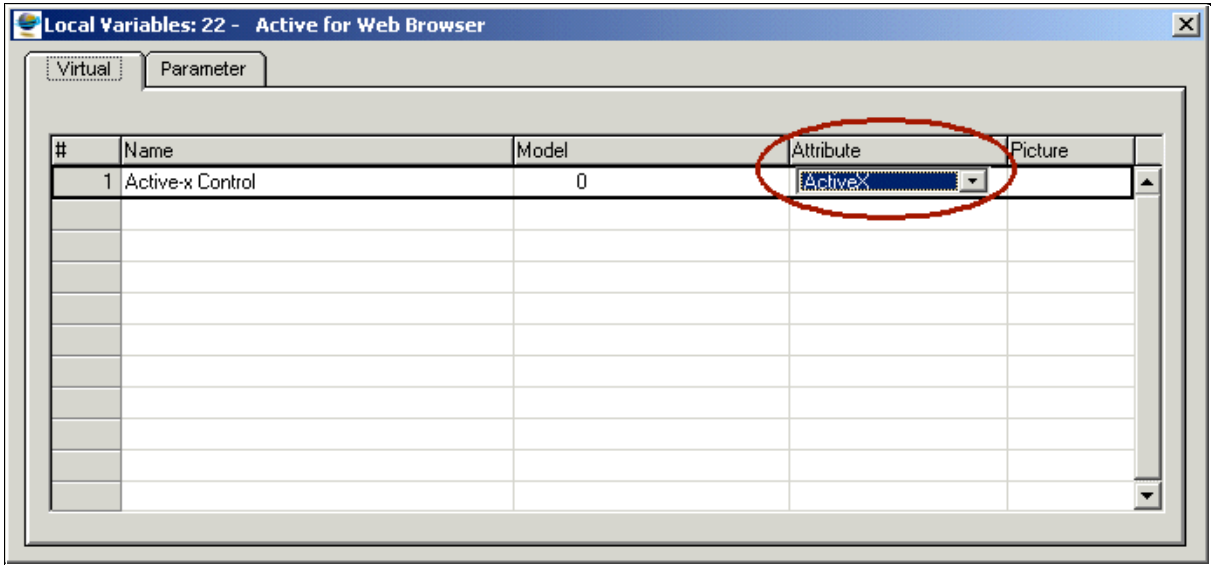
- Defining an ActiveX Control
- Setting a COM Object Property
- Calling a COM Object Method

Defining an ActiveX Control

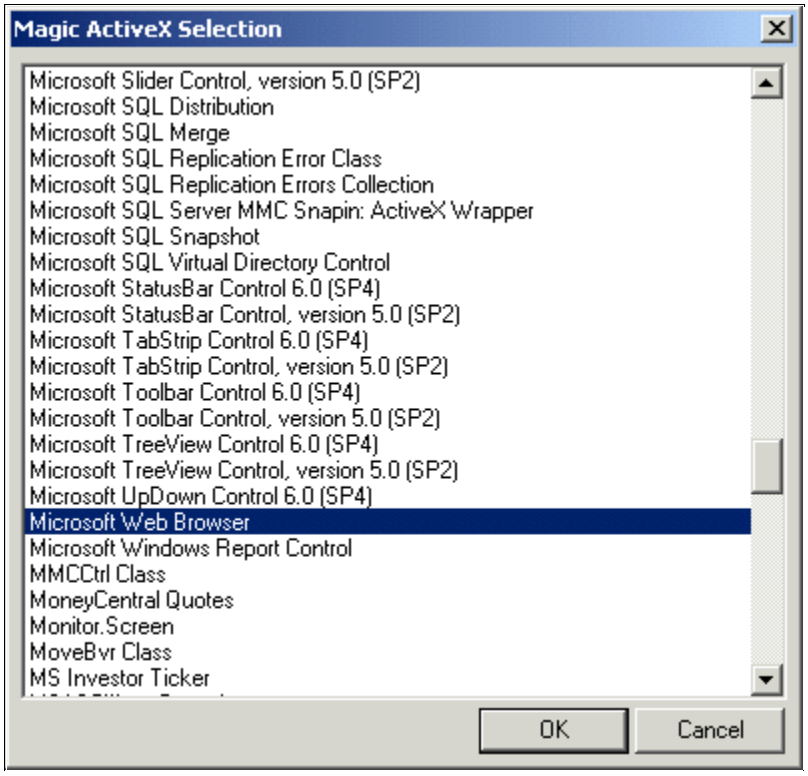
This topic will show you how to define an ActiveX control for the Web browser and progress bar, how to add the ActiveX control to the screen and how to activate it.

Creating an ActiveX control for the Web browser

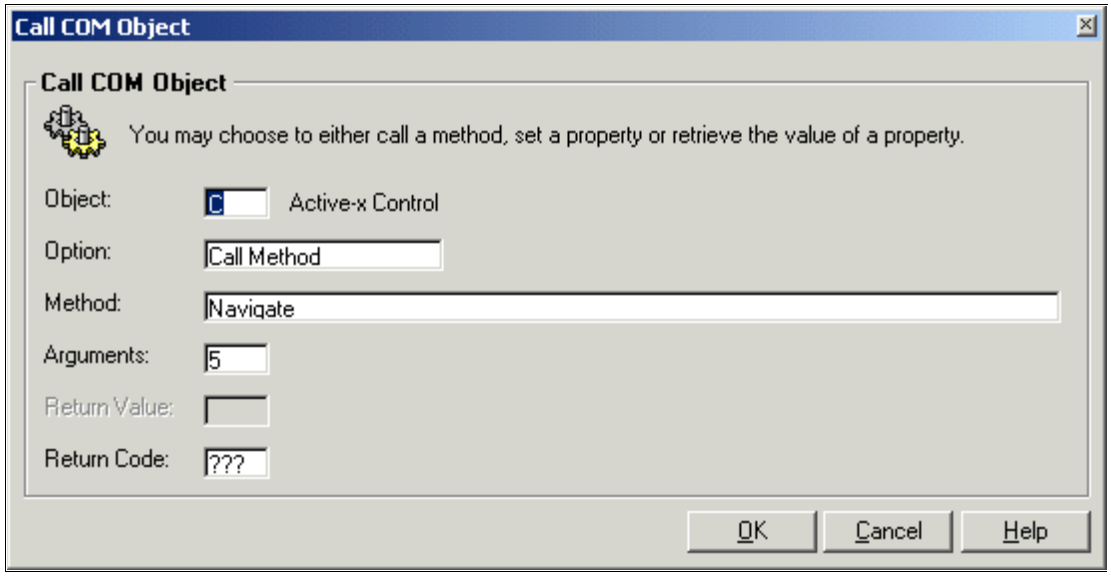
1. Create a virtual variable and set the attribute to **ActiveX**.



