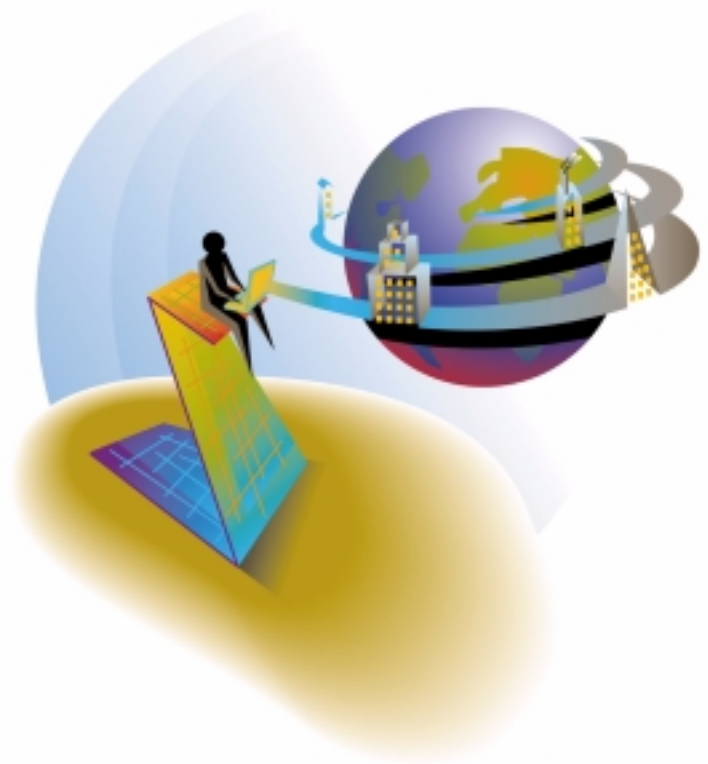


eDeveloper **V9.4**



Interactive Web Application Development & Deployment

The information in this document is subject to change without prior notice and does not represent a commitment on the part of MSE.

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

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

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

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

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

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

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

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

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

PC/TCP® Network Software is a registered trademark of FTP Software Inc.

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

Macromedia® Dreamweaver® is a registered trademark of Macromedia, Inc.

VeriSign® is a registered trademark of VeriSign, Inc.

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

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

03 02 01 6 5 4 3 2 1

Copyright • 2001, 2003 by Magic Software Enterprises Ltd. All rights reserved.

Table of Contents

<i>INTERACTIVE WEB APPLICATIONS.....</i>	<i>5</i>
Why Browser-based?	5
World Wide Web	5
A Common Client	5
A Thin Client.....	5
Browser-based Development	6
Browser-based Deployment.....	6
What's Next?	6
<i>SUPPORTING ARCHITECTURE.....</i>	<i>7</i>
Magic Application Server Infrastructure	7
Distributed Modules.....	7
Magic Browser Client Construction	9
Creating the HTML Result Page.....	9
Persistent Browser Client Module.....	10
<i>BROWSER TASK LIFE CYCLE</i>	<i>12</i>
Browser Task Initialization	12
Browser Task in Action	13
Server-side and Client-side Logic	15
Context Management	17
Transactions	18
<i>CONSTRUCTING A BROWSER TASK.....</i>	<i>20</i>
Basic Definitions.....	20
Browser Task Type	20
Dataview Definition	20
Task Interface.....	21
Browser Form.....	21
Simply HTML	21

Browser Form Editing	24
Line Mode Display	25
Subforms	27
The VCR Toolbar	29
The Edit Toolbar	30
Browser Task Settings	31
Chunk Size Expression	31
Exit URL	31
Selection Table	31
Main Display	31
Transaction Mode	31
SQL Command	31

RUNTIME BEHAVIOR AND CONSIDERATIONS.....32

Call Operation	32
Calling another Browser Task	32
Calling a Batch Task	34
Calling an Online Task	34
Verify Operation	34
Client-side Messages	34
System Event Handlers	35
Browser Internal Handling	35
Error Handling	35
Exceptions in the Browser Task	35
Closing the Browser Window	36

SUMMARY.....37

Easy Programming	37
Unified Concept	37
Simplified Paradigm	37
The Tool	37

Interactive Web Applications

Easy development and deployment of Interactive Web Applications

Magic eDeveloper lets you easily develop and deploy high-level business applications using a browser as the front-end of your application. Magic eDeveloper provides full interactive browser client abilities, supported by a robust Magic Application Server, all generated by the simple table-driven development paradigm. This document provides a technical overview of this new technology for rapid Interactive Web Application development and deployment.

Why Browser-based?

There are many reasons why your application should be browser-based.

World Wide Web



The browser is the most familiar way to access Web sites, Web-related material, and Web applications. By making your application browser-based, it becomes a Web-based application that can be accessed around the world, using the common Internet network.

A Common Client



The browser has become a standard way to access various types of information, and it can be found on any desktop computer. It is therefore familiar to most users.

A Thin Client



With eDeveloper there is no need to install proprietary software on the client side. Magic's Application Server provides the browser with all the data, logic, and flow-management module on demand. All the application's software elements reside solely on the server

side, which allows ease of management and maintenance, low ownership costs, and scalability.

Browser-based Development

eDeveloper lets you rapidly develop interactive Web applications. Using the easy table-driven toolkit, you can define your application infrastructure, components and logic with ease. No knowledge of common browser-related programming, such as Java or Java script, is required.

Browser-based Deployment

The Magic Multi-threaded Application Server handles high volume transactions and high rates of requests with ease. The Magic Application Server handles each individual client and transparently manages the context of each current user. The HTML-based application interface is passed to the client together with the XML-formatted application logic.

What's Next?

Read this document to gain a full understanding of the concept behind Magic's Interactive Web Application Development and Deployment paradigm. Learn how it works and how you, as a Magic developer, can easily create your own browser-based applications.

Supporting Architecture

How does an end user access the application? And, how does the Magic Application Server handle the user's request?

The Magic Application Server deployment environment is easily constructed using the Distributed Application Architecture provided by eDeveloper. The Magic Application Server provides your front end with automatic and optimized context management designed for handling large-scale concurrent users.

Magic Application Server Infrastructure

The Magic application runtime engine can be developed to handle a browser request using a Web browser.

Distributed Modules



The basic modules required to construct the Magic Application Server are:

WWW Service Capabilities

The WWW server, Web server, is required to receive an HTTP request from remote browser clients. Using the Magic Internet Requester, the Web server can forward the HTTP request to the Magic Application Server.

Magic Internet Requester

eDeveloper provides you with an Internet Requester module, which is made available to the Web server as an executed module. When a request is made by a browser to the Requester, the module passes the request, with its accompanying data, to an idle Magic Application Server. The Magic Internet Requester module can locate an idle application server using the pool of server engines maintained by the Magic Request Broker.

