

# **Informix Web DataBlade Module**

## User's Guide

Version 3.3  
May 1997  
Part No. 000-3947

Published by INFORMIX® Press

Informix Software, Inc.  
4100 Bohannon Drive  
Menlo Park, CA 94025-1032

Copyright © 1981-1997 by Informix Software, Inc. or its subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

INFORMIX®; Illustra™; DataBlade®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

#### ACKNOWLEDGMENTS

Documentation Team: Sandra Farkas

Contributors: Kimberly Bostrom, Cal Collier, John Gaffney, Daniel Howard, Mark Mears, Dorothy Moore, Karin Moore, Bob Nowacki, Sean Riley, Juliet Shackell, Martin Siegenthaler, Scott Stark, Jan Strother, Tony Sweeney

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

#### RESTRICTED RIGHTS/SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

# Table of Contents

## Introduction

About This Manual . . . . .	3
Organization of This Manual . . . . .	3
Types of Users . . . . .	5
Software Dependencies . . . . .	5
Documentation Conventions . . . . .	5
Typographical Conventions . . . . .	6
Icon Conventions . . . . .	6
Screen-Illustration Conventions . . . . .	8
Additional Documentation . . . . .	8
Printed Documentation . . . . .	8
On-Line Documentation . . . . .	9
Informix Welcomes Your Comments . . . . .	10

## Chapter 1 Overview of the Web DataBlade Module

Product Architecture . . . . .	1-3
Product Features . . . . .	1-5

## Chapter 2 Setting Up Your Webdriver Environment

Configuring Webdriver . . . . .	2-3
Invoking AppPages . . . . .	2-7
Implementing AppPage-Level Security . . . . .	2-7
Linking AppPages . . . . .	2-10
ANCHOR Tag . . . . .	2-10
FORM Tag . . . . .	2-10
Retrieving Large Objects . . . . .	2-13
Retrieving Large Objects by Name . . . . .	2-13
Retrieving Large Objects by LO Handle . . . . .	2-15

<b>Chapter 3</b>	<b>Using AppPage Builder</b>	
	Invoking AppPage Builder . . . . .	3-3
	Creating Web Applications in AppPage Builder . . . . .	3-6
	Multimedia Content Support . . . . .	3-6
	Administration Features . . . . .	3-7
<b>Chapter 4</b>	<b>Using Variables and Tags in AppPages</b>	
	Web DataBlade Module Variables . . . . .	4-4
	MISQL Tag . . . . .	4-5
	Using System Variables to Format the SQL Results . . . . .	4-7
	MAXROWS Attribute . . . . .	4-14
	MIVAR Tag . . . . .	4-15
	NAME and DEFAULT Attributes . . . . .	4-16
	MIBLOCK Tag . . . . .	4-17
	COND Attribute . . . . .	4-18
	MIERROR Tag . . . . .	4-19
	TAG Attribute . . . . .	4-20
	ERR Attribute . . . . .	4-21
	Creating a Generic Error Handler . . . . .	4-22
	Handling Error Conditions . . . . .	4-22
	Processing Errors with Webdriver . . . . .	4-25
	Special Characters in Web DataBlade Module Tags . . . . .	4-27
	Special HTML Characters . . . . .	4-28
	Special Formatting Characters . . . . .	4-28
<b>Chapter 5</b>	<b>Using Variable Processing Functions in AppPages</b>	
	Variable Processing Functions . . . . .	5-3
	Using Variable Expressions in AppPages . . . . .	5-8
	Using Arithmetic Functions in Variable Expressions. . . . .	5-8
	Using SEPARATE and REPLACE in Variable Expressions. . . . .	5-9
	Using Variable Expressions to Format Output Conditionally. . . . .	5-12
	Special Characters in Variable Expressions . . . . .	5-16
<b>Chapter 6</b>	<b>Using Dynamic Tags in AppPages</b>	
	Invoking Dynamic Tags in AppPages . . . . .	6-3
	Using System Dynamic Tags . . . . .	6-4
	CHECKBOXLIST . . . . .	6-4
	RADIOLIST . . . . .	6-7
	SELECTLIST . . . . .	6-10
	Creating User Dynamic Tags . . . . .	6-13

	Special Characters in Dynamic Tags . . . . .	6-15
<b>Chapter 7</b>	<b>Using Server Functions in AppPages</b>	
	WebExplode . . . . .	7-4
	WebLint . . . . .	7-7
	WebRelease . . . . .	7-10
	WebUnHTML . . . . .	7-12
	WebURLDecode . . . . .	7-14
	WebURLEncode . . . . .	7-15
<b>Chapter 8</b>	<b>Using Advanced Webdriver Features</b>	
	Adding HTTP Headers to AppPages . . . . .	8-3
	Retrieving Non-HTML Pages . . . . .	8-4
	Using Cookies . . . . .	8-4
	Uploading Client Files . . . . .	8-7
	Passing Image Map Coordinates . . . . .	8-11
	IMG Tag . . . . .	8-12
	FORM Tag . . . . .	8-13
	Configuring Webdriver Connection Settings . . . . .	8-15
<b>Chapter 9</b>	<b>Caching Large Objects Using Webdriver</b>	
	Configuring Webdriver Large Object Caching . . . . .	9-4
	Configuring Webdriver Large Object Cache Cleanup . . . . .	9-6
<b>Chapter 10</b>	<b>Using NSAPI Webdriver Features</b>	
	NSAPI Webdriver Architecture . . . . .	10-3
	Configuring the Netscape Web Server . . . . .	10-3
	Configuring Startup Configuration Information . . . . .	10-4
	Configuring Object Management. . . . .	10-4
	Starting and Stopping the NSAPI Webdriver Processes . . . . .	10-6
	Invoking the NSAPI Webdriver . . . . .	10-6
	Implementing Security with the NSAPI Webdriver . . . . .	10-7
	Passing Image Map Coordinates with the NSAPI Webdriver . . . . .	10-10
	Error Logging with the NSAPI Webdriver . . . . .	10-10

<b>Appendix A</b>	<b>Debugging Web DataBlade Module Applications</b>
<b>Appendix B</b>	<b>AppPage Builder Schema</b>
<b>Appendix C</b>	<b>Web DataBlade Module Variables</b>
	<b>Index</b>

# Introduction

About This Manual . . . . .	3
Organization of This Manual . . . . .	3
Types of Users . . . . .	5
Software Dependencies . . . . .	5
Documentation Conventions . . . . .	5
Typographical Conventions . . . . .	6
Icon Conventions . . . . .	6
Comment Icons . . . . .	7
Product and Platform Icons . . . . .	7
Screen-Illustration Conventions . . . . .	8
Additional Documentation . . . . .	8
Printed Documentation . . . . .	8
On-Line Documentation. . . . .	9
Informix Welcomes Your Comments. . . . .	10





# T

his chapter introduces the *Informix Web DataBlade Module User's Guide*. Read this chapter for an overview of the information provided in this manual and for an understanding of the conventions used throughout.

---

## About This Manual

The *Informix Web DataBlade Module User's Guide* explains how to use the Informix Web DataBlade module to create Web applications that dynamically retrieve data from the INFORMIX-Universal Server database.

This manual provides information about the features that are provided with the Web DataBlade module to assist you in developing Web-enabled database applications.

This section discusses the organization of the manual, the intended audience, and the associated software products that you must have to develop applications using the Web DataBlade module.

## Organization of This Manual

This manual includes the following chapters:

- This Introduction provides an overview of the manual, describes the documentation conventions used, and explains the generic style of this documentation.
- [Chapter 1, “Overview of the Web DataBlade Module,”](#) provides an overview of the architecture and features of the Web DataBlade module.

- [Chapter 2, “Setting Up Your Webdriver Environment,”](#) provides configuration information for the Webdriver interface to AppPages and multimedia content residing in the INFORMIX-Universal Server database.
- [Chapter 3, “Using AppPage Builder,”](#) describes how to create and maintain Web DataBlade module applications using AppPage Builder.
- [Chapter 4, “Using Variables and Tags in AppPages,”](#) describes how to use Web DataBlade module variables and tags to create Web-enabled applications.
- [Chapter 5, “Using Variable Processing Functions in AppPages,”](#) describes how to use variable processing functions to create variable expressions within AppPages.
- [Chapter 6, “Using Dynamic Tags in AppPages,”](#) describes how to use dynamic tags to share AppPage segments among multiple AppPages.
- [Chapter 7, “Using Server Functions in AppPages,”](#) describes the **WebExplode** function and additional server functions you can use to simplify AppPage design.
- [Chapter 8, “Using Advanced Webdriver Features,”](#) describes Webdriver features including adding HTTP headers to your AppPages, uploading client files, passing image map coordinates, and configuring Webdriver connection settings.
- [Chapter 9, “Caching Large Objects Using Webdriver,”](#) describes how to store and retrieve large objects in the disk cache for Webdriver.
- [Chapter 10, “Using NSAPI Webdriver Features,”](#) describes the configuration and features of the NSAPI implementation of Webdriver.
- [Appendix 0](#) describes debugging techniques for the Web DataBlade module.
- [Appendix B](#) describes the schema for AppPage Builder.
- [Appendix C](#) lists all Webdriver and **WebExplode** function variables.

## Types of Users

This guide is written for Web application designers who are familiar with HTML (including tables and forms), SQL, and database installation and system administration.

## Software Dependencies

To use this product you must be using INFORMIX-Universal Server as your database server. Check the on-line documentation for specific version compatibility. The on-line documentation also lists the Web servers that have been certified for this release.

---

## Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Screen-illustration conventions

## Typographical Conventions

This manual uses the following standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics. A word in italics also represents a value that you must supply, such as a database, file, or program name.
<b>boldface</b>	Identifiers (names of classes, objects, constants, events, functions, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
<code>monospace</code>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of product- or platform-specific information.






***Tip:** When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.*

## Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

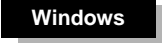

## Comment Icons

Comment icons identify warnings, important notes, or tips. This information is always displayed in italics.

Icon	Description
	The <i>warning</i> icon identifies vital instructions, cautions, or critical information.
	The <i>important</i> icon identifies significant information about the feature or operation that is being described.
	The <i>tip</i> icon identifies additional details or shortcuts for the functionality that is being described.

## Product and Platform Icons

Product and platform icons identify paragraphs that contain product-specific or platform-specific information.

Icon	Description
	Identifies information that is specific to the Windows environment.
	Identifies information that is specific to the UNIX environment.

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the product- or platform-specific information.

## Screen-Illustration Conventions

The illustrations in this manual represent a generic rendition of various windowing environments. The details of dialog boxes, controls, and windows have been deleted or redesigned to provide this generic look. Therefore, the illustrations in this manual depict Web browser output a little differently than the way it appears on your screen.