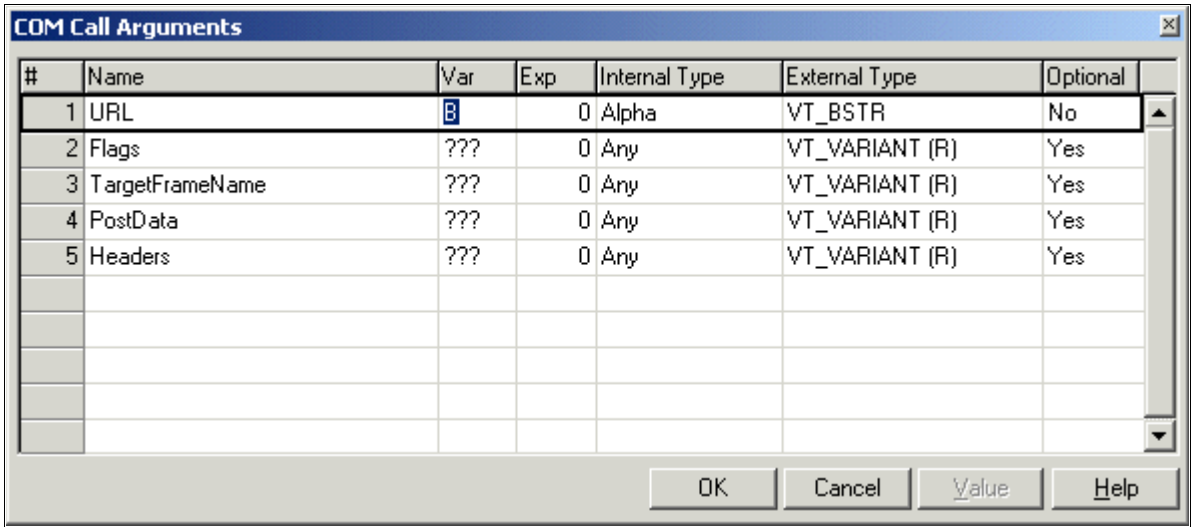
2. From the **Local Variable Properties** sheet, zoom from the **Object name** property to the **ActiveX Selection** list.



3. Select the **Microsoft Web Browser** option.
4. Create an alpha virtual variable and define the URL Init value as 'www.magicsoftware.com'
5. Open the **Form Editor** and drag the ActiveX control and the URL control from the Variable palette onto the form.
6. Create an internal handler with a Zoom event for the URL control.
7. In the handler, create a **Call COM** operation and zoom to the **Call COM Object** dialog box.
8. In the **Object** field, select the ActiveX control.



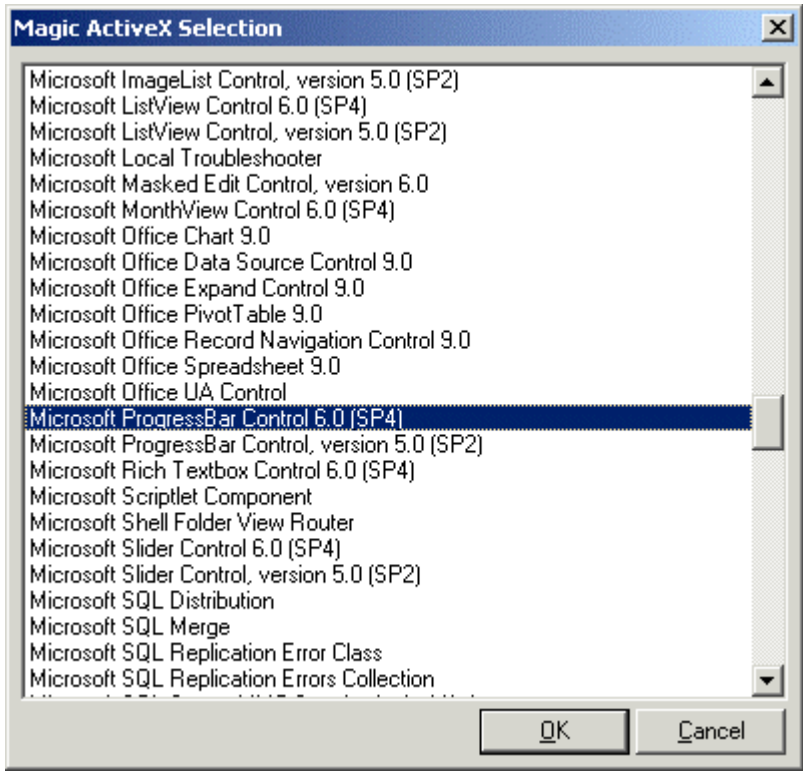
9. In the **Option** field, select **Call Method**.
10. In the **Method** field, select **Navigate**.
11. From the **Arguments** field, zoom to the **COM Call Arguments** dialog box.



12. From the URL entry, zoom from the **Var** column.
13. Select the URL field.
14. Run the program and zoom from the URL field.

Creating an ActiveX control for the Progress bar

1. Create a virtual field and set the attribute to **ActiveX**.
2. From the **Local Variable Properties** sheet, zoom from the **Object name** property to the **ActiveX Selection** list.
3. Select the **Microsoft ProgressBar Control, version 6.0** option.