Magic Request Broker

eDeveloper provides you with an easy middleware agent known as the Magic Request Broker. The Magic Request Broker handles all the available Magic

Application Server engines and directs each request from the Magic Internet Requester to the available application server engine. The Magic Request Broker provides load balancing and recovery capabilities to handle any fail over.

Magic Application Server

The Magic Application Server lies at the heart of Interactive Web Application deployment. It is the actual runtime unit, which handles each request and executes the entire application logic for each type of request it receives. The Magic Application Server needs to know the location of the Magic Request Broker, connect to it, and then make itself available to the Magic Internet Requester.

The Magic Application Server engine is designed to handle multiple requests using a single engine process. This is achieved using the multi-threading capabilities of the Magic Server engine.

Distribution of the Modules

The above modules can be installed on the same machine or distributed among different machines, even on machines using different operating systems.

The installation procedure for eDeveloper and the Magic Enterprise Server automatically installs all the required modules and configures the system to prepare the infrastructure for deployment. The Web server must reside on the same machine as the installed Magic product.

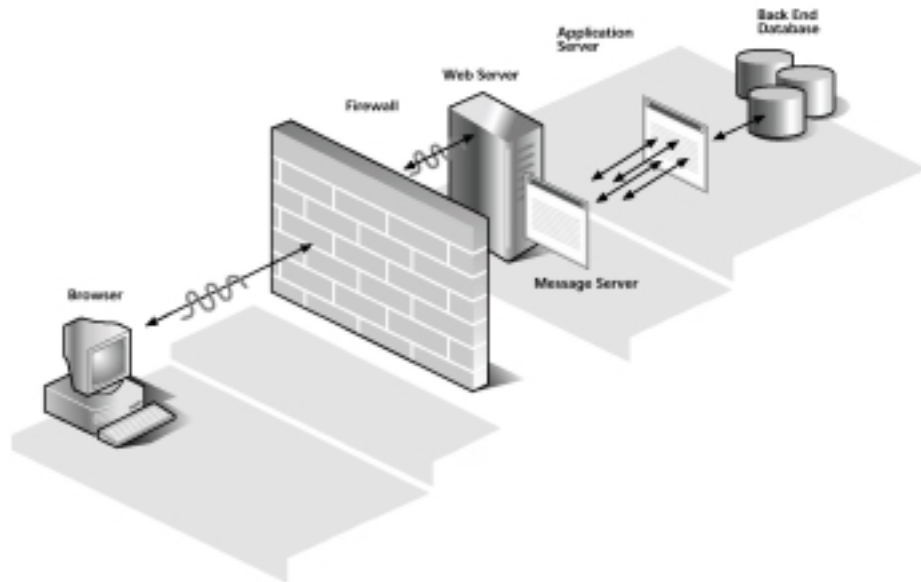


FIGURE 1: This image illustrates how a browser client communicates with the application server, and how the back end interacts with the distributed Magic modules

Magic Browser Client Construction

Although the entire definition of a browser task is constructed and defined using the simple Magic paradigm, the Magic Application Server translates the entire task interface information, logic, and data content into one HTML result page. This page displays the interface according to its design, functions, and the defined task logic.

Creating the HTML Result Page



All the data required for the browser task, and the logic defined for the task, is added to the simple HTML interface definition in XML format. The XML-formatted information is handled by the browser client modules, and provides the end user with a full interactive real-time data application.

XML-formatted Information

The XML portion of the page is hashed so that the end user viewing the source of the HTML result page cannot see the inner structure of the task. The XML content is also compressed into a small volume, to reduce the amount of information returned to the browser.

Browser Client Modules

All task information and data provided in XML format is handled and executed using two modules. One module is a Java applet that serves as the actual browser client engine. The other module is a JavaScript module, which provides a means of interaction between the HTML content and Java applet. Each eDeveloper browser task uses the same browser client modules. This means that the browser client modules are the same for every browser task in your application.

The module files are provided with the installation of the Magic product.

The Module File Names

The file names of these modules have different names for each major version and maintenance version (service pack) of the Magic Engine.

For example, the Java applet provided with eDeveloper version 9.01 Service Pack 1 is **MGBC901_03.cab**, and the JavaScript module for this version is **MGBC901_03.js**. The Magic Application Server embeds the location and names of these modules into the HTML result page.

The names of the browser client module files derive from the major version of the engine (e.g. 901) and the applet version (e.g. 03). Therefore, the module names for this version would be **MGBC901_03.cab** and **MGBC901_03.js**.

Where the browser client modules are supplied without an accompanying major version or a service pack, the name is followed by an additional string. The string should be specified in the Magic Engine Environment Setting, Settings\Environment\Application Sever**Browser Client sub-version**. The string you specify here will be added at the end of the file name, preceded by an underscore. For example, if the browser client module provided after Service Pack

1 is called **MGBC901_03_ISFGE.cab**, you would enter `ISFGE` in the Environment setting.

File Location

The location of these files should be in a directory that is available to the Web server, a virtual directory. The alias of this directory is provided to the Magic Application Server engine by setting the **Web Document Alias** property under Settings\Environment\Application Server.

Persistent Browser Client Module



The Java applet, which is the main part of the browser client engine module, needs to be loaded by the browser client. With some browsers, the module can take considerable download time. The Magic Application server provides you with the ability to make the Java applet module persistent on the client side. This means that once the applet is loaded for the first time, it is kept on the client machine, and every other browser task requiring this module, takes it from the client.

The persistent option of the browser client module requires the end user's confirmation in the Browser Confirmation dialog box.

Browser Task Launcher

The initial response to a request for the browser task is a small HTML page that comes with a small launcher applet. This applet sends the browser client module to the client and returns the link information to the application server. If the browser client module is already persistent on the client side, the launcher immediately returns the request to the application server, informing the server of the local browser client module's location. The launcher module file, **MGLauncher.cab**, should be placed in the same directory as the browser client modules.

The response of the returned request results in the final HTML page, which includes the interface definition and XML information about data and logic.

Since the launcher module is designed to place the browser client module locally, confirmation by the end user is required. If the end user does not make the confirmation, the launcher does not preload the browser client module. The launcher then directs the application server to load the applet with the HTML result page, and the URL of the embedded browser client module is directed to the applet residing on the Web server.

If the user accepts the launcher certification, additional confirmation is required for the actual browser client module. If the user chooses to trust the provider of the browser client module, Magic Software Enterprises Ltd., the user's confirmation is no longer required.

Signed Browser Client Module

The Browser Client Java applet module has been certified and digitally authenticated by VeriSign®. This means that the applet code is verified by the VeriSign® authentication certificate, assuring the end user that the module has not been tampered with.

The signed applet file differs from the regular applet file. Its file name is the same as the unassigned module, and is followed by the letter *s*: for example, e.g. **MGBC901_03S.cab**.