---

## Additional Documentation

The Web DataBlade module documentation set includes printed documentation and on-line documentation.

This section describes additional documentation available from Informix:

- Printed documentation
- On-line documentation

## Printed Documentation

The following related Informix documents complement the information in this manual:

- If you have never used Structured Query Language (SQL), read the *Informix Guide to SQL: Tutorial*. It provides a tutorial on SQL as it is implemented by Informix products. It also describes the fundamental ideas and terminology for planning and implementing a relational database.
- A companion volume to the *Tutorial*, the *Informix Guide to SQL: Reference*, includes details of the Informix system catalog tables, describes Informix and common environment variables that you should set, and describes the column data types that Informix database servers support.

- An additional companion volume to the *Reference*, the [Informix Guide to SQL: Syntax](#), provides a detailed description of all the SQL statements supported by Informix products. This guide also provides a detailed description of Stored Procedure Language (SPL) statements.
- The *SQL Quick Syntax Guide* contains syntax diagrams for the SQL statements and segments described in this manual.
- The *UNIX Products Installation Guide* contains installation instructions for your particular release to ensure that your Informix product is properly set up before you begin to work with it.
- The [INFORMIX-Universal Server Administrator's Guide](#) will help you understand, install, configure, and use INFORMIX-Universal Server and change its characteristics to meet your needs.
- The [DataBlade Developers Kit User's Guide](#) describes the architecture of the INFORMIX-Universal Server database system and how DataBlade modules work with INFORMIX-Universal Server.
- The [DB-Access User Manual](#) describes how to invoke the DB-Access utility to access, modify, and retrieve information from Informix database servers.

## On-Line Documentation

In addition to the Informix set of manuals, the following on-line files, located in the `$INFORMIXDIR/extend/web.3.30.UC1` directory, supplement the information in this manual.

On-Line File	Purpose
webdc330.txt	Describes features that are not covered in the manual or that have been modified since publication.
webrl330.txt	Describes any special actions that are required to configure and use the Web DataBlade module on your computer. This file also describes new features and feature differences from earlier versions of the Web DataBlade module and how these differences might affect current products. Additionally, this file contains information about any bugs and their workarounds.
webmc330.txt	Describes platform-specific information regarding the release.

Please examine these files because they contain vital information about product and performance issues.

---

## **Informix Welcomes Your Comments**

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Please include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.  
Technical Publications  
300 Lakeside Dr., Suite 2700  
Oakland, CA 94612

If you prefer to send electronic mail, our address is:

`doc@informix.com`

Or, send a facsimile to Technical Publications at:

415-926-6571

We appreciate your feedback.



---

# Overview of the Web DataBlade Module

Product Architecture . . . . . 1-3

Product Features . . . . . 1-5



**T**he Web DataBlade module enables you to create Web applications that incorporate data retrieved dynamically from the INFORMIX-Universal Server database.

In typical Web database applications, most of the logic is in gateway application code written in Perl, Tcl, or C. This *Common Gateway Interface (CGI)* application connects to a database, builds and executes SQL statements, and formats the results.

Using the Web DataBlade module, you need not develop a CGI application to dynamically access database data. Instead, you create HTML pages that include Web DataBlade module tags and functions that dynamically execute the SQL statements you specify and format the results. These pages are called *Application Pages (AppPages)*. The types of data you retrieve can include traditional data types, as well as HTML, image, audio, and video data.

---

## Product Architecture

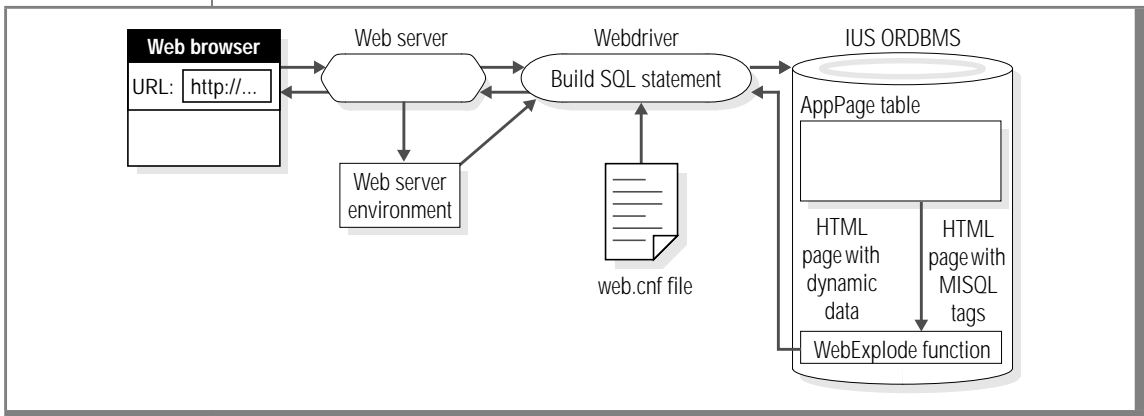
The Web DataBlade module consists of three main components:

- **Webdriver.** As an INFORMIX-Universal Server client application, Webdriver builds the SQL queries that execute the **WebExplode** function to retrieve AppPages from the INFORMIX-Universal Server database. Webdriver returns the HTML resulting from calls to **WebExplode** to the Web server. You can choose either the *Netscape Server API (NSAPI)* or the CGI implementation of Webdriver as the database interface to your Web server, depending on your needs.

- **WebExplode function.** The **WebExplode** function builds dynamic HTML pages based on data stored in the INFORMIX-Universal Server database. **WebExplode** parses AppPages that contain Web DataBlade module tags within HTML and dynamically builds and executes the SQL statements and processing instructions embedded in the Web DataBlade module tags. **WebExplode** formats the results of these SQL statements and processing instructions, and returns the resulting HTML page to the client application (usually Webdriver). The SQL statements and processing instructions are specified using SGML-compliant processing tags.
- **Web DataBlade module tags and attributes.** The Web DataBlade module includes its own built-in set of SGML-compliant tags and attributes that enable SQL statements to be executed dynamically within AppPages.

The following diagram illustrates the architecture of the Web DataBlade module.

**Figure 1-1**  
*Web DataBlade Module Architecture*



When a URL contains a Webdriver request, the Web browser makes a request to the Web server to invoke Webdriver. Based on configuration information, Webdriver composes an SQL statement to retrieve the requested AppPage and then executes the **WebExplode** function. **WebExplode** retrieves the requested AppPage from the Web application table (stored in the INFORMIX-Universal Server database), executes the SQL statements within that AppPage by expanding the Web DataBlade module tags, and formats the results. **WebExplode** returns the resulting HTML to Webdriver. Webdriver returns the HTML to the Web server, which returns the HTML to be rendered by the Web browser.

Webdriver also enables you to retrieve large objects, such as images, directly from the INFORMIX-Universal Server database when you specify a path that identifies a large object stored in the database.

---

## Product Features

The Web DataBlade module provides the following features:

- Web DataBlade module tags enable you to:
  - embed SQL statements directly within AppPages.
  - handle errors within AppPages.
  - execute statements conditionally within AppPages.
  - manipulate variables within AppPages using variable processing functions.
  - use other advanced query processing and formatting techniques.
- Web DataBlade module *dynamic tags* allow you to reuse existing AppPage segments to simplify the construction and maintenance of your Web applications. The Web DataBlade module provides *system dynamic tags* that you can use. You can also create *user dynamic tags*.
- Webdriver allows you to customize Web applications using information from its configuration file, the Web server environment, URLs, HTML forms, and your own Web application variables, without additional CGI programming.

- *AppPage Builder (APB)* allows you to create and maintain Web DataBlade module applications and manage multimedia database content.
- The NSAPI implementation of Webdriver allows you to use Netscape Web server security features and eliminate CGI process overhead.

---

# Setting Up Your Webdriver Environment

Configuring Webdriver . . . . .	2-3
Invoking AppPages . . . . .	2-7
Implementing AppPage-Level Security . . . . .	2-7
Linking AppPages . . . . .	2-10
ANCHOR Tag . . . . .	2-10
FORM Tag . . . . .	2-10
Retrieving Large Objects . . . . .	2-13
Retrieving Large Objects by Name . . . . .	2-13
Retrieving Large Objects by LO Handle . . . . .	2-15





# W

ebdriver provides an environment for creating customized Web applications using Web DataBlade module tags and functions. Webdriver provides the interface to the INFORMIX-Universal Server database, from which both the AppPages and the results of SQL statements are dynamically retrieved. To design a Web application, you create AppPages and specify the flow between these AppPages. Webdriver allows you to customize and control your Web application environment.

---

## Configuring Webdriver

A Web server calls Webdriver through a CGI or NSAPI interface to retrieve AppPages from the INFORMIX-Universal Server database. Webdriver obtains configuration information about the Web application environment from the following sources:

- The **web.cnf** file
- The Web browser
- The Web server environment

Webdriver obtains configuration information about Web applications from its own set of environment variables. *Mandatory* environment variables are variables that must be set to invoke a particular Webdriver feature. The default settings for Webdriver environment variables are specified in the Webdriver configuration file, **web.cnf**, which resides in the same directory as the Webdriver executable file. You can modify the values of Webdriver environment variables in a URL that invokes Webdriver or within AppPages.

In addition to setting Webdriver environment variables, you can set your own Web application variables in the **web.cnf** file and modify the values of these variables within a URL or within your AppPages. You can also obtain the values of Web server environment variables within AppPages. For more information on Web DataBlade module variables, see [“Web DataBlade Module Variables” on page 4-4](#).

For each request from a Web server, Webdriver determines which AppPage to retrieve from the INFORMIX-Universal Server database based on Webdriver environment variables.

The following table lists Webdriver environment variables that identify a particular AppPage to retrieve when Webdriver is invoked.

Variable	Mandatory?	Content
MItab	Yes	Name of the database table in which the AppPages for the Web application are stored.
MIcol	Yes	Name of the HTML column that contains the AppPage in the Web application table.
MInam	Yes	Name of the VARCHAR column that identifies the appropriate row of the Web application table.
MIval	Yes	Value to check against the MInam column to identify the row you want to retrieve in the Web application table.
WEB_HOME	No	URL-mapped path to Webdriver.
MI_USER	Yes	Name of the database user.
MI_PASSWORD	Yes	Password for the user specified by MI_USER.
MI_DATABASE	Yes	Name of the database to connect to when the user makes a Webdriver request.