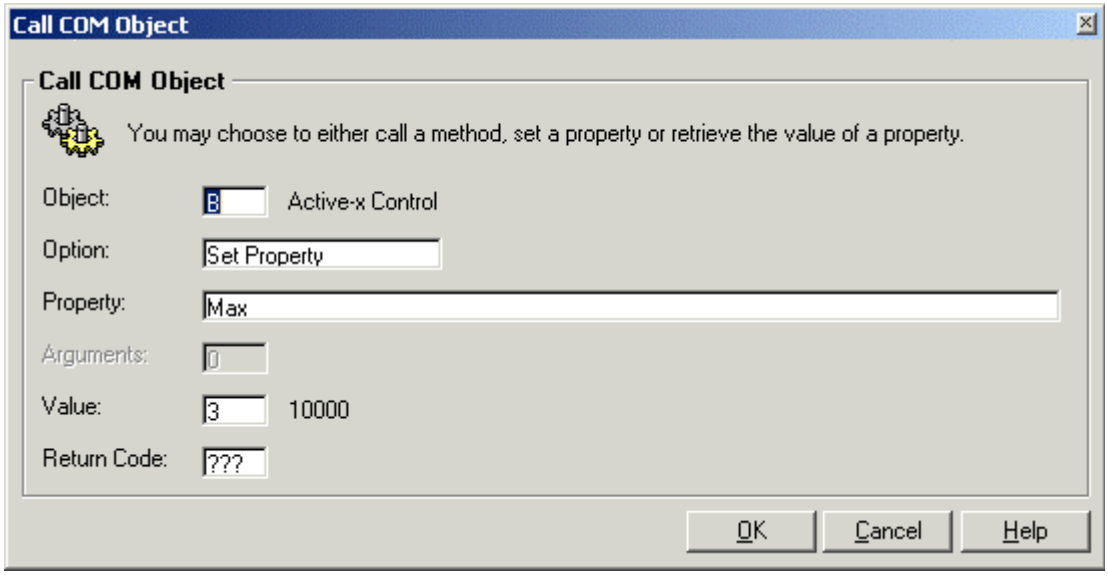


4. In the **Task Properties** dialog box, change the task to a **Batch** task and set the End Task condition to: **Counter(0)>10000**
5. In the **Behavior** tab of the **Task Control** dialog box, set the **Open task window** parameter to **Yes**.
6. In the **Form Editor**, drag the ActiveX control from the Variable palette onto the form.
7. In the **Task Prefix**, create a **Call COM** operation and zoom to the **Call COM Object** dialog box.
8. In the **Object** field, select the ActiveX control.

9. In the **Option** field, select **Set Property**.
10. In the **Property** field, select **Max**.
11. Zoom from the **Value** field to the **COM Call Arguments** dialog box.
12. Zoom from the **Exp** column and create a new line in the Expression Rules repository with a value of **10000**
13. In the **Record Suffix**, create a **Call COM** operation and zoom to the **Call COM Object** dialog box.
14. In the **Object** field, select the ActiveX control.
15. In the **Option** field, select **Set Property**.
16. In the **Property** field, select **Value**.
17. Zoom from the **Value** field to the **COM Call Arguments** dialog box.
18. Zoom from the **Exp** column and create a new line in the Expression Rules repository with a value of **Counter(0)**
19. Run the program.

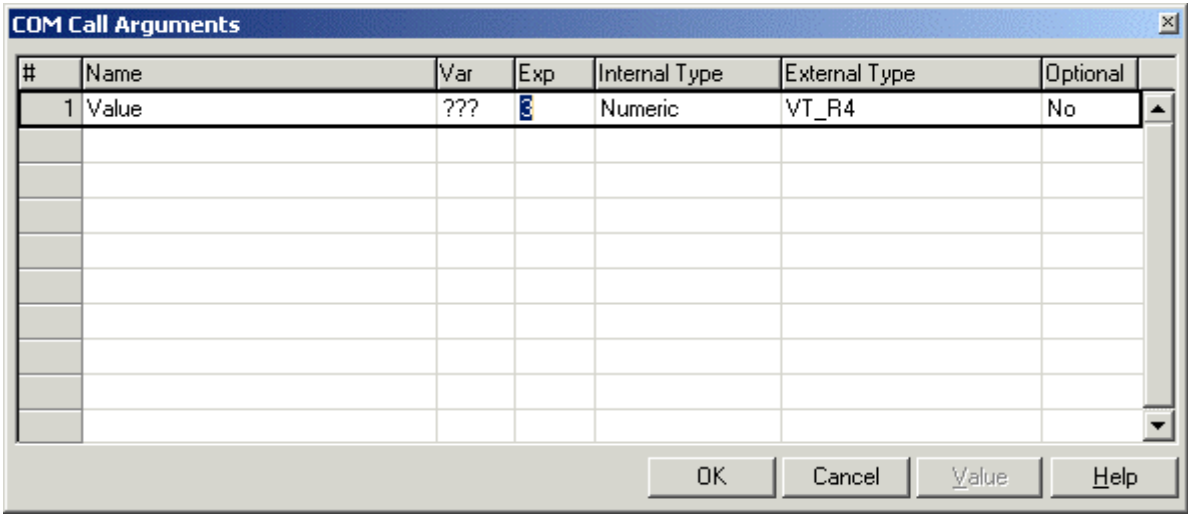
Setting a COM Object Property

1. Create a **Call COM** operation and zoom to the **Call COM Object** dialog box.



2. In the **Object** field, select an **ActiveX** virtual variable.
3. In the **Option** field, select **Set Property**.
4. In the **Property** field, zoom to select a property.

5. Zoom from the **Value** field. The **COM Call Arguments** dialog box will open.



6. Zoom from the **Exp** column, create a new line in the **Expression Rules** repository and set the relevant value. Or, zoom from the **Var** column and select the field that contains the relevant value.
7. Click **OK**.

Calling a COM Object Method

1. Create a **Call COM** operation and zoom to the **Call COM Object** dialog box.
2. In the **Object** field, select an **OLE** or **ActiveX** virtual variable.
3. In the **Option** field, select **Call Method**.
4. Zoom from the **Arguments** field. The **COM Call Arguments** dialog box will open.
5. Zoom from the **Var** column or the **Exp** column and connect the argument with the relevant value.
6. Click **OK**.

Sending and Receiving Data 22

Sending Data to the Clipboard from eDeveloper

Usually, in order to send data to the clipboard, you use the common Ctrl+C shortcut. In eDeveloper, the more acceptable shortcut is Ctrl+Ins, which originated in DOS.

In eDeveloper, there are functions that handle the clipboard: ClipAdd, ClipWrite and ClipRead.

The ClipAdd and ClipWrite functions let the programmer write to the clipboard and later paste the information in other applications by using the Windows pasting ability.

ClipAdd – Adds a value and its picture to the clipboard for operating systems that support clipboard functionality.

This function expects to receive at least two parameters – the value that you want to add (any valid value other than a BLOB value) and its picture.

ClipWrite – The function places the buffer created by the ClipAdd function into the clipboard by using the CF_Text clipboard format.