Note 1

Using the Persistent Browser Client Module saves download time from the Web server. It is best to use this feature in your application. If you choose to utilize the Persistent Browser Client Module, we suggest you instruct your end users to trust the signed applet they will be loading.

Note 2

You can define your application server to work with a non-persistent applet module. You can set the **Persistent Browser Client module** property under the Settings\Environment\Application Server section to **NO**. By doing this the initial response does not contain the **MGLauncher.cab** applet, and the page only loads the main browser client module directly from the Web server.

Browser Task Life Cycle

Learn more about the lifecycle of a browser task, from the moment a request is made, through the activity performed within the task, to the completion of the task

A browser task has the full functionality of an online task. A browser task can open a transaction at the Record or Task levels, and execute any handler. A browser task can also implicitly perform the Task, Record and Control level handlers, handle all basic data querying and manipulation (scroll, modify, delete, create etc.), and it can perform any Magic operation and function.

Browser Task Initialization



Upon the initial request to execute a browser program, the Magic Engine opens the requested browser task. A context is opened for this request and a unique context ID generated. A short response is immediately returned to the browser, informing it of the created context.

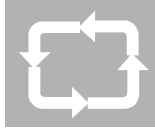
If a Persistent Java applet is detected, the launcher module is invoked on the client. As the context response completes its short process on the client, a consecutive request is made back to the application server, using the context ID generated for this context.

After receiving the second request, the browser task performs the following actions on the server:

1. Implicit initialization, which involves opening the DB tables provided for the task, initializing the virtual variables, determining range and locate values, and creating the initial Dataview.
2. The Task Prefix level handler set of operations.

Following these server-side operations, the HTML result page is created, using all the related data affected by the first Task and Record Prefix modifications. This page is returned to the browser client to be constructed and then made available to the end user.

Browser Task in Action



Once the browser task is constructed on the client side, it becomes available to the end user for handling the data and submission to the logic defined for the task and its data. The task logic, written in the form of handlers, is executed as defined on the page, in addition to standard runtime functionality.

Navigation

The browser task provides the end user with an easy way to navigate through the data supplied.

You can scroll through the data using key combinations to perform the following actions:

Key	Action
Up	Previous Record
Down	Next Record
Page Up	Previous page
Page Down	Next page
CTRL+HOME	Beginning of Table
CTRL+END	End of Table

Easy scrolling is also provided by a VCR image that you can place on the HTML page. For more information see page 29.

Using the **TAB** and **SHIFT+TAB** key combinations, you can move through the fields, while maintaining record cycling ability.

Handlers

The Browser Client Modules automatically invoke all relevant handlers defined for the task, both implicit and explicit.

TASK LEVEL

The Task level handlers, both Prefix and Suffix, are executed on the server side. This means that no operation that halts the task execution can be executed in these levels.

RECORD LEVEL

The Record Prefix for each record is performed when the end user skips to that record. The Record Suffix is performed either after exiting a modified record, exiting a record when the task property 'Force Record Suffix' is set to Yes, or after deleting a record.

The Record Main level is only used for dataview definition. This means that it can only execute the Select and Link operations. Other operations in the Record Main level will be ignored.

CONTROL LEVEL

The Control Prefix of any control is performed when parking on a control. The Control Suffix of any control is performed once the cursor is removed from that control. The control verification handler of a control is performed when exiting a control before its Suffix, and when passing over this control where the record has been modified.

ERROR LEVEL

An Error level handler is executed when the corresponding error is encountered. For more information see the Error Handling chapter of the *Magic Reference Guide*.

OTHER HANDLERS

All other handlers such as System, Internal, User, Time, and Expression handlers are executed when the corresponding event is raised and the task is in idle mode.

Task Modes

The browser task keeps the basic task mode rules of the Query, Modify, Create and Delete modes, which 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.

To switch among the task modes in runtime, you only need to raise the corresponding internal events: Query Records, Modify Records, Create Records, and Delete Records.

Data Manipulation

The Browser Client Module handles any modification of the data, whether it is updating a record, creating a new record, or deleting a record. All data manipulation statements are kept by the Browser Client Module and are transmitted to the server at the following instances:

1. After exiting a record in a Record level transaction.
2. After closing the task in a Task level transaction.
3. When the task returns to the server due to a server-side operation or function. For more information on server side and client-side operations and functions, see the section on **Server-side and Client-side logic** on page 15.

Re-computation

The Browser Client Module performs any required re-computation upon modification of the data. The values of variables, linked records, and visibility properties based on modified data are automatically re-computed.

Task Termination

A browser task only terminates properly if the **Exit** or **Close** internal events are raised correctly, or if the **End Task Condition** and the **Evaluate Task Condition** task properties are evaluated in a way that causes the task to terminate. Closing the browser task window does not terminate the task properly.

If the task is not properly terminated, the Task level transaction or the Record level transaction of the last parked record is rolled back.

Server-side and Client-side Logic



Browser task logic may include all the available Magic operations and most functions. However, some of the Magic operations and functions cannot be executed on the client side because by their nature they cannot be executed by a browser. Some examples are the Output Form operation or the DBDEL function.

Transparent Handling

The Browser Client Module automatically distinguishes between server-side and client-side operations and functions. When the HTML result page is created, the XML-based logic information already includes the information for each operation, whether server side or client side, according to the type of operation or the function that the operation uses.

Note

In a given list of operations, the browser client performs each operation according to its resolved-side execution. This means that a list of mixed-side operations will cause the browser client to switch back to the server for each operation that differs in its side execution. It is best to minimize the use of server-side functions and operations. If you decide to use these functions and operations, we suggest that you list them in the Operations list consecutively.

Client-Side Operations

Below is a list of all the operations that can be executed on the client:

- Select
- Verify
- Update
- Block
- Evaluate
- Raise Event

All other operations are server side only.

Client-Side Functions

Below is a list of all the functions that can be executed on the client:

+, -, *, /, MOD, ^, &, =, <>, <, <=, >, >=, OR, AND, NOT, ABS, ACOS, ADDDATE, ADDTIME, ASC, ASIN, ATAN, BOM, BOY, CALLJS, CALLOBJ, CASE, CDOW, CHEIGHT, CHKDGT, CHR, CLEFT, CLICKCX, CLICKCY, CLICKWX, CLICKWY, CMONTH, COS, CRC, CTOP, CTRLNAME, CWIDTH, CURROW, DATE, DAY, DBROUND, DEL, DOW, DSTRT, DVAL, EDITGET, EDITSET, EOM, EOY, EXP, FILL, FIX, FLIP, FLOW, HOUR, HSTR, HVAL, IDLE, IF, INS, INSTR, ISDEFAULT, ISNULL, LASTPARK, LEFT, LEN, LEVEL, LOG, LOWER, LTRIM, MAX, MAXMAGIC, MID, MIN, MINMAGIC, MINUTE, MONTH, MSTR, MVAL, NDOW, NMONTH, NULL, RAND, RANGE, REP, REPSTR, RIGHT, ROUND, RTRIM, SECOND, SETCRSR, SIN, SOUNDX, STAT, STR, STRTOKEN, STRTOKENCNT, TAN, TDEPTH, THIS, TIME, TRIM, TSTR, TVAL, UPPER, VAL, VARATTR, VARCURR, VARINP, VARMOD, VARNAME, VARPRESV, VARSET, VIEWMOD, WINBOX, YEAR.