Specify variable pairs to set default values for Webdriver and Web application variables in the **web.cnf** file. Use a tab at the beginning of a line to indicate a variable pair. These pairs must be separated by white space. Leading and trailing blanks are trimmed from names and values. Indicate that the variable assignment in the **web.cnf** file can be overridden by placing a question mark (?) immediately following the name of the variable. You can then override the variable assignment specified in the **web.cnf** file in an AppPage, HTML form, or URL. Indicate comments with a number sign (#). Lines beginning with # or # preceded by white space are ignored.

The following is an example **web.cnf** file:

```
# web.cnf file
    INFORMIXDIR      /local1/web/sqlldist
    INFORMIXSERVER   IUS9_web
    MI_USER          webdba
    MI_PASSWORD      secret
    MI_DATABASE      webdb
    MIval?           apb
    MInam            ID
    MIcol            object
    MItab            webPages
    WEB_HOME         /cgi-bin/webdriver
```

The **webPages** table, specified by the MItab variable in the preceding **web.cnf** file, contains the AppPages that drive one or more Web applications. When you install the Web DataBlade module, the **webPages** table, described in [Appendix B](#), is created.

The following is a sample of the contents of the **webPages** table.

ID	description	object	....
apb	APB Main Menu	<HTML>....<?MISQL>....	....
....	....	....	....
sql_prog	Issue SQL commands	<HTML>....<?MISQL>....	....
....	....	....	....

The **ID** column (MInam) contains a unique identifier for each row of the **webPages** table. The **object** column (MIcol) contains the AppPage. The **apb** value (MIval) specifies the unique value to determine which row to retrieve. Based on these variables, Webdriver generates an SQL statement to retrieve the AppPage. Webdriver builds the following call to the **WebExplode** function:

```
SELECT WebExplode($MIcol,'ENVIRONMENT') FROM $MITab WHERE $MInam = '$MIval';
```

In the preceding example, the call to **WebExplode** becomes:

```
SELECT WebExplode(object,'ENVIRONMENT') FROM webPages WHERE ID = 'apb';
```

**ENVIRONMENT** in the preceding SQL statement consists of *name/value pairs*. Name/value pairs are the name of a variable and its value, separated by ampersands (for example, `name1=value1&name2=value2...`).

**ENVIRONMENT** includes the following values:

- Web server environment variables, such as `SERVER_NAME`, `SERVER_SOFTWARE`, and `HTTP_USER_AGENT`
- The `QUERY_STRING` environment variable, which is the portion of a URL following a `?`, for example, `MIval=apb` in the URL  
`http://myhost/cgi-bin/webdriver?MIval=apb`
- The `PATH_INFO` environment variable, which is the portion of a URL consisting of name/value pairs following the pathname and preceding a `?`, for example, `MImap=on` in the URL  
`http://myhost/cgi-bin/webdriver/MImap=on?100,32`
- An HTML form (if the request is processing a form)

These environment variables are accessible within the Web DataBlade module tags in AppPages.

---

## Invoking AppPages

There are two approaches to invoking AppPages using Webdriver URLs:

- Specify all Webdriver environment variables on each request.
- Set default Webdriver environment variable values in the **web.cnf** file and pass only specific information on each request.

In the first approach, URLs take the following form:

```
http://myhost:port/cgi-bin/webdriver?  
MITab=webPages&Micol=object&Minam=ID&Mival=apb
```

In the second approach, the MITab, Micol, and Minam environment variables are set in the **web.cnf** file, and URLs take the following form:

```
http://myhost:port/cgi-bin/webdriver?Mival=apb
```

The second example uses the default values for all of the variables except Mival. For most requests, the defaults can be used, thus keeping the URLs simple. You can override the environment variables with individual requests whenever necessary, providing you have specified that the variables can be overridden. See [“Configuring Webdriver” on page 2-3](#) for information on Webdriver environment variables.

---

## Implementing AppPage-Level Security

If you are using the NSAPI implementation of Webdriver, see [“Implementing Security with the NSAPI Webdriver” on page 10-7](#) for information on using Netscape Web server security. If you are using a Web server other than Netscape, you can use Webdriver to do AppPage-level authorization.

The following table lists Webdriver environment variables that configure AppPage-level authorization.

Variable	Mandatory?	Content
MIpagelevel	Yes	Name of the INTEGER column of the AppPage table (MITab) that contains the access level of the AppPage.
MI_WEBACCESSLEVEL	Yes	Access level of all users for a particular <b>web.cnf</b> file.
MI_WEBREDIRECT	No	URL to redirect users to if they do not have access to the AppPage they attempt to retrieve.



**Important:** If *MIpagelevel* is not set in the **web.cnf** file, no authorization check is performed.

The following is an example **web.cnf** file with AppPage-level authorization settings:

```
# web.cnf file
INFORMIXDIR          /local1/web/sqlldist
INFORMIXSERVER       IUS9_web
MI_USER              webdba
MI_PASSWORD          secret
MI_DATABASE          webdb
Mival?               apb
MInam                ID
Micol                object
MITab                webPages
WEB_HOME              /cgi-bin/webdriver
# Webdriver AppPage-level authorization variables
MIpagelevel          read_level
MI_WEBACCESSLEVEL    1
MI_WEBREDIRECT        http://cgi-bin/errors
```

If the access level of the retrieved AppPage is less than or equal to the value of the **MI\_WEBACCESSLEVEL** variable, the user can see the AppPage. **MI\_WEBACCESSLEVEL** cannot be overridden in a URL.

If authorization for an AppPage is denied because the value of MI\_WEBACCESSLEVEL is less than the access level of the retrieved AppPage, you can redirect the browser to another URL by setting the MI\_WEBREDIRECT variable to that URL. If MI\_WEBREDIRECT is not set and a user attempts to access an AppPage with an access level higher than the value of MI\_WEBACCESSLEVEL, an access error is raised.

If you are using Web server authentication, the Web server stores the name of the remote user in the REMOTE\_USER Web server environment variable. You can access the value of REMOTE\_USER within your AppPages. Webdriver does not allow this variable to be overridden in the URL.

---

## Linking AppPages

There are two methods for linking AppPages within a Web DataBlade module application. You can link AppPages by:

- using the ANCHOR tag to generate dynamic links.
- creating a hidden INPUT button in an HTML form.

### ANCHOR Tag

Use the HREF attribute of the ANCHOR tag to link AppPages in your Web application to each other. Use the value of the WEB\_HOME environment variable and the MIVAR Web DataBlade module tag to dynamically generate these links. Set WEB\_HOME to the URL-mapped path to the Webdriver executable file. The following ANCHOR tag is a link to the URL for AppPage Builder:

```
<A HREF=<?MIVAR>$WEB_HOME<?/MIVAR>?Mival=apb></A>
```

The MIVAR tag enables the value of the WEB\_HOME variable to be dynamically generated. Therefore, if WEB\_HOME is set to /cgi-bin/webdriver, the resulting URL is:

```
<A HREF=/cgi-bin/webdriver?Mival=apb></A>
```

For more information about the WEB\_HOME variable, see [“Configuring Webdriver” on page 2-3](#). For more information about the MIVAR tag, see [“MIVAR Tag” on page 4-15](#).

### FORM Tag

Another way to link the AppPages in your Web application is to create a hidden INPUT button in an HTML form. The FORM tag for the button must specify WEB\_HOME as the action, for example:

```
<FORM METHOD=POST ACTION=<?MIVAR>$WEB_HOME<?/MIVAR>>
```

When you submit the form, the following INPUT tag causes the **display\_table** AppPage to be invoked:

```
<INPUT TYPE=HIDDEN NAME=Mival VALUE=display_table>
```



The following **select\_table** AppPage allows you to type a table name into the **table\_name** text-entry field, then submit the form. When you submit the form, the **display\_table** AppPage is invoked, a SELECT from the specified table is performed, and the output is displayed. The following is the **select\_table** AppPage:

```
<HTML>
<HEAD><TITLE>Select from Table</TITLE></HEAD>
<BODY>
<FORM METHOD=POST ACTION=<?MIVAR>$WEB_HOME<?/MIVAR>>
<?MIVAR NAME=$table_name><?/MIVAR>
Select from table: <HR>
<?MIVAR>
<INPUT TYPE=INPUT SIZE=40 NAME=table_name VALUE=$table_name>
<?/MIVAR>
<INPUT TYPE=SUBMIT VALUE=Select>
<INPUT TYPE=HIDDEN NAME=Mival VALUE=display_table>
</FORM>
</BODY>
</HTML>
```

For information on the MIVAR tag used in this example, see [“MIVAR Tag” on page 4-15](#).

The following is sample Web browser output for the **select\_table** AppPage.

**Figure 2-1**  
*Select from Table*

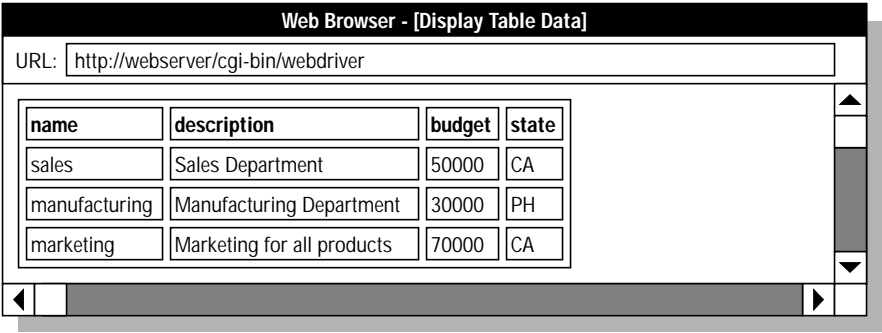
When you submit the form displayed by **select\_table**, the **display\_table** AppPage is invoked. The **display\_table** AppPage retrieves the column headers for the specified table in the submitted **table\_name** field from the **syscolumns** and **systables** system catalog tables and displays the column headers and the rows of the specified table within an HTML table. The following is the **display\_table** AppPage:

```
<HTML>
<HEAD><TITLE>Display Table Data</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<TR>
<?MISQL SQL="select a.colname, colno from syscolumns
a, systables b where a.tabid = b.tabid and
b.tabname = trim('$table_name')
order by colno;"><TH>$1</TH>
<?/MISQL>
</TR>
<?MISQL SQL="select * from $table_name;">
<TR> { <TD>$*</TD> } </TR><?/MISQL>
</TABLE>
</BODY>
</HTML>
```

For more information on the MISQL tag used in this example, see “[MISQL Tag](#)” on page 4-5.

The following is sample Web browser output for the **display\_table** AppPage.

*Figure 2-2  
Display Table Data*



## Retrieving Large Objects

Webdriver gives you two methods for retrieving large objects from the database. You can retrieve large objects by:

- setting Webdriver variables to uniquely identify an object within a table.
- obtaining large object handles (LO handles) dynamically from a SELECT statement.