To send data to Clipboard from an eDeveloper application:

1. Create an online program with one parkable field.
2. Add a user event with **Force Exit** = **Control**
3. Add a handler to the user event.

4. In the handler, add two **Evaluate** operations:
    - evaluate (ClipAdd(a,varpic('A'var,0))
    - evaluate (ClipWrite())
5. Once this program is executed, enter a value in the field and trigger the event.
6. Open Notepad and paste the value.

Note:

You can add more than one value at the same time.

Simply use the ClipAdd function several times before using the ClipWrite function.

A good example would be using the Clipboard function in a multi-marking program.

Sending and Receiving Messages

23

In distributed applications, two different applications often need to communicate with one another to transfer information. One of the ways to accomplish this is message queuing.

Message queuing allows applications to send messages without waiting for a reply and to continue with other processes. Other applications can receive the messages at their leisure and continue processing according to the data received.

This chapter covers the topics listed below:

- Installing eDeveloper's Messaging Capability
- Sending a Message from eDeveloper to MSMQ
- Sending a Transacted Message from eDeveloper to MSMQ
- Sending a Message from eDeveloper to JMS
- Receiving the Messaging Error
- Changing the Location of the Messaging Component

Installing eDeveloper's Messaging Capability

As eDeveloper's messaging capability is not part of the typical installation, you first have to install it.

Part 1
To install eDeveloper's messaging capability:

1. Re-run the eDeveloper installation via the **Add Remove** option of the **Control Panel**.
2. Select the **Messaging Component** option. eDeveloper will create a new directory under the eDeveloper installation directory, called Messaging. It will also add two Logical Names to your Magic.ini file.
   - **MessagingComponentDir** – points to the Messaging component location
   - **MessagingErrorLogFile** – the name of the error log

For more information about these logical names, please consult the Messaging Connectivity Reference Guide. Once you have installed the messaging capability, this document can be found using the Windows Start menu: Programs/Magic 9.4 eDeveloper/ Magic Online Manuals.

Part 2
To add the Messaging component to the Component repository:

1. Create an eDeveloper application or use an existing one.
2. Go to the **Component** repository and create a new entry. Give the new entry a name, for example: "MSMQ".
3. From the **Options** menu, select **Load/Reload**. This will display a dialog asking for the component interface file. The file that should be loaded will be found in the Messaging subdirectory of the eDeveloper installation.
4. In the Messaging subdirectory, you will see a number of .mci files, which are the eDeveloper component interface files. If you want to work with Microsoft

Messaging Queues (MSMQ), select the **MSMQ.mci** file.

5. Return to the **Component** repository and press **F5** to make sure that eDeveloper has located the files correctly. The component is now ready to use.

Sending a Message from eDeveloper to MSMQ

This topic describes how to send a text message to the Microsoft Messaging Queuing (MSMQ) system using eDeveloper. This topic assumes that you have MSMQ installed on your computer as well as eDeveloper's messaging capability (see p.292). For the purpose of this example, we will use a MSMQ private queue called: *eDeveloper*. We will assume that you have loaded the MSMQ component and called it "MSMQ".

To send a message, you need to use three programs from the component:

1. Open Destination – Opens the queue that you need to send a message to
2. Send Message – Sends the message
3. Close Queue – Closes the queue

While the queue is open, we can send as many messages as required.

Part 1

To create the Open Queue program for sending a message to MSMQ:

For the sake of simplicity, we will use a batch program.

1. Create a batch program to send the message. The program should have a numeric virtual variable to hold the internal queue identifier. This should be defined as allowing negative values, for example N17.
2. In the **Task Prefix** of this program, create a **Call Program** operation.
3. Zoom to the Program selection list and you will get the list of programs, including programs from the component. These will appear in red. Each program is prefixed by the name of the component; in our case MSMQ.

4. Select the program called **MSMQ.MSMQ Open Queue**. This program receives 5 parameters:

   - **Format Name** – You should use 0. If you prefer to use a TCP address, you should use 1.
   - **Address** – Enter your computer name. If you entered zero in the first parameter or a valid TCP address if you entered 1.
   - **Queue Name** – This is the full name of the queue. In our example the queue name will be private$\eDeveloper
   - **Access** – You should give the value 'W' because we want to write to this queue. If we wanted to read from this queue, we would give 'R'. When opening a queue in MSMQ, the queue may be opened either for read or for write but not for both.
   - **Share** – Define how other users will access this queue while this program has opened the queue. In our case, we will enter 'A' to allow full access to others.

5. In the return value of the **Call** operation you must enter the **Queue Identifier** variable that was defined earlier. The reason for this is that the program will return a number. This number will either be positive, which will therefore be the queue identifier, or negative in the case of an error. If the open was successful and an identifier used, this will be the number used in the operations that come after.
6. The next operations should be in a block. Set the condition of the block to: if the Queue Identifier is greater than zero. Any other value is an indication that the queue was not opened.

Part 2

To create the Send Message program for sending a message to MSMQ:

1. Create a **Call** operation to a Send Message program.
2. Select the program called **MSMQ.MSMQ Send Message**. This program

receives 8 parameters, only the first five are relevant for this example:

- **Queue Handle** – The queue identifier returned by the open program.
- **Message** – The message itself. If you define a variable to hold the message, this variable should be a BLOB in which the GUI Display style is defined as **Rich Edit**. For your convenience the component has a model that you can use.
- **Datatype** – The Datatype that may be used. To send a regular string, you should enter 'A' for Alpha.
- **Picture** – Only relevant if the Datatype is numeric ('N'). For any other type you can send an empty string.
- **Transaction Mode** – Defines what kind of transaction will be sent. In our example no transaction will be used, so you should enter 'N' for None.

Part 3

To create the Close Destination program for sending a message to MSMQ:

1. Create a **Call** operation to the Close Destination program.
2. Select the program called **MSMQ.MSMQ Close Destination**. This program receives a single parameter, the queue identifier.

Sending a Transacted Message from eDeveloper to MSMQ

This topic describes how to send a transacted text message to the Microsoft Messaging Queuing (MSMQ) system using eDeveloper.

This topic assumes that you have MSMQ installed on your computer and that you are familiar with sending regular messages from eDeveloper to MSMQ.

For the purpose of this example, we will use a MSMQ private queue called, *eDeveloper*. We will assume that you have loaded the MSMQ component and called it "MSMQ".