Server-Side Functions

Below is a list of all the functions that are executed by the server:

ANSI2OEM, BLB2FILE, CALLDLL, CALLDLLF, CALLDLLS, CALLPROG, CLRCACHE, CNDRANGE, COUNTER, CTXKILL, CTXLSTUSE, CTXNUM, CTXPROG, CTXSIZE, CTXSIZE, CTXSTAT, CURRPOSITON, DBCACHE, DBCOPY, DBDEL, DBDISCNT, DBERR, DBEXIST, DBNAME, DBRECS, DBRELOAD, DBSIZE, DDEBEGIN, DDEEND, DDEGET, DDEPOKE, DDERR, DDEXEC, DELAY, DISCSVR, EOF, EOP, ERRDATABASENAME, ERRDBMSCODE, ERRDBMSMESSAGE, ERRMAGICNAME, ERRPOSITION, ERRTABLENAME, EUROCNV, EURODEL, EUROGET, EUROSET, EUROUPD, EVALSTR, EXPCALC, FILE2BLB, FILE2OLE, FLWMTR, GETLANG, GETPARAM, GROUPADD, INIGET, INIGETLN, INIPUT, INTRANS, IOCOPY, IOCURR, IODEL, IOEXIST, IOREN, IOSIZE, KBGET, KBPUR, LIKE, LINE, LMCHKIN, LMCHKOUT, LMUVSTR, LOCK, LOGICAL, MDATE, MLSTRANS, OEM2ANSI, OWNER, PAGE, PPD, PREF, PROG, PROGIDX, RIGHTADD, RIGHTS, ROLLBACK, RQEXE, RQLOAD, RQQUEDEL, RQQUELST, RQQUEPRI, RQREQINF, RQREQLST, RQRTAPP, RQRTAPPS, RQRTINF, RQRTS, RQRTTRM, RQSTAT, RUNMODE, SETLANG, SETPARAM, SYS, TERM, TEXT, TRANSMODE, UDF, UNLOCK, USER, USERADD, VARPIC, VISUAL.

Irrelevant functions

Below is a list of all the functions that are irrelevant to a browser task:

CLEFTMDI, CTOPMDI, CTRLHWND, FILE2REQ, FILEDLG, HITZRDER, MENU, MMCURR, MMSTOP, MMCOUNT, MNUENABL, MNUSHOW, MNUCHECK, RESMAGIC, WEBREF, WINHWND.

Dynamic Interface Elements Presentation

HTML Control properties that are defined by an expression cannot include a server-side function. Any HTML Control property defined by a server-side function will be disregarded.

Link Operation

Although the Link operation is not a procedural operation, it is a server-side operation. This means that every time a link is re-computed, the browser refers to the server to execute the new link.

Context Management



Each new activation of a browser task creates a context on the server side. The context logs the state of the task from the moment it is activated to the moment it is terminated. Each context is identified by a context ID that lets the server identify each consecutive request of the same context.

Context ID

When a new context is created, a unique context ID is generated for the new context, and this context ID is transmitted back to the client. The client returns the context ID to the application with every request made throughout the life cycle of the browser task. In this way, the application server can identify each request from any browser as belonging to a specific context, and can keep serving this client within the context of its task.

Context Content

The context contains all the information about the task structure, such as all the tasks and subtasks opened from the main browser task, and the location of the client within the task runtime tree structure. The context also keeps the database cursors opened for this task and its subtasks. Thus, each retrieval of data is carried out using these cursors.

The context keeps all data manipulation statements that are passed back to the server within a defined transaction.

The context maintains the runtime information of the browser task, which is local for this specific context. The runtime information includes memory tables, resident tables, Main Program variables, global parameters, which are parameters set by the SETPARAM function, and Environment settings. The instantiation of these runtime information units occurs separately for each new context and any modification is only visible to that context.

Context Inactivity Timeout

The context information kept on the server side requires memory resources. To relieve the server from keeping numerous contexts that might use too many system resources, a timeout can be set for the context.

The **Context Inactivity Timeout** setting can be found in Settings\Environment\Application Server. This setting determines the time interval in which the client is checked for inactivity. Context inactivity is defined as an absence of client/server interaction during the life of a browser task.

The context inactivity timeout is set in 10ths of a second. The default setting is 6000 (10 minutes).

When the context timeout is reached, the context is cleared from the server. By setting a reasonable timeout, you can prevent abandoned contexts from stacking up on your server.

When a browser is activated from Toolkit mode, the Context Inactivity Timeout is infinite.

Post Context Unload Timeout

The browser can exit a running browser task by moving back or forward from the browser task page, or by closing the browser window. These actions move the context into pending mode, where the browser awaits a return to the browser task page.

When the browser returns to the browser task, the context is reloaded, the information preserved on the server is refreshed, and the end user can continue using the browser task.

The **Post Context Unload Timeout** setting can be found in Settings\Environment\Application Server. This setting determines the time interval during which the context is kept until the browser returns to the browser task.

The Context Inactivity Timeout is set in 10ths of a second.

RECONSTRUCTION OF THE TASK

When a user returns to the browser task, by moving forward or back, a request is sent to the server requesting recovery of the context. The context is then recovered, and the information retained by the context recreates the browser as a full-running browser task.

BROWSER TASK CHILD WINDOWS

Other browser tasks that were called from the main browser task are closed when the browser leaves the page of the parent browser task. When the browser returns to the parent browser task, the child browser tasks are reopened at their last state known to the server.

REFRESH

The browser's refresh action reconstructs the browser task page as if the browser had moved backwards or forwards and returned to the browser task.

Transactions



The Magic Application Server engine lets you handle data transactions of data modifications generated by the browser client task. The transaction used for a browser task is called a Deferred Transaction.

DEFERRED TRANSACTIONS

All the data manipulations reported by the client are kept by the context and are not passed to the database engine until the transaction is completed. If the transaction is rolled back, the retained transaction information is discarded. This manner of handling data manipulation is referred to as a deferred transaction, where the Magic engine defers the physical transaction of the database engine.

This handling of the transaction saves the repeated transmission to the database engine on every particular data manipulation statement, and provides greater scalability of the Application Server capacity.