## Retrieving Large Objects by Name

You can use Webdriver to retrieve a large object from a table in the database by specifying the Webdriver variables that uniquely identify the object. The following table lists the Webdriver environment variables used to identify a large object.

Variable	Mandatory?	Content
MItabObj	Yes	Name of the database table in which Web application large objects are stored.
MicolObj	Yes	Name of the BLOB column that contains large objects in the MItabObj table.
MInamObj	Yes	Name of the VARCHAR column that identifies the appropriate row of the MItabObj table.
MlvalObj	Yes	Value to check against the MInamObj column to identify the row you want to retrieve in the MItabObj table.
MltypeObj	Yes	MIME type and subtype used to export the large object.

The following is an example **web.cnf** file with default values for retrieving large objects by name:

```
# web.cnf file
INFORMIXDIR           /local1/web/sqlldist
INFORMIXSERVER        IUS9_web
MI_USER               webdba
MI_PASSWORD           secret
MI_DATABASE           webdb
Mival?                apb
Minam                 ID
Micol                 object
MITab                 webPages
WEB_HOME              /cgi-bin/webdriver
# Retrieve large objects by name
MinamObj              ID
MicolObj              object
MITabObj?             webImages
MitypeObj?            image/gif
```

The **webImages** table, specified by **MITabObj** in the preceding **web.cnf** file, contains images stored as large objects. When you install the Web DataBlade module, the **webImages** table, described in [Appendix B](#), is created. The **ID** column (**MinamObj**) contains a unique identifier for each row of the **webImages** table. The **object** column (**MicolObj**) contains the large object. The **MivalObj** value, specified in the URL, specifies the unique value to determine which row to retrieve. Based on these variables, Webdriver builds the following SELECT statement to retrieve the large object:

```
SELECT $MicolObj FROM $MITabObj WHERE $MinamObj = '$MivalObj';
```

You can override the default values for the Webdriver variables in the URL to retrieve the large object. For example, you can specify the URL for a JPEG image stored as a large object within the database as follows:

```
http://myhost:port/cgi-bin/webdriver?MivalObj=apb_logo&MitypeObj=image/jpeg
```

Based on the preceding **web.cnf** file and URL, the call to retrieve the large object becomes:

```
SELECT object FROM webImages WHERE ID='apb_logo';
```

If the **MivalObj** variable is set, all other Webdriver environment variables (including **MITab**, **Micol**, **Minam**, and **Mival**) are ignored, and the specified large object is extracted. Therefore, do not specify **MivalObj** in your **web.cnf** file.

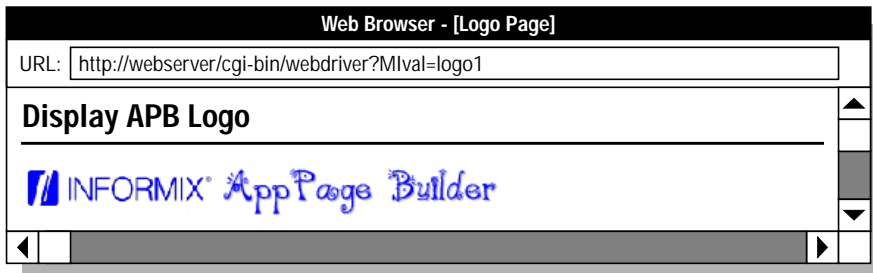
The following **logo1** AppPage retrieves an image from the **webImages** table:

```
<HTML>
<HEAD><TITLE>Logo Page</TITLE></HEAD>
<BODY>
<H3>Display APB Logo</H3>
<HR>
<IMG SRC=<?MIVAR>$WEB_HOME<?/MIVAR>?MivalObj=apb_logo&MItypeObj=image/gif>
</BODY>
</HTML>
```

For information on the MIVAR tag used in this example, see [“MIVAR Tag” on page 4-15](#).

The following is sample Web browser output for the **logo1** AppPage.

**Figure 2-3**  
*Display APB Logo*



## Retrieving Large Objects by LO Handle

You can retrieve large objects by large object handle (LO handle) when you dynamically retrieve the results of a SELECT statement. Set the following Webdriver variables to specify large object handles and their output MIME type.

Variable	Mandatory?	Content
LO	Yes	Large object handle.
MItypeObj	Yes	MIME type and subtype used to export the large object.

For example, the resulting URL for a GIF image stored as a large object within the database follows:

```
/cgi-bin/webdriver?  
LO=00000000a6b7c8d9000000002000000020000001a000001f60000000000010000000000  
686f626a656374e018000000400000001f0000000000000048009a9ecc0c7656b00c74df50  
&MItypeObj=image/gif
```

To retrieve large object handles, select them from the table that stores the large objects (you would not type in an LO handle as shown in the preceding URL). This method of retrieving large objects is useful when you are retrieving a dynamic list of objects.

If the LO variable is set, all other Webdriver environment variables (including MItab, Micol, MInam, and Mival) are ignored, and the specified large object is extracted. Therefore, do not specify LO in your **web.cnf** file.

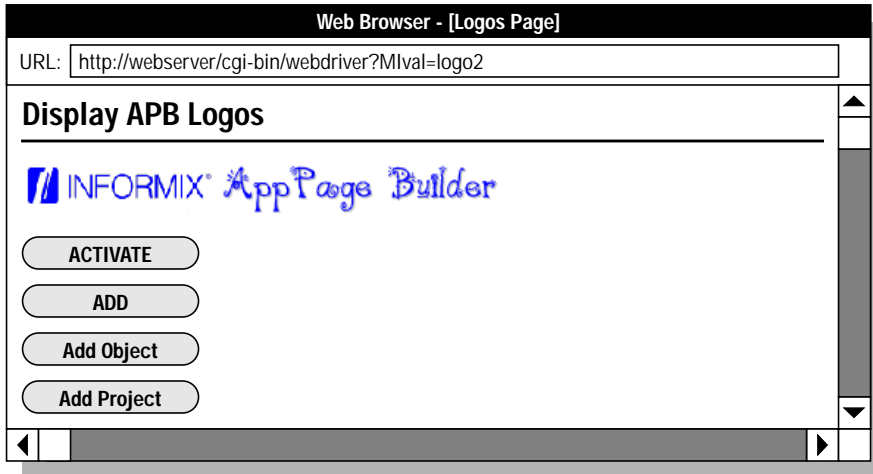
The following **logo2** AppPage retrieves images from the **webImages** table in the database, where the **object** column stores the large object handles of the images:

```
<HTML>  
<HEAD><TITLE>Logos Page</TITLE></HEAD>  
<BODY>  
<H3>Display APB Logos</H3>  
<HR>  
<?MISQL SQL="select object from webImages  
      where ID like 'apb_%';">  
<IMG SRC=$WEB_HOME?LO=$1&MItypeObj=image/gif><BR>  
<?/MISQL>  
</BODY>  
</HTML>
```

For more information about the MISQL tag used in this example, see “[MISQL Tag](#)” on page 4-5. For more information about large objects, see [Informix Guide to SQL: Reference](#).

The following is sample Web browser output for the **logo2** AppPage.

**Figure 2-4**  
*Display APB Logos*







---

# Using AppPage Builder

Invoking AppPage Builder . . . . .	3-3
Creating Web Applications in AppPage Builder . . . . .	3-6
Multimedia Content Support . . . . .	3-6
Administration Features. . . . .	3-7





**A**ppPage Builder (APB) is a Web DataBlade module application that enables you to create and maintain the AppPages that make up your Web applications. You can use APB to create AppPages with any Web browser that supports forms and tables, as defined in the HTML 3.0 specification. If you use a Web browser that supports client file upload, you can also use APB to manage multimedia content in the database.

***Important:** At the time that this guide was printed, only the Netscape 3.0 browser supported client file upload (using the ENCTYPE attribute of the FORM tag). However, this file upload capability has been proposed as part of future HTML standards.*

---

## Invoking AppPage Builder

To install APB, follow the instructions in the **\$INFORMIXDIR/extend/web.3.30.UC1/apb/README** file. To invoke APB, specify the following variable settings for Webdriver.

Variable	Value
MItab	webPages
MIcol	object
MInam	ID
MIval	apb

Set all of the preceding variables in your **web.cnf** file, or specify one or more of them in the URL that invokes APB. For the following **web.cnf** file, only the **Mlval** variable is omitted:

```
# web.cnf file
INFORMIXDIR      /local1/web/sqlldist
INFORMIXSERVER   IUS9_web
MI_USER          webdba
MI_PASSWORD      secret
MI_DATABASE      webdb
MInam            ID
MIcol            object
MItab            webPages
WEB_HOME         /cgi-bin/webdriver
```

Specify **Mlval** in the URL when you invoke APB:

```
http://myhost:port/cgi-bin/webdriver?Mlval=apb
```

For more information on invoking AppPages, see [“Invoking AppPages” on page 2-7](#).

When you invoke APB, the following AppPage is displayed.

**Figure 3-1**  
APB Main Menu

The screenshot shows a web browser window titled "Web Browser - [APB - Main Menu]". The address bar displays the URL: `http://webserver/cgi-bin/webdriver?Mlval=apb`. The main content area features the "INFORMIX AppPage Builder" logo at the top. Below the logo is a section titled "Main Menu" containing three buttons: "Add Object" (with description: "Add a new object to the database."), "Edit Object" (with description: "Edit, delete, and view objects stored in the database."), and "Admin Menu" (with description: "Manage user accounts, manage projects, and import AppPages."). At the bottom, there is a table of configuration settings.

User Name: <u>default</u>	Default Project: <u>doc</u>	TEXTAREA Width: <u>80</u>	Page Versions: <u>On</u>
User Level: <u>0</u>	Default Object: <u>AppPage</u>	TEXTAREA Height: <u>20</u>	WebLint Checking: <u>Level2</u>

## Creating Web Applications in AppPage Builder

Use APB to create and maintain AppPages and other multimedia objects that make up your Web applications. The following table describes the APB options displayed in the preceding screen.

Option	Action
Add Object	Add a new AppPage, Dynamic Tag, Audio, Document, Image, Video, or other Web application object.
Edit Object	Edit, delete, or view a Web application object.
Admin Menu	Edit or delete a user account, project, MIME type, or object type.

You can add or edit AppPages by typing or pasting into the text area or by uploading a client file. You can add and edit multimedia objects by uploading a client file.

The Admin Menu is described in [“Administration Features” on page 3-7](#).

## Multimedia Content Support

The following table lists the object types that are currently supported.