A transaction in MSMQ is not a part of the Open Queue operation itself but an external entity. Therefore, the user can open two different queues, one for Read access and the other for Write access, within the same transaction.

Transactions in MSMQ are managed by two separate entities:

- Internal – this is managed by MSMQ itself
- MTS – this is managed externally from MSMQ by the Microsoft Transaction Server

When beginning a transaction, the user is required to define where the transaction will be managed.

How does a transaction begin, using eDeveloper's Messaging component?

The component exposes three programs that deal with transactions:

1. Begin Transaction – This creates the transaction itself.
2. Commit Transaction – This commits the transaction and ends it. The transaction may no longer be used.
3. Rollback Transaction – This aborts the transaction and ends it. The transaction may no longer be used.

The Begin Transaction receives a single parameter – I for an internal transaction and M for an MTS transaction. If the transaction was successful, the component returns a number. This number or handle should be used in subsequent operations.

So, how do we send a message that will be part of a transaction? You should be aware that you can only send a transacted message to a queue that was created as a transactional queue. When creating a queue in MSMQ, you define whether or not the queue is a transactional queue.

The eDeveloper Open Queue program is the same whether you want the messages to be sent or read within a transaction. It is up to the developer to open a queue that is a transactional queue.

The order in which you open the queue or begin the transaction is not important.

Once you have opened the queue and begun the transaction, you have two handles, one returned by the Open Queue and the other returned by the Begin Transaction. The handle returned by the Begin Transaction operation is used in the Send Message program in the parameter Transaction Handle. But this is not enough. You also have to define the Transaction Mode as T. This means "Within a defined transaction."

Why do you have to define the transaction mode? There are two types of transactions that may be used in a transacted queue: a regular transaction and a *single message* transaction.

A single message transaction is a transaction that explicitly performs its own Begin Transaction, Send Message, and Commit Transaction for each message. In this case, the transaction handle should not be passed. The advantage of sending a single message transaction is that MSMQ handles all the definitions for the user and ensures that the message is either sent or rolled back.

If a transaction has been opened, you should ensure that you commit the transaction.

Note:

Since a transaction is not part of the queue, you can open a number of different queues and send and receive messages from different queues, all within the same transaction.

Sending a Message from eDeveloper to JMS

This topic describes how to send a text message to the Sun Java Message Service (JMS) using eDeveloper. This topic assumes that you have a server using JMS API installed on your computer as well as eDeveloper's messaging capability (see p.292). We will assume that you have loaded the JMS component and called it "JMS".

To send a message, you need to use three programs from the component:

1. Open Destination – Opens the queue that you need to send a message to
2. Send Message – Sends the message
3. Close Destination – Closes the queue

While the queue is open, we can send and read as many messages as required. You should note that a JMS queue may be opened both for receiving and sending messages at the

same time.

Part 1
To create the Open Destination program for sending a message to JMS:

1. Create a program to send the message. The program should have a numeric virtual variable to hold the internal queue identifier. This should be defined as allowing negative values, for example N17.
2. In the **Task Prefix** of this program, create a **Call Program** operation.
3. Zoom into the Program selection list and you will get the list of programs, including programs from the component. These will appear in red. Each program is prefixed by the name of the component, in our case JMS.
4. Select the program called **JMS.JMS Open Destination**. This program receives various parameters. Only the first 6 are necessary:
    - **Factory Name** – Enter the name of the JNDI queue factory name. For Sun Reference and JBOSS, there is a built-in queue factory name called, QueueConnectionFactory.
    - **Queue Name** – Enter the name of the queue that you want to send the messages to. Sun Reference provides a predefined queue called jms/Queue.
    - **Username** – The user name for the queue. This is an optional parameter.
    - **Password** – The password for the queue. This is an optional parameter.
    - **Transacted Messages** – Here you define whether the messages are part of a transaction or not. In JMS, you define a transaction when you open a queue. From that point on, everything is within a transaction. Therefore, if you do not want a transaction, you should send FALSE to this parameter.
    - **Acknowledgement Mode** – The Acknowledgement mode defines where the acknowledgement will be created. The various options are:
        - Auto Acknowledge – With this mode, when the message is received, the message is automatically acknowledged. This is the preferred method. If you want to use this mode, you should pass 1 to the parameter.

- Client Acknowledge – With this mode, when the message is received, the acknowledgement has to be performed manually. If you want to use this mode, you should pass 2 to the parameter.
- Dups OK – With this mode, acknowledgement is made by JMS but not automatically. Therefore, if you use this method, it is possible that duplicate messages will be sent. If you want to use this mode, you should pass 3 to the parameter.

5. In the return value of the **Call** operation you must enter the **Queue Identifier** variable that was defined earlier. The reason for this is that the program will return a number. This number will either be positive, which will therefore be the queue identifier, or negative in the case of an error. If the open was successful and an identifier used, this will be the number used in the following operations.
6. The next operations should be in a block. Set the condition of the block to: if the Queue Identifier is greater than zero. Any other value is an indication that the queue was not opened.

Part 2

To create the Send Message program for sending a message to JMS:

1. Create a **Call** Operation to the Send Message program.
2. Select the program called **JMS.JMS Send Message**. Only some of the parameters will be discussed:
   - **Queue Handle** – This is the queue identifier returned by the open program.
   - **Message Type** – This is the type of message. For this example we will use a text message. For other types of messages, please refer to the Messaging Connectivity Guide. In order to send a text message, please send **T** as the argument.
   - **Message** – This is the message itself. If you define a variable to hold the message, this variable should be a BLOB in which the GUI Display style is defined as **Rich Edit**. For your convenience the component has a model that you can use.

The other parameters are optional.

Part 3

To create the Close Destination program for sending a message to JMS:

1. Create a **Call** operation to the Close Destination program, in order to close the queue.
2. Select the program called JMS.JMS Close Destination. This program receives a single parameter, the queue identifier.

Note:

Remember that as JMS is based on Java and Java is case-sensitive, all names should be entered exactly as provided by the server.

Receiving the Messaging Error

Messaging errors are automatically written to a log. Often a developer wants to receive the error and perform an action as a result.

An example of this is the "no messages in queue" error or return code –10. The developer may want a message dialog to appear for the end user, such as "There are currently no messages in the queue. Please try again later." This topic describes how to handle these errors in the host application.