For more information about Deferred Transactions, refer to the Data Management chapter in the *Magic Reference Guide*.

Constructing a Browser Task

eDeveloper gives you an easy and simple paradigm for defining the browser task logic and its interface handling

The dataview definition and browser task logic are constructed similarly to an Event-driven Magic online task. A browser task differs from an online task in interface definition and changes in the general behavior of the task that derive from the nature of a browser client.

Basic Definitions

Browser Task Type



In the task properties, you define the task type as `Browser`. Setting the task as `Browser` provides you with the appropriate functionality for a browser task both in Toolkit and in Runtime.

Dataview Definition

The dataview of a browser task is defined the same as for an online task.

Main Table

A browser task can display a real dataview from a defined Main Table. Scrolling through the data in runtime involves scrolling through the records fetched from the Main Table. As with any online task, a browser task can display a virtual dataview where no Main Table is defined. Scrolling through a dataview with no defined Main Table means scrolling through an infinite number of virtual records.

The Main Table property is set on the **Properties** tab of the **Task Properties** dialog box.

Selecting the Participating Fields

Using the **Select** operation in the Record Main handler, you can define the fields that will be part of the task's dataview.

You can define any type of Select operation: Real, Virtual, or Parameter.

Extending the Dataview

You can extend your dataview by linking to additional records in other tables using the Link operation.

Note

Any re-computation of a Link operation requires the browser client to address the Application Server engine and fetch the new record. This means that using too many Link operations may be costly in terms of server interaction. Therefore, it is recommended that you limit use of the Link operation. In many cases you may be able to substitute the Link operation with a data control, a selection control like a combo box that displays the value of a range field taken from a table.

For more information about defining the data structure of your application and defining the dataview of a task, refer to the *Magic Reference Guide*

Task Interface

Browser Form



The underlying interface definition of a browser task is an external HTML page. This way the task interface is not bound to the Magic application structure and can be maintained externally and updated without interfering with the application code. This lets you focus on the task logic while, an HTML designer can independently define the interface design of your task. eDeveloper smoothly integrates with the HTML-based elements at any stage if the basic structure of the interface is maintained.

Simply HTML

Every browser task is constructed with at least one main display form of a Browser type. The form definition can be found in the task's Form repository (CTRL+F). One of the main properties of this type of form is the HTML interface file. This property uses the path and file name of the HTML file that will be used to define the task interface.

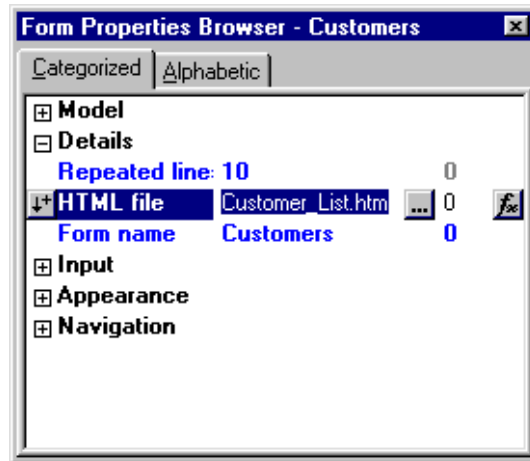


FIGURE 2: The HTML interface file property of the Browser Form.

HTML Controls List

Zooming in from the Browser Form entry displays the HTML Controls list window. In this list you can define each HTML element on the page, assign data to it, and define the properties freely as if the HTML element were a Magic control.

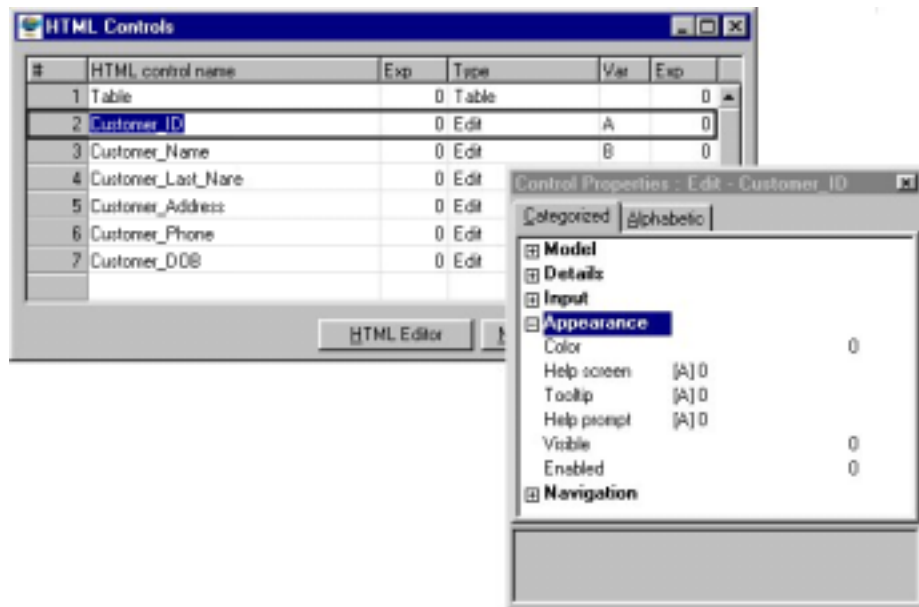


FIGURE 3: The list of HTML controls in the browser task. Each control is assigned data and a set of properties.

The Magic Application Server and browser client module handle these controls with their defined data, once the HTML result page is constructed on the browser client.

New HTML controls can be added to the list by clicking on the **New HTML Tags** button. A new list is displayed showing the remaining HTML elements that have not yet been used by the browser form.

HTML Controls and the HTML Page

The Browser interface definition supports a variety of HTML elements. For each HTML element the property sheet provides a corresponding list of properties. Each HTML Control type corresponds to a defined set of HTML tags.

eDeveloper identifies each HTML element by its given name or ID. This is also a required setting of the HTML control definition. The **HTML Control name** column is the control identifier and should have the same name as the designated HTML element. Therefore, the naming of the HTML controls should keep each control unique.

EDIT

The HTML Edit control corresponds to the `<input type=text>` HTML tag. This control is used as an edit box for displaying and entering data and supports the data of any attribute except for a BLOB.

LIST BOX

The HTML List Box control corresponds to the `<select>` HTML tag. This control is used to display a value from a range of available values and lets the end user change the value to a new value from the given range. This control supports the data of any attribute except for a BLOB.

PUSH BUTTON

The HTML Push button control corresponds to the `<input type=button>` HTML tag. This control is used to allow the end user to click and raise a desired event. This control supports the data of any attribute except for a BLOB.

RADIO BUTTON

The HTML Radio Button control corresponds to the `<input type=radio>` HTML tag. This control is used like a List Box, to display a value from a range of available values, and lets the end user change it to a new value from the given range. However, unlike List Boxes, Radio Buttons can display the entire available range of values at different points of the form. This control supports the data of any attribute except for a BLOB.

COMBO BOX

The HTML Combo Box control corresponds to the `<select>` HTML tag. This control is used, like the list box, to display a value from a range of available values and allows the end user to select a new value from this given range. However, unlike a List Box, the Combo Box shows a single line of value. This control supports the data of any attribute except for a BLOB.

CHECK BOX

The HTML Check Box control corresponds to the `<input type=checkbox>` HTML tag. This control is used to display and define a Boolean value. This control supports the data of logical attributes.

IMAGE

The HTML Image Type control corresponds to the `` HTML tag. This control is used to display an image. The control supports data of alpha and memo attributes. The data of the image control should be the image URL.

HYPERTEXT

The HTML Hypertext control corresponds to the `<a>` or `<div>` HTML tag. This control is used to display any string from a simple static text to a complex HTML sequence. This control is mostly used to display static text and to easily define a dynamic hyperlink for this control. The HTML control supports data with alpha and memo attributes.

TABLE

The HTML Table control corresponds to the `<table>` HTML tag. This control is used to display your data in a scrollable table. The Table control has no data assigned to it, but it is set with various properties that define the way the table will be presented. See the **Line mode** section on page 25 for a description of the behavior of the scrollable table and its properties.

SUBFORM

The HTML Subform control does not correspond to any HTML tag. This control is the logical definition of an additional browser task that handles different parts of the HTML page. See the **Subforms** section on page 27 for a description of a Subform and its behavior.

Browser Form Editing



eDeveloper lets you edit your HTML page using the HTML authoring tool you are most comfortable with. From the Magic Toolkit environment you can easily edit forms using your favorite editor, or you can take advantage of the fact that the HTML interface definition is external to the Magic application and edit it directly using any HTML authoring tool.

Using Your Preferred Authoring Tool

The HTML Controls list lets you zoom even further to edit your HTML page using your preferred editor. By clicking on the HTML Editor button in the HTML Controls list window, Magic invokes the editor as defined in the Magic Environment property under Settings\Environment\Application Sever\Web Authoring Tool. This property defines the path and name of the HTML editor's executable file.

You can edit your HTML page, save the changes and return to the Magic Toolkit.

Drag and Drop

You can drag and drop variables defined in your task onto the HTML page opened by the HTML editor, using the accompanying variables palette. As you drop a variable, Magic creates the HTML input element as defined by the style

settings of the variable, and creates a corresponding entry in the HTML Controls list.

Currently the Drag and Drop option is available only for HTML authoring tools such as FrontPage® and Macromedia® Dreamweaver®.

Control Properties

You can easily set the properties of every HTML element defined as an HTML Control for the browser form definition. Properties whose initial values can be set by the HTML script, such as color or font, are assigned their values by the HTML authoring tool, and this forms part of the HTML script. These properties can only be set by eDeveloper using a dynamic value, an Expression. Other properties, which are not HTML properties by nature, for example Format or Select Program, can be set by eDeveloper using both fixed values and dynamic expressions.

Line Mode Display



You can easily define your data bound controls to be displayed in Line Mode fashion using the HTML table element. This method of display provides the end user with an easy scrollable table of data.

The HTML Table Tag

To produce a scrollable Line-mode display of the data, you should create an HTML Table and position the data-bound controls in one line of the table designated to be the repeated line. You should also uniquely identify the table element by providing an ID for the table tag (e.g. `<table id="Customers_List">`).

Table Control

You should define the table element as an HTML control in the Browser Form HTML Controls list.

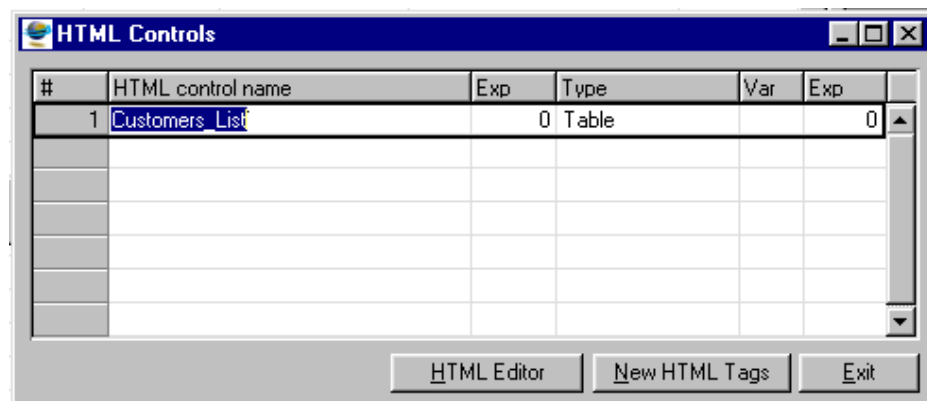


FIGURE 4: The HTML table control defines the HTML table tag that uses the table identifier.

Defining the Repeated Line

The **Detail Line #** property of the HTML Table Control definition specifies the number of the line whose content will be repeated throughout the table.

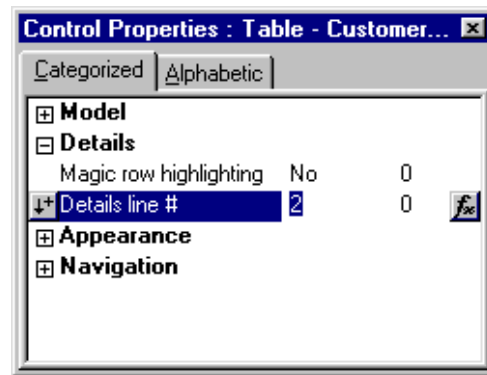


FIGURE 5: The Detail Line # property of the HTML Table Control Entry

Any table row above the designated repeating line is considered to be the table's header. Any data-bound control placed on the header will not be repeated but will show the proper content for each record.

Boundaries of the Repeated Area

An additional important property of the Line Mode setting is the **Repeated Lines** property of the form. This property defines how many times the repeated line should be repeated.

If this property is set with a number greater than zero, at runtime the table control repeats the repeated line the amount of times defined. Any table row defined below the repeated line is considered the table's footer. The repeated lines are added before the remaining table rows. Any data-bound control placed on the footer is not repeated but shows the proper content for each record.

If this property is set to zero, the repeated line is repeated in the remaining rows of the HTML Table control. The number of remaining rows determines the size of the repeated area.

Multiple Tables

The browser task lets you define multiple tables in the same task. All the tables are affected by the navigation simultaneously. This means that by pressing the **Down** key on one table, all other tables of the same task move to the next record.

The repeated area of all the tables is always the same as the number defined in the **Repeated Lines** property of the form. If this property is set to zero, the repeated area size is the same as the smallest amount of remaining rows in the tables.