For simplicity, this topic assumes that you have messaging capabilities on your computer and that you already know how to work with eDeveloper's Messaging component.

eDeveloper's Messaging component automatically traps messages to a log. Some of these messages are warning messages and some are errors.

The name of this log is defined in the logical name:

```
MessagingErrorLogFile
```

When the messaging component encounters an error, it raises an event that the developer can handle. The component exposes a public event called **Public Event**.

To define the handler for the exposed event:

1. Create a new **User Handler** and zoom to the list.
2. Select the event exposed by the component.
3. Create four **Select Virtual** operations for the four parameters that are returned by the event.

   a. **Message System** – An alpha parameter, which will return one of the following values:

   - **M** – Microsoft Messaging Queue (MSMQ).
   - **J** – Java Messaging Service (JMS)
   - **W** – Websphere Messaging Queue

   b. **Error Code** – The actual error code. A numeric variable that receives negative values, such as N17. Errors will be negative values.

   c. **Validation** – A logical value. True means that the error was a result of the component checking the parameters passed to it. False means that this was an error returned by the messaging system itself.

   d. **Error Message** – The error text as returned by the component.

If you want to provide your own dynamic text for a certain error, such as –10 which is "No messages", then create a Verify operation in which the text is "Dear <Username>, the queue is currently empty". The condition for the operation will be when the Error Code is equal to –10.

Changing the Location of the Messaging Component

The Messaging component and most of its files are stored in the Messaging sub-directory of the eDeveloper installation. Errors are automatically written to a log.

To change the location of the Messaging component:

The location of the Messaging sub-directory is kept in a logical name, which is added to the Magic.ini file during installation. The name of the logical name is:

```
MessagingComponentDir
```

To change the location, all you have to do is change the translation of the logical name.

For example, if the logical name MessagingComponentDir points to: C:\Magic\Magic940\Messaging\ , then change the data so that it points to F:\Magic\Messaging\ .

You must remember to use the slashes appropriate for your operating system. For example, if you are using the Windows operating system, please use "\".

It is important to remember that the logical name must have a trailing slash.

Using Drag-and-Drop Functionality

24

Using eDeveloper, you can easily transfer information between your application and other applications using drag-and-drop functionality. You can also drag and drop information within the same eDeveloper application.

The drag-and-drop feature is used with GUI elements such as the Form, Edit control, Table control, and other controls. You can enable drag and drop in your application by specifying that GUI elements allow a Drag operation from an object or allow a Drop operation on an object.

This chapter covers the topics listed below:

- Dragging Data from eDeveloper to External Applications
- Dragging Data from External Applications to eDeveloper
- Determining Drag-and-Drop Mouse Pointer Appearance
- Dragging Several Controls Together
- Dragging and Dropping User-defined Formats
- Dragging Multiple Records From One Table to Another

Dragging Data from eDeveloper to External Applications

Data can be dragged from eDeveloper to external applications that support drag-and-drop functionality. Follow the steps below to be able to mark the contents of a field in an eDeveloper program and drag the field contents to another application, such as Microsoft Word.

To enable dragging data from eDeveloper to other applications:

1. Go to the task's form, and mark the control from which the data should be dragged.
2. Go to the control property sheet, and set the Allow Drag property to Yes in the details section.

Dragging Data from External Applications to eDeveloper

Objects can be dragged from an external application that supports drag and drop into an eDeveloper application.

Data can be dropped into virtual or real variables when the task is in Create or Modify mode.

There are two ways to drop object data into an eDeveloper application:

- Using a Simple Edit Control
- Using an RTF Edit Control Connected to a BLOB Variable

Using a Simple Edit Control

To enable dragging data from external applications to eDeveloper using a simple Edit control:

1. Select the Edit control where the data should be dropped in the eDeveloper program form.
2. Set the Allow Drop property to Yes on the Edit control property sheet.

Using an RTF Edit Control Connected to a BLOB Variable

To enable dragging data together with its visual attributes from external applications to eDeveloper:

1. Define a virtual variable with a BLOB attribute in your eDeveloper application.
2. Define the GUI display as an RTF Edit control in the property sheet.
3. Generate a form for the task

   .

   The Allow Drag and Allow Drop properties do not appear in the property sheet because the RTF Edit control automatically allows drag and drop.

   When text is dragged from an external application to the RTF Edit control, the data will retain its visual format.

Determining Drag-and-Drop Mouse Pointer Appearance

The eDeveloper engine determines how the mouse pointer icons associated with the drag-and-drop operation look.

To determine the appearance of the drag-and-drop mouse pointer icons, you must:

- Define an Internal Drag Begin handler event
- Use the Evaluate operation with the DragSetCrsr function

Defining an Internal Drag Begin Handler Event

To define the Drag Begin handler event:

1. Define two alpha variables in a task, and generate a form for the task.
2. Set the first Edit control's Allow Drag property to Yes.
3. Set the second Edit control's Allow Drop property to Yes
4. Define a new handler in the task. The event on which the handler is defined is the internal Drag Begin event.

Using the Evaluate operation with the DragSet Crsr Function

You can use the Evaluate operation with the DragSetCrsr function in the Drag Begin handler for different modes that determine how the mouse pointer appears when placed over an area where:

- Data can be dropped - Mode 1
- Data cannot be dropped - Mode 2

For example,

```
DragSetCrsr (1,'PathToCursorFile.cur')
```

causes the new mouse pointer to appear when the data can be dropped, either in an eDeveloper application or in an external application, such as MS Word

Dragging Several Controls Together

eDeveloper's default drag operation only allows the operation to be performed on a single control. Additional programming is required to enable several controls to be dragged together, such as the entire record in a table.

After you follow the steps below, you will be able to mark a record in a program and drag it to an external application, such as Excel or Word, or to an RTF Edit control within an eDeveloper application.

To allow several controls to be dragged together:

1. Define a virtual alpha variable in a line-mode task. The size of the variable depends on the total size of the dragged fields because the variable will be used to concatenate the string contained in the drag buffer.
2. Select the table control on the GUI form, and set the Allow Drag property to yes.
3. In the task, define a handler on the internal Drag begin event.
4. Update the alpha variable in the handler with an empty string, in case the variable has been used before.
5. Update the alpha variable according to the variables you want it to contain. Make the needed data conversions, and concatenate each field in turn to the already existing string. Between every two values, concatenate chr(9), to serve as a delimiter.
6. Evaluate the DragSetData() function with the alpha variable and the number 1, to indicate that the data format is text, as parameters.