Static Tables

An HTML table tag that is not defined as a Magic HTML control will not be handled and will remain as a static part of the interface design.

Subforms



eDeveloper lets you easily display data from different tasks in one single HTML interface. Each task will handle its designated part of the HTML interface according to the logic defined for it. The browser client transparently switches from one task to the other according to the location of the insertion point.

When is a Subform Needed?

A Subform is required where the browser client is designed to display extra data on top of the main view, when the extra data is usually made up 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. The Subform control enables the two separate tasks to be displayed on the same HTML interface.

How to Define a Subform

If you want to present two subtasks on the same HTML page follow these general steps:

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 Subform. 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.

Creating a task constructed with Subforms is easy. However, before constructing your Subform, you should read further to learn more about the elements involved.

Subform Control

The Subform control is defined as one of the HTML controls of the main browser task, the task that calls the child task. The Subform control is not represented by any HTML tag on the page. It is, in fact, a logical definition that tells the main task that another task handles part of its HTML page. Since the Subform control need not be related to any HTML element, the name of the Subform control is merely descriptive.

The Subform control has the following properties:

- **Connect to:** Sets the type of task to be called – Program or Subtask.
- **PRG/TSK num:** The number of the called task.
- **Arguments:** Data, variables or expressions, passed as arguments to the called task. The passed arguments are not just required to pass data to the called task. The arguments are the criteria by which the browser client refreshes the view of the child task. Whenever the variable passed as a Subform argument is changed, the view of that Subform is automatically refreshed. Arguments passed as expressions will not refresh the Subform view if their value is changed. If no variable is passed as an argument, the Subform will never be refreshed when scrolling through the main task records.

Note

If the descendant task is a subtask of the main task, the data provided by the main task view can be accessed directly without passing it as arguments. However, the variables responsible for the child task view must still be passed as arguments to serve the Subform refresh criteria.

Nested Subforms

Each main task can have several Subforms. Each Subform can have its own nested Subforms.

Subform Life Cycle

The Magic Application Server automatically opens all the tasks that are defined as Subforms. You do not need to define any Call operations for the Subform task.

BROWSER TASK INITIALIZATION

After executing the Record Prefix of the first record of the main task, the task defined in each Subform definition is opened, thus executing the Task Prefix and Record Prefix of the first record. The tasks are opened according to the order of the Subform controls in the HTML Controls list. Nested Subforms open after the Record Prefix of the first record of the parent Subform.

BROWSER TASK IN ACTION

While scrolling through the main task and changing the content of a variable that is passed as an argument, the Subform dataview is automatically refreshed. Refreshing the Subform task does not invoke the Subform task's Task Prefix or Record Prefix.

Switching from a main task control to a Subform task control switches to the Subform task and executes the Record Prefix of the relevant Subform task record.

BROWSER TERMINATION

On exiting a browser task with Subforms, the Task Suffix of each Subform is executed just before the Task Suffix of its parent task.

Note

A Subform task cannot be terminated by itself. Any attempt to close the Subform task, by a Raise Event operation of the Exit internal event for example, will close the main task as well.

The VCR Toolbar




A VCR control is an image of a VCR panel, which provides the end user with easy navigational functionality for the current dataview.

The VCR Image

The VCR image, shown below, is supplied as a JPG file with your Magic installation, Mgvcr.jpg.





The following list explains the function of the buttons on the VCR panel.


Click  to invoke the Begin Table event.

Click  to invoke the Previous Page event.

Click  to invoke the Previous Record event.

Click  to invoke the Next Record event.

Click  to invoke the Next Page event.

Click  to invoke the End Table event.

Adding the VCR to the Page

Inserting the VCR image into your HTML page is straightforward. You should create an HTML image tag using your HTML authoring tool. The source of the image tag should point to the Mgvcr.jpg, and the name should be MG_VCR. For example:

```
<IMG SRC="/Magic9Scripts/mgvcr.jpg" name="MG_VCR">
```

The Browser client identifies this image by its name and handles it automatically.

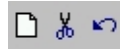
Using a Different VCR image


You can also create your own VCR image and opt not to use the one supplied with eDeveloper. In this case, you should ensure that your image has six consecutive items that correspond to the functions described above.


The Edit Toolbar


You can place an additional image control that provides the end user with a toolbar to create the New Line, Delete Line, and Cancel events.

eDeveloper provides the Mgedit.jpg image, which looks like this:



Click  to invoke the New Line event.

Click  to invoke the Delete Line event.

Click  to invoke the Cancel event.

Adding the MGEDIT Image to the page

You can add the MGEDIT image in the same way as you add the VCR image. Just create an HTML image tag using your HTML authoring tool. The Image tag should point to the Mgedit.jpg file, and its name should be MG_EDIT. For example:

```
<IMG SRC="/Magic9Scripts/mgedit.jpg" name="MG_EDIT">
```

Browser Task Settings



Most browser task settings are the same as the settings of an online task, although some online task settings are irrelevant to a browser task. However, other settings are new and relevant only for a browser task. This section describes some of these settings.

Chunk Size Expression

Task\Properties\Advanced

The **Chunk Size Expression** property defines the number of records to be passed to the browser client upon each request for additional records.

For example, if this property is set to 100, when the task opens the application server will pass the first 100 records to the client. This allows the end user to browse the first 100 records locally. When the end user tries to scroll beyond a given range of records, the client contacts the server and receives an additional batch of 100 records.

Each chunk of records is accumulated on the client as a local cache of records. If the end user goes to the end or beginning of the table, the local cache is cleared and the cache accommodates a single batch of records.

Exit URL

Task\Properties\Advanced

You use the **exit URL** property to define the next URL to be opened in the browser, after a browser task has ended.

Selection Table

Task\Properties\Advanced

As in a regular online task, setting the **Selection Table** property to a true value causes the browser task to behave as a selection table task, executing the **Record Suffix** handler and exiting the task upon invoking the **Select** internal event.

Main Display

Task\Properties\Advanced

As with a regular online task, you can define several browser forms and, using the **Main Display** property, you can provide an expression that evaluates to the form number.

Transaction Mode

Task\Properties\Enhanced

A browser task can only be set with a **Deferred Transaction** or a **Nested Deferred Transaction**.

SQL Command

The **SQL** command is not available for a browser task.

Runtime Behavior and Considerations

Special behavior of the browser task in Runtime

Due to constraints imposed by the browser, certain runtime behavior needs to be taken into consideration before defining your browser task logic.

Call Operation

Calling another Browser Task



You can call from one browser task to another using the Call Program operation by creating a Call operation and selecting the number of the browser task you want to call. You can also pass arguments to the calling browser task.

Modal Browser Window

In the Browser Form properties, you can set the window to Modal. When a browser task calls another browser task that has been set as Modal, the flow of the calling task is halted until the called task is completed. While the called task is running, the focus cannot be set back to the calling task. This is similar to the existing interaction between regular online programs.

When a Browser Form task is not set as modal, the flow of the calling task is not interrupted when the called task window is opened, and the focus can be moved freely from the called task to the calling task.

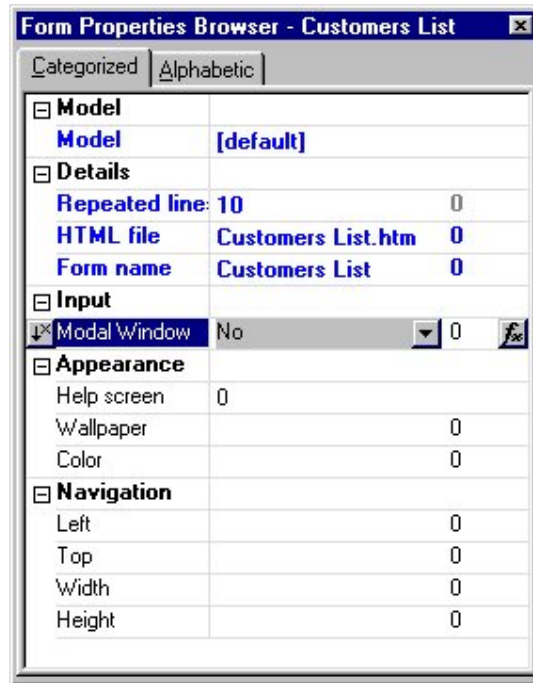


FIGURE 6: The Browser Form property sheet

Note

The main form task determines the modality of the task window. If you want the Call operation to determine the modality of the called task, you can define a relevant parameter to set the modality and have the Call program pass the desired value to this parameter.

Task Initialization

It is important to remember that the execution of the Task Prefix, the initialization phase of the task, is executed on the server side as a whole. This means that in the initialization phase any Call operation to a browser task will be subjected to the following rules:

- All Call operations will be executed at the designated time, but the Call task's window is opened on the client side after completion of the Task Prefix. This means that all Call operations will take place after the other operations are executed.
- The Modality setting of the Browser Form will be ignored. All Call operations to a browser task will be opened as if they were set to Modal=No.

Task Termination

Browser tasks called from the Task Suffix will not be displayed on the browser because the calling task has already been terminated. A browser task called from the Task Suffix level handler will be run and immediately closed on the server side; that is the Task Prefix and Suffix level handlers of the task will be executed automatically on the server side.

Calling a Batch Task



You can easily call any batch task from your browser task to create any required batch processing.

Synchronous Call

A call operation to a batch task is always synchronous. This means that the client waits until the batch task is complete.

The Call operation to a batch task will be synchronous even if it is invoked from a handler whose event was raised asynchronously.

Start/End Execution

The Start and Stop Execution dialog boxes are not supported for batch tasks that are called from a browser task.

Calling Another Browser Task from a Batch Task

Another browser task cannot be called from a batch task.

Calling an Online Task



An online task cannot be called from a browser task.

Verify Operation

Client-side Messages



You can use the Verify operation to provide the end user with messages throughout the application.

Display

The verify message can be displayed in the following two ways:

- **Box** – A dialog box is displayed on the browser. The task flow will not proceed until this box is closed.

- **Message** – The browser’s status bar displays a message. The task flow continues while the message is displayed, and no user interaction is required.

Task Initialization

As with a Call operation to a browser task, the Verify operation, defined as part of a server-side cycle, will appear on the client after the server-side cycle is completed. This means that any Verify operation in the initialization phase will be executed on the client side after completion of the Task Prefix.

Task Termination

A Verify operation in the Task Suffix will not be displayed on the browser because the calling task has already been terminated.

System Event Handlers

Browser Internal Handling



A running browser window has its own internal handling for specific system events such as keyboard commands.

For example, the F4 key opens the Address box. The browser client module cannot prevent the browser window from handling these system events. Although you can create your own handlers for key combinations, the browser eventually executes its own handling for them. This means that the propagate property of the handler of these system events will be ignored.

Handled System Events

The following are the key combinations that are handled by the browser:

F1, F3, F4, F5, F6, F10, F11, ALT+F4, CTRL+F1, CTRL +F4, CTRL +F5, CTRL +F6, CTRL +F10, CTRL +A, CTRL +C, CTRL +F, CTRL +O, CTRL +P, CTRL +V, CTRL +X.

Error Handling

Exceptions in the Browser Task



Errors that occur in a running browser task are handled in the same way as they are handled by other tasks in Magic. Exceptions to the error handling rules specific to a browser task are described below.

For more information about basic Error Handling, see the Error Handling chapter of the *Magic Reference Guide*.

Call and Verify Operations

The entire handler of an error event is executed on the server. This means that operations such as a Call operation to another browser task, or a Verify operation, are reflected on the browser client after completion of the error handler. In addition, the Modality of a browser window called from an error handler will be ignored, and will always be considered as Modal=No.

ROLLBACK Function

The **ROLLBACK** option that displays a confirmation dialog box cannot be executed on the browser client. This means that the first parameter of the **ROLLBACK** function will be ignored.

Closing the Browser Window



Closing the browser window by clicking on the system menu **X** button, by choosing **Close** from the **File** menu, or by pressing **ALT+F4** are not considered proper ways to terminate a browser task. The browser task behavior in these cases is described below:

- **No handlers are executed:** The Control Verification, Suffix handlers, Record Suffix, and Task Suffix are not executed.
- **Open Transaction:** An open transaction will not be committed. An open transaction may be the task transaction, if the transaction was set to be of a task, or the last record transaction, if the transaction was set to be of a record. When the transaction relates to a record, the records that have been updated and exited prior to closing the window will already be committed in the database.
- **Descendant Tasks:** Descendant browser tasks that were called from a task whose window is closed will also automatically be closed.
- **Preventing the window from closing:** Due to the nature of the browser, the Magic browser client module cannot halt the browser in its closing phase. This means that there is no way to prevent the closing phase of the window once the end user has invoked it.

Summary

eDeveloper is your tool for assembling high-level Interactive Web applications.

Magic eDeveloper's browser-based functionality provides an easy way to create fully interactive large scale applications on a browser client. This means that such an application can be operational over any local or wide-area network.

Easy Programming

Unified Concept



The Interactive Web Application Development and deployment paradigm has been designed to compliment Magic's inherent application development capabilities. The concept behind each application task is identical to conventional Magic programming providing an easy transition from conventional client/server application development to developing browser-based applications.

Simplified Paradigm



eDeveloper simplifies the development process by automatically handling any required Runtime functionality for the browser client, such as event handling, component display, and data manipulation. In addition, eDeveloper handles any required Runtime functionality of the supporting application server.

The Tool



eDeveloper is *the* tool for creating your own browser-based applications.