Dragging and Dropping User-defined Formats

eDeveloper recognizes Text, OEM text, rich text, HTML, and hyperlink formats. In addition to these standard formats, eDeveloper can also handle other formats that have been defined and recognized within eDeveloper.

Before you try to drag and drop user-defined formats, you should check that the format is supported by the dropped object. Use the DropFormat function, which returns true if the format specified is supported by the object.

To enable user-defined formats to be dragged and dropped:

1. Click Environment on the Settings menu, and set the Drop data supported user formats to `FileName` on the External tab.
2. Define two virtual variables with an alpha attribute. Place them on a screen and set the Allow Drop property of the first one to yes.
3. Define a handler on the internal Drop event. Update the second variable in the handler with the expression `DropGetData (0,'FileName')`.
4. To check that the FileName format defined in this example can be dragged and dropped, run the program. Open Windows explorer and drag a file from it to the first control. The second control will be updated with the file name.

Dragging Multiple Records From One Table to Another

You can also drag and drop a set of multiple records in eDeveloper by using an auxiliary vector that stores all the indexes of the marked records in the source table. These indexes are then pasted in the destination table when they are dropped there.

Allowing multiple records to be dragged from one table to another includes several steps, which include:

- Creating the User Interface
- Defining Two Handlers
- Defining Two Link Operations

Creating the User Interface

The first thing you must do is define two database tables that have the same structure. One table will serve as the source table, and the other will be the destination table. For this example, you should use two-column tables that contains a numeric code and an alpha value that represents the name.

Creating the program that contains the tables:

1. Generate an online line-mode program that browses the source table.
2. In the program, define two calls for a subtask: one in the record main level and one in the task prefix level.
3. Define the subtask as another online task that browses the destination table in line mode.
4. Set the end condition of the subtask to `Level (1)='TP'` in the task properties.
5. Set the Close task window parameter to No on the Behavior tab of the Task

Control dialog.

6. Design the screen so that the two tables seem to belong to the same task.
7. Return to the parent task, and define a virtual variable. Name the variable `Index vector` and assign it a vector attribute.

Defining Two Handlers

Now you need to define two handlers, one for the Drag begin event and one for the Drop operation.

Defining the Drag Begin Event Handler

To define the internal Drag begin event handler:

1. Define a new handler for the internal event Drag begin as described in the section on Determining Drag-and-Drop Mouse Pointer Appearance on page 306.
2. Update the index vector with null() in the handler if this is the first record the handler runs. Specify the condition as `MMCurr (0)=1`
3. Set a new value in the vector for each marked record using an evaluate operation with the expression:
   `VecSet (reference to the index vector variable ,MMCurr (0), Code field from the source table)`
   This results in a vector that contains the code indexes for all the marked records.

Defining the Drop Operation Handler

To define the internal Drop operation handler:

1. Go back to the subtask, and define an internal Drop operation handler.
2. Set a block loop in the handler with the condition to
   `LoopCounter ()<=VecSize (Index Vector)`
3. Call another subtask inside the loop.

4. Pass the current value of the index vector to the subtask as an argument using the expression `VecGet(Index vector, loopcounter()).`
5. Outside the block, raise the view refresh internal event.
6. Define the called sub-task as a batch process with no main table. Set the end task condition to Yes and the evaluate condition parameter to after.
7. In the subtask, define a numeric parameter where the current index value will be received.

Defining Two Link Operations

Now you must define two different link operations, one for the source table and one for the destination table.

1. Define a **Link Query** operation for the source table, and specify the numeric parameter defined in the subtask for the Locate value.
2. Define a **Link Write** operation for the destination table, and specify the numeric parameter defined in the subtask for both the Locate value and the init column expression.
3. In the **Record Suffix**, update the destination table's alpha variable with the alpha value from the source table.

Using the Block Loop Operation

25

The Block Loop operation allows a set of operations to run continuously while a certain condition is evaluated to true, just like a DO loop.

You can set any valid eDeveloper expression as the condition of a Block Loop, and you can also have a new function dedicated to the Block Loop to serve as an inner counter for the operation iterations.

The block operation is not meant to be used within the record main level.

This chapter covers the topics listed below:

Running a Set of Operations Continuously

Monitoring the Number of Iterations

Running a Set of Operations Continuously

To cause a set of operations to be repeated continuously:

1. Define a new Block operation in eDeveloper, and set the block type to Loop.
2. In the Condition column, set the condition of the block.
   Set the condition of the block to LoopCounter()<=N to have the loop run N times.

   You can also use any valid eDeveloper expression as the condition for the block loop. For example, you can easily parse a string by using a block loop and setting the condition to LoopCounter ()<=StrTokenCnt(string, delimiter), while each iteration makes use of the LoopCounter () function again as the index for the

StrToken() function.

Monitoring the Number of Iterations

Monitoring the number of iterations is accomplished using the LoopCounter() function.

Set the condition to LoopCounter()<=N to have the set of operations performed N times. This makes it unnecessary to have an application counter monitor the number of times the set of operations in the block has been executed.

The use of the LoopCounter() function is limited to the execution of the block. Outside the block, the function returns 0.

Using the List Box 26

New eDeveloper list boxes let you select more than one value. This is done by connecting the list box to a vector type variable that allows a set of values to be stored. Depending on the cell model of the vector, the list box can be associated with different data types, not necessarily alpha types.

This chapter covers the topics listed below:

Enabling Selection of More Than One List Box Item

To allow more than one item to be selected from a list box:

1. Define a model for the vector cell, such as alpha 6.
2. Define a virtual variable with a vector attribute. In the variable properties, set the cell model to the model you defined.
3. Define another virtual variable, and set its attribute to BLOB. In the properties of this variable, set the GUI display to rich text edit. This variable will later contain the list of selected items.
4. Create a form for the task. Connect the vector to a list box control. In the list box property sheet, define the selection mode as Multiple.
5. Enter values in the list by connecting the list to a table or by defining the label.

Checking the Multiple Selection List

To check the selection list:

1. Run the task.
2. Press the control button.
3. Use the mouse to select more than one item.
4. Click the Rich Edit Format control to enter the values selected from the list box.

Entering the Values While Remaining on the Selection List

Note that a multiple-selection list box only triggers the control change event when the focus is moved to another variable. If you want the control change event to be triggered when the focus is still on the list box, you need to modify the program so that it has a similar handler on some other event, such as a key combination. In this case, because the variable will not yet be updated, the concatenation should be done with the expression `VecGet(editget(),LoopCounter())`

Allowing Retrieval of Non-string Attributes from a List Box

To allow non-string attributes to be selected from a list box, you need to define a Control Change level for the vector variable and follow the steps below in the level:

1. Update the BLOB variable with a blank string.
2. Open a block loop, and define the expression LoopCounter() `<= VecSize(vector variable)` as the condition for the loop.
3. Inside the loop, update the BLOB variable so that it concatenates `VecGet(vector, LoopCounter())&', '` to the previous contents of the BLOB field.

Handling Buffers 27

An eDeveloper application is sometimes required to interact with third-party modules. This interaction may involve an exchange of information through binary collection of data commonly known as *structures* or more generally known as *buffers*.

Creating a Structure

eDeveloper lets you handle buffers using BLOB fields. By using the special Buffer management functions, it is possible to construct the structure required. In addition, by using matching functions it is possible to retrieve data from a structure.

To create a structure:

1. Define a BLOB variable that will hold the binary information.

   Buffers are binary streams of data. You will need to be familiar with type sizes as they are implemented in computer languages. For example, a Numeric value in the range of –32,768 to 32,767 or 0 to 65,575 can be stored in a numeric memory variable called "Double" and use 2 bytes of memory.

2. Use the special Buffer Management functions in eDeveloper to add values and store them in the structure that is stored in the binary BLOB.

3. If you want to add a numeric value into a buffer, use the BufSetNum() function. For example: BufSetNum ('H'VAR,1,B,3,8)

   The BufSetNum parameters are:

   - **Variable pointer** – Like all BufSet() functions, the first parameter is a variable

pointer to a BLOB variable that holds the current structure that should be added with a value.

- **Position** – The position to which the value should be added.
  In the example, the position is 1, meaning the beginning of the structure.

  Remember that a buffer is a stream of data. Therefore, position is very important as to indicate where in the binary stream the value should be added.
  The position of any consecutive value should take into consideration the position and length of the values set before it.

- **Value** – The value to be added to the structure. The value should match the type of function. For example, it should be numeric for certain functions such as BufSetNum() and BufSetBit(), while it should be an Alpha value for the BufSetAlpha() function.
  The value can be any valid expression, such as variable and static.

- **Storage** – The fourth parameter for all BufSet functions, except BufSetVariant(), is the storage type, indicated by a numeric value.
  In the example, the Numeric values added to the 'H'Var buffer is stored as a Float. The available storage types are displayed in the eDeveloper Storage Type table (see p.318).

- **Length** – The last parameter is the length of the value that is being added. It should match the value storage type and size. In the example it is set to 8.

eDeveloper Storage Type table

#	Attribute	Storage ID	Storage Name	Length
1	Alpha	1	String – Non-null terminated string	<32K
2	Alpha	2	Zstring – Null-terminated string	<32K
3	Alpha	3	Lstring – Compressed string with a short integer containing the storage length	<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	1
15	Logical	2	Dbase containing T,F	1
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)	

For more details about adding elements to structures, please see the Buffer Management function descriptions in the *eDeveloper Reference Guide* or the help (F1).

Sending a Buffer from eDeveloper

You can use the Call UDP operation to connect to third-party DLLs as well as pass arguments and receive arguments.

1. Create a structure expected by the third-part DLL function (as described in the Creating a Structure topic)
2. Once the structure is stored in a BLOB field in eDeveloper, create a **Call UDP** (User Defined Procedure) operation to connect with the third-party DLL.
3. In the Call UDP operation, enter a string expression that indicates the path to the function that needs to be called.
   Usually, these DLLs are not compiled especially for eDeveloper's usage, so the string should start with the '@' symbol.
   For example:

   ```
   '@kernel32.GetSystemTime'
   ```

   This tells eDeveloper to find the "GetSystemTime" function from the Kernel32.DLL, which is not a DLL compiled especially for eDeveloper's use.

   It is also possible to set a file path to the DLL; for example:

   ```
   '@C:\Windows\System32\kernel32.GetSystemTime'
   ```

When no path is specified, eDeveloper searches for the DLL, first in eDeveloper's directory and then anywhere else, according to the environment variable "PATH". Since System32 directory is usually in the path, there is no need to write the path into the string.

4. Define the first parameter for the Call UDP operation, which should be the "Function Mask" – a string value that represents the function header declaration. In the example above, the GetSystemTime function's header is:

```
Public Declare Sub GetSystemTime Lib "kernel32" Alias
"GetSystemTime" (lpSystemTime As SYSTEMTIME)
```

Note: The description of the function can only be obtained from the vendor of the DLL or if you have access to its source.

In this example, the "Function Mask" is 'T0' because the function expects a Structure called SYSTEMTIME and it is declared as Sub (void), so there is no return value.

The "Function Mask" is a string with the following characters:
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
T – Structure

5. Set the next parameters, which are the arguments to be sent to the function. In the example, you need to send the BLOB variable that holds the structure SYSTEMTIME.
When sending a variable it will be sent by Reference and the function will update

it accordingly.

6. Use the Buffer Management functions to fetch the values.

Advanced notes:

The SYSTEMTIME structure is defined as (according to the DLL sources):

```
Public Type SYSTEMTIME
        wYear As Integer
        wMonth As Integer
        wDayOfWeek As Integer
        wDay As Integer
        wHour As Integer
        wMinute As Integer
        wSecond As Integer
        wMilliseconds As Integer
End Type
```

To create the SYSTEMTIME structure, you will need to run the following Buffer Management functions on a BLOB field:

```
BufSetNum ('B'VAR,1,0,1,2)
BufSetNum ('B'VAR,3,0,1,2)
BufSetNum ('B'VAR,5,0,1,2)
BufSetNum ('B'VAR,7,0,1,2)
BufSetNum ('B'VAR,9,0,1,2)
BufSetNum ('B'VAR,11,0,1,2)
BufSetNum ('B'VAR,13,0,1,2)
BufSetNum ('B'VAR,15,0,1,2)
```

This will initialize the BLOB ('B'VAR) with an empty structure full of Zero values to all the elements in it.

Using the matching BufGetNum() function you will be able to read each of the values returned by the DLL into the buffer (after the Call UDP operation was executed).

Index