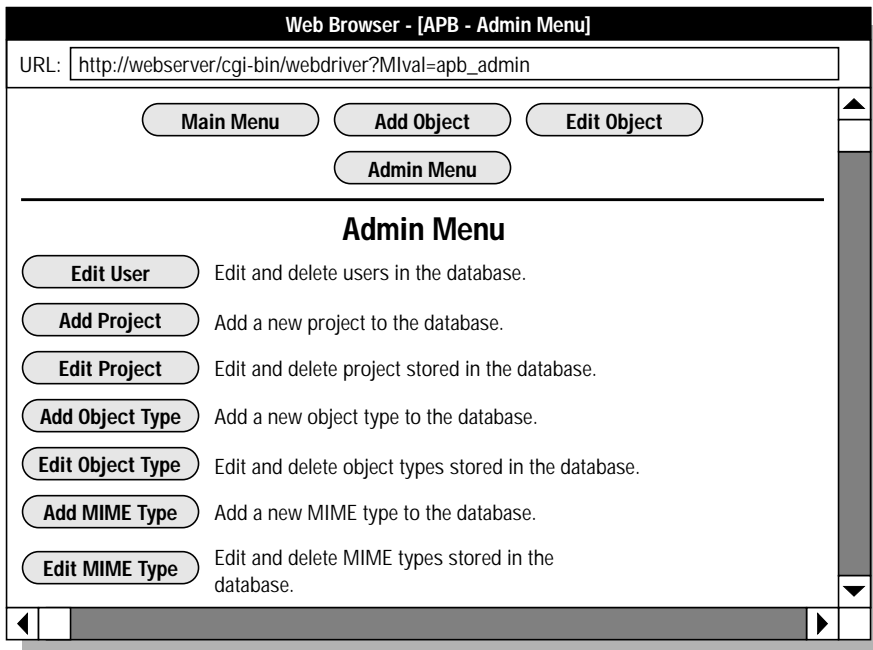
Object Type	Supported Formats	Table
AppPage	HTML	webPages
Dynamic Tag	HTML	webTags
Audio	AU, WAV, and AIFF	webAudios
Document	MS Word, MS PowerPoint, and Adobe PDF	webDocuments
Image	GIF and JPEG	webImages
Video	Quicktime, MPEG, and AVI	webVideos

[Appendix B](#) describes the complete APB schema and information on adding new object types and MIME types.

## Administration Features

When you invoke the **Admin Menu** of APB, the following AppPage is displayed.

**Figure 3-2**  
APB Admin Menu



The following table describes APB administrative features.

Option	Action
Edit User	Modify user preferences, including changing the user password, changing the default project or object type, changing the TEXTAREA height or width, turning AppPage versioning on or off, and changing the level of WebLint checking for syntax errors.
Add Project	Add a new project. A project contains all of the AppPages and other objects associated with a particular Web application.
Edit Project	Change the owner or the description of a project.
Add Object Type	Add a new multimedia object type to APB. For information on adding support for new object types, see <a href="#">“Adding Object Types” on page B-2</a> .
Edit Object Type	Modify the page suffix for an object type.
Add MIME Type	Add a new MIME type to an existing object type. For information on adding MIME types, see <a href="#">“MIME Types” on page B-2</a> .
Edit MIME Type	Edit an existing MIME type.



# Using Variables and Tags in AppPages

Web DataBlade Module Variables . . . . .	4-4
MISQL Tag . . . . .	4-5
Using System Variables to Format the SQL Results . . . . .	4-7
Column and Row Variables . . . . .	4-7
Processing Variables. . . . .	4-10
MI_NULL and MI_NOVALUE Variables . . . . .	4-13
MAXROWS Attribute . . . . .	4-14
MIVAR Tag . . . . .	4-14
NAME and DEFAULT Attributes . . . . .	4-15
MIBLOCK Tag . . . . .	4-17
COND Attribute . . . . .	4-17
MIERROR Tag . . . . .	4-19
TAG Attribute . . . . .	4-20
ERR Attribute . . . . .	4-21
Creating a Generic Error Handler . . . . .	4-22
Handling Error Conditions. . . . .	4-22
Processing Errors with Webdriver . . . . .	4-25
Special Characters in Web DataBlade Module Tags . . . . .	4-27
Special HTML Characters . . . . .	4-28
Special Formatting Characters . . . . .	4-28



**T**ags identify the elements of an HTML page and specify the structure and formatting for that page. The Web DataBlade module includes a set of tags that are processed by the **WebExplode** function.

Use the tags and tag attributes described in this chapter to create AppPages stored in the INFORMIX-Universal Server database. Use Web DataBlade module variables within these tags to customize your Web application. Web DataBlade module tags and variables enable you to dynamically generate queries to the database and control the HTML output resulting from the execution of these queries.

The following table lists the Web DataBlade module tags.

Tag	Description
<?MISQL><?/MISQL>	Contains SQL statements and formatting specifications for the data retrieved.
<?MIVAR><?/MIVAR>	Creates, assigns, and displays variables.
<?MIBLOCK><?/MIBLOCK>	Delimits logical blocks of HTML.
<?MIERROR><?/MIERROR>	Manages error processing.



**Important:** You can nest all Web DataBlade module tags within the MIBLOCK tag. You cannot nest Web DataBlade module tags within tags other than MIBLOCK.

**Tip:** The Web DataBlade module tags use the SGML processing instruction tag format, <?tag\_info>, <?/tag\_info>. An SGML processor ignores tags that it does not recognize, including Web DataBlade module tags.

## Web DataBlade Module Variables

Variable names are case sensitive, preceded by a dollar sign ( \$ ), and consist of alphanumeric and underscore characters. Variables beginning with an underscore are reserved for system use.

You can create user-defined variables and assign default values to them by adding them to your **web.cnf** file or by setting them using the NAME attribute of the MIVAR tag within an AppPage. You can override default values for existing user-defined variables in an MIVAR tag, an HTML form, or a URL that invokes an AppPage. For more information on how to assign and display variables using the MIVAR tag, see [“MIVAR Tag” on page 4-15](#).

Variables are global in scope within an AppPage and within any AppPages called recursively from that AppPage using the **WebExplode** function. For information on calling **WebExplode**, see [“WebExplode” on page 7-4](#). To pass variable values between AppPages that are not called recursively with **WebExplode**, you must explicitly pass the variables in a URL or an HTML form.



**Important:** Variables are only interpreted within MISQL, MIVAR, and MIERROR tags, and within the COND attribute of the MIBLOCK tag.

Web DataBlade module system variables are set by the database engine when an SQL statement is executed within the MISQL tag. For more information on using these variables, see [“Using System Variables to Format the SQL Results” on page 4-7](#).

You can also access Web server environment variables within AppPages in the same way that you access other Web DataBlade module variables. The following **env\_var** AppPage displays the value for the HTTP\_USER\_AGENT Web server environment variable:

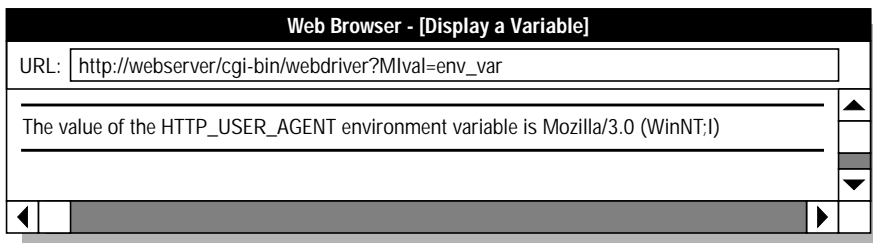
```
<HTML>
<HEAD><TITLE>Display a Variable</TITLE></HEAD>
<BODY>
<HR>The value of the HTTP_USER_AGENT environment variable is
    <?MIVAR>$HTTP_USER_AGENT<?/MIVAR><HR>
</BODY>
</HTML>
```

The following is sample output returned to the client:

```
<HTML>
<HEAD><TITLE>Display a Variable</TITLE></HEAD>
<BODY>
<HR>The value of the HTTP_USER_AGENT environment variable is
      Mozilla/3.0 (WinNT;I)<HR>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 4-1**  
*Display a Variable*



## MISQL Tag

Use the MISQL tag to execute SQL statements and to format the results of those statements in AppPages. The expansion of SQL takes place in the database engine before the resulting HTML is returned to the client (usually Webdriver).

The MISQL tag has the following tag attributes.

Attribute	Mandatory?	Description
SQL	Yes	Specifies a single SQL statement. The statement must be executable inside a transaction block.
NAME	No	Name of the variable to which the formatted results of the MISQL tag are assigned. If NAME is not specified, the results are output.
MAXROWS	No	Specifies the maximum number of formatted rows to be displayed.
COND	No	Tag is enabled only if this condition evaluates to TRUE (nonzero).
ERR	No	Specifies how an error should be processed. Because multiple errors can occur on an AppPage, use the ERR attribute to link the error processing to a particular MIERROR tag.

For more information on the ERR attribute, see [“MIERROR Tag” on page 4-19](#). For more information on the COND attribute, see [“MIBLOCK Tag” on page 4-17](#).

Specify the SQL statement to retrieve or modify database data in the SQL attribute of the MISQL tag. Specify formatting information, which indicates how to display the results of the SQL statement, between the start and end MISQL tags. The following section describes how to format the results of the SQL statement executed in the MISQL tag. The following is an example of an MISQL tag:

```
<?MISQL SQL="select name, company from customers;">$1 $2<?/MISQL>
```

## Using System Variables to Format the SQL Results

For each row that is returned by the SQL statement executed, the output is formatted according to the specifications between the start and end MISQL tags. You can use Web DataBlade module variables to:

- specify column and row formatting information.
- display processing information retrieved from the database engine, such as the number of columns or rows returned by a query.
- specify replacement values for NULL values or columns that do not have a value.

The following sections describe the system variables you can use to format SQL output.

### *Column and Row Variables*

To specify a column variable, use the format \$# where # is a column number from 1 up to the maximum number of columns in the row. Column variables are \$1 for the first column, \$2 for the second column, and so on. To specify all the columns, use an asterisk ( \$\* ) as described later in this section. The following **select1** AppPage illustrates the use of column variables:

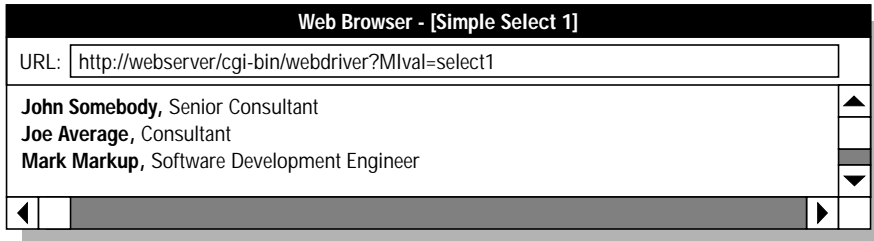
```
<HTML>
<HEAD><TITLE>Simple Select 1</TITLE></HEAD>
<BODY>
<?MISQL SQL="select first_name, last_name, title
      from staff;">
<B>$1 $2</B>, $3<BR><?/MISQL>
</BODY>
</HTML>
```

The following is sample output returned to the client:

```
<HTML>
<HEAD><TITLE>Simple Select 1</TITLE></HEAD>
<BODY>
<B>John Somebody</B>, Senior Consultant<BR>
<B>Joe Average</B>, Consultant<BR>
<B>Mark Markup</B>, Software Development Engineer<BR>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 4-2**  
Simple Select 1



To specify a row index, use the format [#], where # is a number from 1 to the maximum number of rows in the result set. If you do not specify a row index, [1] is assumed. The highest row index dictates the size of the *data window* that is displayed. The following **select2** AppPage illustrates column and row formatting specifications and the corresponding output:

```
<HTML>
<HEAD><TITLE>Simple Select 2</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<?MISQL SQL="select first_name, last_name from staff;">
<TR> <TD> $1 $2 </TD><TD> $1[2] $2[2] </TD> </TR>
<?/MISQL>
</TABLE>
</BODY>
</HTML>
```

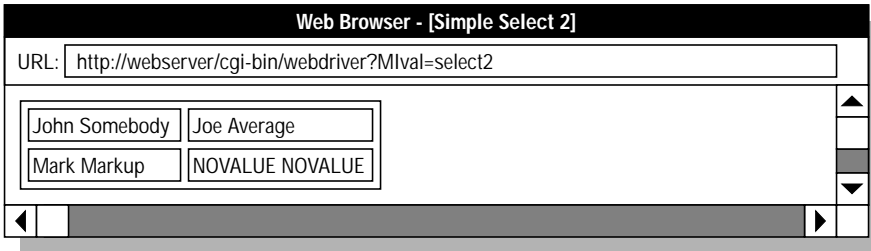
The following is sample output returned to the client:

```
<HTML>
<HEAD><TITLE>Simple Select 2</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<TR> <TD> John Somebody </TD><TD> Joe Average</TD> </TR>
<TR> <TD> Mark Markup </TD><TD> NOVALUE NOVALUE</TD> </TR>
</TABLE>
</BODY>
</HTML>
```



The following is sample Web browser output.

**Figure 4-3**  
Simple Select 2



The preceding data set is processed two rows at a time because [2] is the highest row index specified. If [3] was the highest row index specified, the data would be processed three rows at a time, and so on.

When you process multiple rows at a time, you might need to display rows with no value for the columns. See [“MI\\_NULL and MI\\_NOVALUE Variables” on page 4-13](#) for information on how to specify the output format for columns that have no value.

To display items that are repeated with every column, use a \$\* within curly braces ( { } ). This formatting technique is useful when you do not know the number of rows or columns that will be retrieved for display. The **select3** AppPage displays each column in a separate table cell:

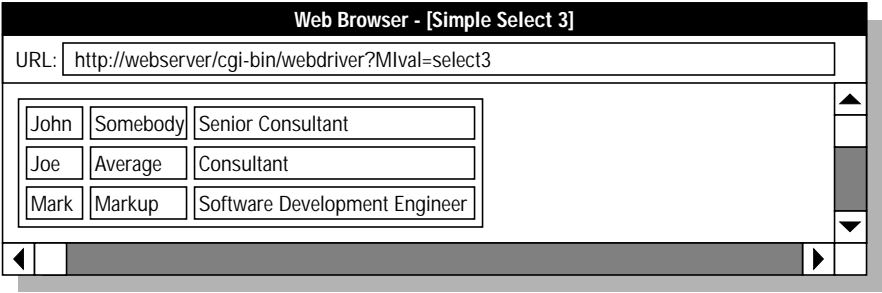
```
<HTML>
<HEAD><TITLE>Simple Select 3</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<?MISQL SQL="select * from staff;">
<TR> {<TD> $* </TD>} </TR>
<?/MISQL>
</TABLE>
</BODY>
</HTML>
```

The following is sample output returned to the client:

```
<HTML>
<HEAD><TITLE>Simple Select 3</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<TR> <TD> John </TD><TD> Somebody </TD><TD> Senior Consultant </TD> </TR>
<TR> <TD> Joe </TD><TD> Average </TD><TD> Consultant </TD> </TR>
<TR> <TD> Mark </TD><TD> Markup </TD><TD> Software Development Engineer </TD> </TR>
</TABLE>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 4-4**  
*Simple Select 3*



### Processing Variables

The following table lists additional system variables set by the database engine when an SQL statement is executed within the MISQL tag. You can use these processing variables to display more information about the results of the SQL statement.

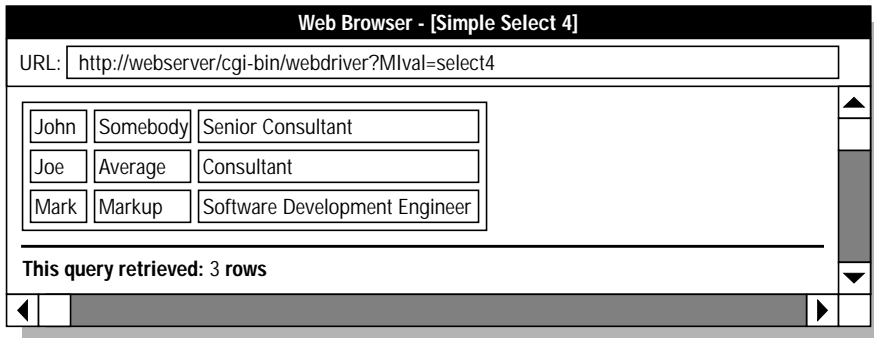
Variable	When Set?	Description
MI_COLUMNCOUNT	On execution	Number of columns retrieved in the SQL statement.
MI_CURRENTROW	On current row	Current formatted row being displayed during execution of the SQL statement. Set to the number of formatted rows displayed after the MISQL tag has been executed.
MI_ERRORCODE	On error	Error code returned from the SQL statement. All <b>WebExplode</b> errors return an error code of -937. For explanations of error codes, see <i>Informix Error Messages</i> . For more information on handling errors, see <a href="#">“MIERROR Tag” on page 4-19</a> .
MI_ERRORSTATE	On error	SQLSTATE returned from the SQL statement when an error occurs. For more information on handling errors, see <a href="#">“MIERROR Tag” on page 4-19</a> .
MI_ERRORMSG	On error	Error message returned from the SQL statement. For more information on handling errors, see <a href="#">“MIERROR Tag” on page 4-19</a> .
MI_ROWCOUNT	After execution	Number of rows retrieved in the SQL statement. Updated after processing is complete.
MI_SQL	On execution	SQL statement executed.

The following **select4** AppPage displays the number of rows returned by the last query executed:

```
<HTML>
<HEAD><TITLE>Simple Select 4</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<?MYSQL SQL="select * from staff;">
<TR> {<TD> $* </TD>} </TR>
<?/MYSQL>
</TABLE>
<HR>
<B>This query retrieved:</B>
<?MIVAR> $MI_ROWCOUNT <?/MIVAR> <B> rows </B>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 4-5**  
*Simple Select 4*



**Tip:** System variables maintain their values, and can be redisplayed, until the next **MISQL** tag is executed.

### ***MI\_NULL and MI\_NOVALUE Variables***

When you format your SQL output, **NULL** is displayed by default if a column has a **NULL** value. **NOVALUE** is displayed by default if you specify a column variable greater than the number of columns in the row or if there is no value for a column when the output is formatted to display multiple rows on the same line. Use the **MI\_NULL** variable to specify the text to be displayed when a **NULL** value is retrieved. Use the **MI\_NOVALUE** variable to specify the text to be displayed when no value is retrieved. In the following **select5** AppPage, the **MI\_NULL** and **MI\_NOVALUE** variables are assigned to a blank space:

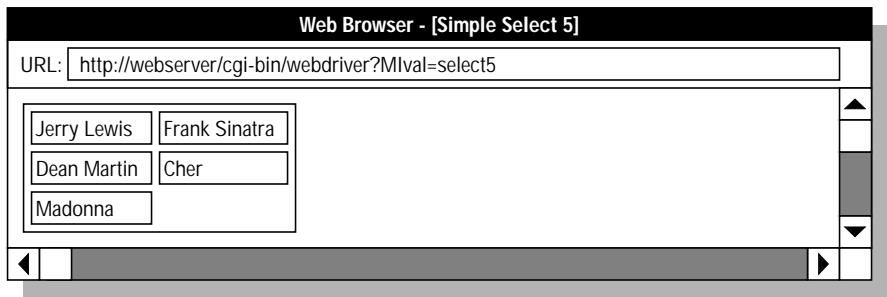
```
<HTML>
<HEAD><TITLE>Simple Select 5</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<?MIVAR NAME=$MI_NOVALUE> <?/MIVAR>
<?MIVAR NAME=$MI_NULL> <?/MIVAR>
<?MISQL SQL="select first_name, last_name from celebrities;">
<TR> <TD> $1 $2 </TD><TD> $1[2] $2[2] </TD> </TR> <?/MISQL>
</TABLE>
</BODY>
</HTML>
```

**The following is sample output returned to the client:**

```
<HTML>
<HEAD><TITLE>Simple Select 5</TITLE></HEAD>
<BODY>
<TABLE BORDER>
<TR> <TD> Jerry Lewis </TD><TD> Frank Sinatra </TD> </TR>
<TR> <TD> Dean Martin </TD><TD> Cher </TD> </TR>
<TR> <TD> Madonna </TD><TD> </TD> </TR>
</TABLE>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 4-6**  
*Simple Select 5*



A blank space replaces the NULL **last\_name** for Cher and Madonna. Because an odd number of rows is retrieved, a blank space also replaces the columns that have no value in the last table cell.

## MAXROWS Attribute

The MAXROWS attribute limits the number of rows that are displayed in the output of the MISQL tag. Use MAXROWS to limit the size of the result set being returned across the network if the queries you are executing might return a very large number of rows. Setting this attribute limits the system resources required to execute the query and return the results to the client. The following example limits the result set to twenty formatted rows displayed:

```
<?MISQL MAXROWS=20 SQL="select * from staff;"> { $* } <BR> <?/MISQL>
```

**Important:** If MAXROWS prevents all of the rows in the result set from being retrieved, MI\_ROWCOUNT is not updated.



## MIVAR Tag

The MIVAR tag enables you to assign and display variables. Use variables with Web DataBlade module tags to dynamically generate and format the results of SQL statements and to process errors.

The MIVAR tag has the following tag attributes.

Attribute	Mandatory?	Description
NAME	No	Name of the variable specified by the text between the start and end MIVAR tags. If NAME is not specified, the text between the start and end MIVAR tags is output. Variables within the text are expanded.
DEFAULT	No	Default value for any unassigned variables between the start and end MIVAR tags. This value can be another variable.
COND	No	Tag is enabled only if this condition evaluates to TRUE (nonzero).
ERR	No	Specifies how an error should be processed. Because multiple errors can occur on an AppPage, use the ERR attribute to link the error processing to a particular MIERROR tag.

For more information on the ERR attribute, see [“MIERROR Tag” on page 4-19](#). For more information on the COND attribute see [“MIBLOCK Tag” on page 4-17](#).

## NAME and DEFAULT Attributes

Use the NAME attribute to assign the value of the text between the start and end MIVAR tags to that variable name. The following **var1** AppPage demonstrates the assignment of variables:

```
<HTML>
<HEAD><TITLE>Variable Assignment 1</TITLE></HEAD>
<BODY>
<?MIVAR NAME=$TITLE>Entrepreneur</MIVAR>
<?MIVAR NAME=$SALUTATION> Dear $TITLE: </MIVAR>
<?MIVAR>$SALUTATION <BR> You are a sweepstakes winner!</MIVAR>
</BODY>
</HTML>
```

When you do not specify the NAME attribute, the text between the tags is output. Variables between the tags are expanded. As a result of the preceding AppPage, the following output is returned to the client:

```
<HTML>
<HEAD><TITLE>Variable Assignment 1</TITLE></HEAD>
<BODY>
Dear Entrepreneur: <BR> You are a sweepstakes winner!
</BODY>
</HTML>
```

Use the DEFAULT attribute to specify a default value for any unassigned variables between the start and end MIVAR tags. In the following **var2** AppPage, the DEFAULT attribute is used to replace any unassigned variables between the start and end MIVAR tags with the value specified in the DEFAULT attribute:

```
<HTML>
<HEAD><TITLE>Variable Assignment 2</TITLE></HEAD>
<BODY>
<?MIVAR NAME=$TITLE DEFAULT="Sir or Madam"> $INPUT_TITLE </MIVAR>
<?MIVAR> Dear $TITLE: <BR> You are a sweepstakes winner! </MIVAR>
</BODY>
</HTML>
```



If the `INPUT_TITLE` variable is unassigned, the preceding AppPage returns the following output to the client:

```
<HTML>
<HEAD><TITLE>Variable Assignment 2</TITLE></HEAD>
<BODY>
Dear Sir or Madam: <BR> You are a sweepstakes winner!
</BODY>
</HTML>
```

If the `INPUT_TITLE` variable is assigned elsewhere—for example, in the calling URL or in an HTML form—that value overrides the default value.

**Important:** Within the `NAME` attribute assignment (`NAME=$varname`), the `$` in front of the variable name is optional. In all other occurrences, you must precede the variable name with a `$`.



## MIBLOCK Tag

The MIBLOCK tag enables you to delimit logical blocks of HTML to be executed based on a variety of conditions.

The MIBLOCK tag has the following attributes.

Attribute	Mandatory?	Description
COND	No	Tag is enabled only if this condition evaluates to TRUE (nonzero).
ERR	No	Specifies how an error should be processed. Because multiple errors can occur on an AppPage, use the ERR attribute to link the error processing to a particular MIERROR tag.

The ERR attribute for the MIBLOCK tag is only invoked if an error occurs when evaluating the condition specified in the COND attribute. For more information on the ERR attribute, see [“MIERROR Tag” on page 4-19](#).

## COND Attribute

All Web DataBlade module and dynamic tags can have a COND attribute. The COND attribute specifies a condition that is evaluated before the tag is processed. If the condition is true, the tag is processed. Conditions are variables or variable expressions that evaluate to FALSE if 0 and TRUE if nonzero.

The following **cond\_display** AppPage uses the COND attribute within an MIBLOCK tag to conditionally display text depending on the value of a variable:

```
<HTML>
<HEAD><TITLE>Conditional Display</TITLE></HEAD>
<BODY>
<?MIVAR COND=$(NXST,$VAR1) NAME=$VAR1>0<?/MIVAR>
This is always displayed.<BR>
<?MIBLOCK COND=$VAR1>
    This is conditionally displayed if VAR1 is nonzero.<BR>
    <B>The value of VAR1 is: <?MIVAR>$VAR1<?/MIVAR></B><BR>
<?/MIBLOCK>
This is always displayed.
</BODY>
</HTML>
```

If the condition in the MIBLOCK tag evaluates to TRUE—that is, if the VAR1 variable has been assigned a value other than 0 in the URL that calls it or in an HTML form—the value of the variable is displayed. For example, if the AppPage is called with the URL `http://myhost/cgi-bin/webdriver?Mival=cond_display&VAR1=1`, the preceding AppPage returns the following output to the client:

```
<HTML>
<HEAD><TITLE>Conditional Display</TITLE></HEAD>
<BODY>
This is always displayed.<BR>
    This is conditionally displayed if VAR1 is nonzero.<BR>
    <B>The value of VAR1 is: 1</B><BR>
This is always displayed.
</BODY>
</HTML>
```

If the VAR1 variable is undefined, the preceding AppPage returns the following output to the client:

```
<HTML>
<HEAD><TITLE>Conditional Display</TITLE></HEAD>
<BODY>
This is always displayed.<BR>
This is always displayed.
</BODY>
</HTML>
```

NXST and other variable processing functions you can use to create variable expressions are described in [Chapter 5, “Using Variable Processing Functions in AppPages.”](#)



**Important:** The MIBLOCK tag is the only tag in which you can nest other Web DataBlade module tags. You can also nest MIBLOCK tags within MIBLOCK tags. Variables are interpreted only within MISQL, MIVAR, and MIERROR tags, and within the COND attribute of the MIBLOCK tag.

---

## MIERROR Tag

Use the MIERROR tag to specify the processing that takes place when an error is encountered within other Web DataBlade module tags. Errors can occur if the database engine cannot successfully process an SQL statement, if you try to access an unassigned variable, or if you use an incorrect tag construct.



**Important:** The placement of MIERROR tags is significant. You must specify MIERROR tags within an AppPage prior to invoking them.

The MIERROR tag has the following tag attributes.

Attribute	Mandatory?	Description
TAG	No	Specifies the type of processing when an error occurs. This attribute must be assigned to an MISQL, MIVAR, or dynamic tag. If you make the TAG=MISQL attribute assignment, the SQL attribute must also be specified. If you make the TAG=MIVAR attribute assignment, the tag is equivalent to an MIVAR tag with no NAME attribute, and the text between start and end tags is output. Default is TAG=MIVAR.
COND	No	Tag is enabled only if this condition evaluates to TRUE (nonzero).
ERR	No	Specifies how an error should be processed. Because multiple errors can occur on an AppPage, use the ERR attribute to link the error processing to a particular MIERROR tag.

For more information on the COND attribute, see [“MIBLOCK Tag” on page 4-17](#).

### TAG Attribute

When an error occurs and an MIERROR tag is invoked, the tag behaves like an MISQL, MIVAR, or dynamic tag, depending on the TAG attribute. Use the TAG=MISQL attribute assignment to execute the SQL statement specified in the SQL attribute. In the following example, the SELECT statement specified in the SQL attribute is performed when the MIERROR tag is invoked. This error handler retrieves an error message from the **my\_weberr\_catalog** table:

```
<?MIERROR TAG=MISQL SQL="select error_msg from my_weberr_catalog
where error_id='$MI_ERRORCODE';">$1<?/MIERROR>
```



This is equivalent to executing the following MISQL tag:

```
<?MISQL SQL="select error_msg from my_webserv_catalog
where error_id='$MI_ERRORCODE';">$1<?/MISQL>
```

***Important:*** When an error occurs during the processing of an AppPage, the entire transaction is rolled back. Therefore an INSERT, UPDATE, or any other update performed by the SQL statement in the MIERROR tag is also rolled back.

An MIERROR tag with the TAG=MIVAR attribute assignment behaves like an MIVAR tag with no NAME attribute. Use the TAG=MIVAR attribute assignment to output an error message. For example, when the following MIERROR tag is invoked, the text between the start and end tags is output:

```
<?MIERROR TAG=MIVAR>
<B>Please contact your Web Administrator.</B><BR><?/MIERROR>
```

This is equivalent to executing the following MIVAR tag:

```
<?MIVAR><B>Please contact your Web Administrator.</B><BR><?/MIVAR>
```

## ERR Attribute

The ERR attribute links an MISQL, MIVAR, MIBLOCK, or dynamic tag with an MIERROR tag to be invoked if an error occurs in the processing of that tag. Specify an ERR attribute in an MISQL, MIVAR, or MIBLOCK tag to invoke an MIERROR tag with a matching ERR attribute when an error occurs. For example, define an MIERROR tag as follows:

```
<?MIERROR ERR=BADTABLENAME TAG=MISQL SQL="select error_msg
from my_webserv_catalog where error_id='BADTABLENAME';">$1<?/MIERROR>
```

This error handler is invoked if an error occurs during the processing of an MISQL or MIVAR tag, the COND attribute of an MIBLOCK tag, or dynamic tag with the same ERR attribute assignment (ERR=BADTABLENAME). If the following MISQL tag generates an error when it is executed, the preceding MIERROR tag, with the matching ERR attribute, is invoked:

```
<?MISQL ERR=BADTABLENAME SQL="select count(*) from $TABLE_NAME">$1<BR><?/MISQL>
```

If no MIERROR tag with a matching ERR attribute precedes the MISQL, MIVAR, or MIBLOCK, or dynamic tag that generates an error in the AppPage, the generic error handler, described in the following section, is invoked.

## Creating a Generic Error Handler

A generic error handler is an MIERROR tag without an ERR attribute. Create a generic error handler to be invoked if an error occurs during the processing of a tag that has no ERR attribute or an invalid ERR attribute. The following is an example of a generic error handler that logs an error message to the trace file:

```
<?MIERROR TAG=MIVAR>$(TRACMSG,An error occurred on page: $Mival.)<?/MIERROR>
```

For more information on the TRACMSG variable processing function, see [“Enabling WebExplode Tracing” on page 0-7](#).

## Handling Error Conditions

When an MIERROR tag is first encountered on an AppPage, only the COND and ERR attributes are evaluated. Variables between the start and end tags are not evaluated until the error condition is encountered. Since the condition is evaluated only the first time the MIERROR tag is encountered, you must call the **WebExplode** function recursively to handle specific error conditions that must be evaluated after an error occurs. The **enter\_table** AppPage allows you to type a table name into the **TABLE** text-entry field in the following HTML form:

```
<HTML>
<HEAD><TITLE>Enter Table Name</TITLE></HEAD>
<BODY>
<H2>Enter table name:<H2>
<?MIVAR NAME=$TABLE><?/MIVAR>
<?MIVAR>
<FORM METHOD=POST ACTION="$WEB_HOME">
<INPUT TYPE=TEXT NAME=TABLE VALUE=$TABLE>
<INPUT TYPE=HIDDEN NAME=Mival VALUE="count_rows">
<INPUT TYPE=SUBMIT VALUE="Count Rows"><HR>
<?/MIVAR>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 4-7**  
Enter Table Name

The following **count\_rows** AppPage processes the preceding form. This processing AppPage contains a generic error handler that uses the **WebExplode** function to call the **error\_handler** AppPage if an error occurs:

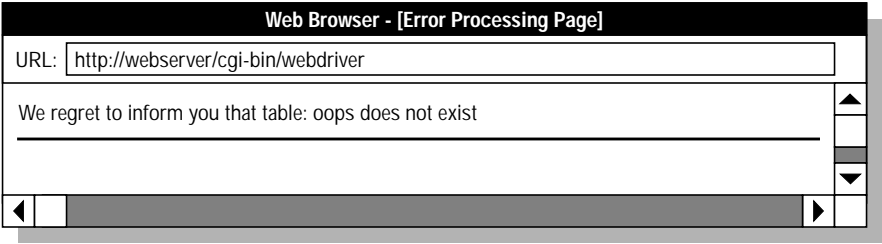
```
<HTML>
<HEAD><TITLE>Count Rows</TITLE></HEAD>
<BODY>
<!-- count the number of rows in the table -->
<!-- specified, call the error_handler page -->
<!-- if an error occurs on this page. -->
<?MIERROR TAG=MISQL SQL="select WebExplode(object, '')
      from webPages where ID='error_handler';">$1<?/MIERROR>
<?MISQL SQL="select count(*) from $TABLE;">
<BR><B>Number of rows in table $TABLE:
</B>$(FIX,$1)<BR><?/MISQL>
</BODY>
</HTML>
```

If an error occurs in the **count\_rows** AppPage, the preceding MIERROR tag is invoked, and the **error\_handler** AppPage is executed. The **error\_handler** AppPage evaluates the error code:

```
<HTML>
<HEAD><TITLE>Error Processing Page</TITLE></HEAD>
<BODY>
<?MIVAR NAME=$done>NO<?/MIVAR>
<?MIBLOCK COND=$(EQ,$MI_ERRORCODE,-206)>
    We regret to inform you that table:
    <?MIVAR>$TABLE<?/MIVAR> does not exist.
    <?MIVAR NAME=done>YES<?/MIVAR>
<?/MIBLOCK>
<?MIBLOCK COND=$(EQ,$MI_ERRORCODE,-201)>
    You entered one or more blank spaces as a table name. Please go
    back and enter a table name.
    <?MIVAR NAME=done>YES<?/MIVAR>
<?/MIBLOCK>
<?MIBLOCK COND=$(AND,$(EQ,$MI_ERRORCODE,-937),$(EQ,$MI_ERRORSTATE,UWEB1))>
    You have not specified a table. Please go back and enter
    a table name.
    <?MIVAR NAME=done>YES<?/MIVAR>
<?/MIBLOCK>
<?MIBLOCK COND=$(EQ,$done,NO)>
    You received an unexpected error:
    <?MIVAR>$MI_ERRORMSG<?/MIVAR> <BR>
    Please contact your administrator.
<?/MIBLOCK>
<HR>
</BODY>
</HTML>
```

The following shows sample Web browser output when the user specifies a nonexistent table.

**Figure 4-8**  
Error Processing Page





## Processing Errors with Webdriver

Each AppPage is executed within a single transaction. When the **WebExplode** function encounters an exception during execution of an AppPage, all of the SQL statements on that AppPage are rolled back. **WebExplode** raises an exception when you execute a tag with an unassigned variable or incorrect tag construct. INFORMIX-Universal Server raises an exception when an SQL error is generated.

If an MIERROR tag is invoked for the exception that occurs, the **WebExplode** function returns a XUWEA1 error code along with the text of the MIERROR message. Webdriver displays the message text of the MIERROR tag returned by **WebExplode** (up to an 8 K buffer limit).

In the following **catch\_error** AppPage, a generic error handler returns a message to the user if the TEST\_VAR variable is unassigned:

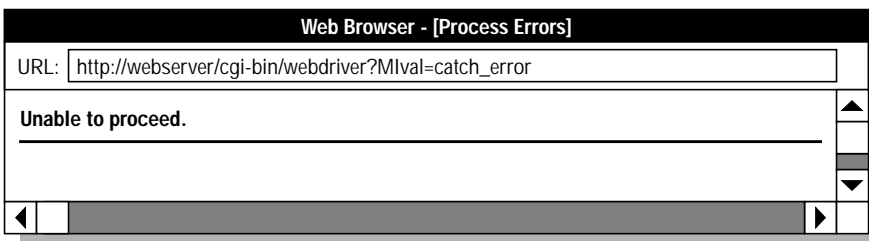
```
<HTML>
<HEAD><TITLE>Error Processing Page</TITLE></HEAD>
<BODY>
<?MIERROR TAG=MIVAR><HTML>
<HEAD><TITLE>Process Errors</TITLE></HEAD>
<BODY><B>Unable to proceed.</B><HR></BODY></HTML><?/MIERROR>
The value of $$TEST_VAR is <?MIVAR>$TEST_VAR</MIVAR>
</BODY>
</HTML>
```



**Tip:** Only HTML within the MIERROR tag is returned to the client.

The following is sample Web browser output.

**Figure 4-9**  
MIERROR Tag Output



If no MIERROR tag exists to handle the error that occurs, Webdriver output depends on the setting of the MI\_WEBSHOWEXCEPTIONS **web.cnf** file variable.

Variable	Mandatory?	Content
MI_WEBSHOWEXCEPTIONS	No	Set to on or off. When on, Webdriver displays the database exception returned by <b>WebExplode</b> . When off, Webdriver displays the HTTP/1.0 500 Server error message. Default is off.

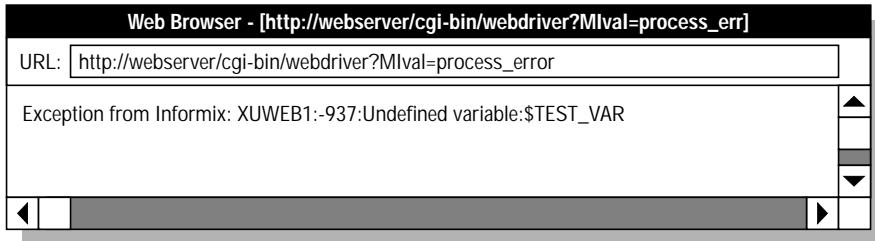
If you set the MI\_WEBSHOWEXCEPTIONS variable to on and no MIERROR tag is invoked for the exception that occurs, the database exception message returned by **WebExplode** is displayed by Webdriver. If you set the MI\_WEBSHOWEXCEPTIONS variable to off and no MIERROR tag is invoked for the exception that occurs, Webdriver displays the HTTP/1.0 500 Server error message.

The following **process\_error** AppPage has no MIERROR tag:

```
<HTML>
<HEAD><TITLE>Error Processing Page</TITLE></HEAD>
<BODY>
The value of $TEST_VAR is <?MIVAR>$TEST_VAR<?/MIVAR>
</BODY>
</HTML>
```

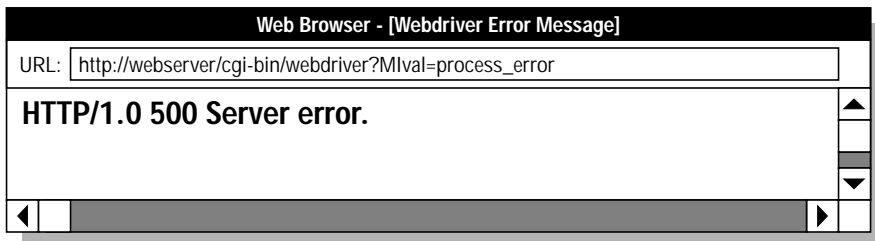
The following is sample output when the TEST\_VAR variable has not been assigned and MI\_WEBSHOWEXCEPTIONS is set to on.

**Figure 4-11**  
*Show Exceptions On*



The following is sample output when the TEST\_VAR variable has not been assigned and MI\_WEBSHOWEXCEPTIONS is set to off.

**Figure 4-11**  
*Show Exceptions Off*



## Special Characters in Web DataBlade Module Tags

You must make the following replacements within your AppPages:

- Special HTML characters within Web DataBlade module tags
- Special formatting characters within formatting specifications

## Special HTML Characters

You must replace the double quote character with its entity reference if the character occurs within Web DataBlade module tags (between angle braces).

Character	Entity Reference
"	&quot;

For example, to execute the SQL statement,

```
insert into staff values ('Walt "Speedy"', 'Wait', 'Engineer');
```

use the following entity reference in the MISQL tag:

```
<?MISQL SQL="insert into staff values  
('Walt &quot;Speedy&quot;', 'Wait', 'Engineer');"> 1 row inserted. </MISQL>
```

## Special Formatting Characters

You must replace characters that normally specify formatting information with the following replacements when they occur within formatting specifications (between the start and end tags).

Character	Replacement
{	{{
}	}}
\$	\$\$

For example, the MIVAR tag:

```
<?MIVAR>You may have won $$1,000,000.00!</MIVAR>
```

returns the following output to the client:

```
You may have won $1,000,000.00!
```

---

# Using Variable Processing Functions in AppPages

Variable Processing Functions . . . . .	5-3
Using Variable Expressions in AppPages . . . . .	5-8
Using Arithmetic Functions in Variable Expressions . . . . .	5-8
Using SEPARATE and REPLACE in Variable Expressions . . . . .	5-9
Using Variable Expressions to Format Output Conditionally . . . . .	5-12
Special Characters in Variable Expressions. . . . .	5-16



# V

variable processing functions enable calculations to be performed using variables that are passed into an AppPage, generated within the AppPage, or returned from the INFORMIX-Universal Server database.

Variables are identified by a dollar sign ( \$ ) followed by alphanumeric and underscore characters. Variable expressions start with a \$ character followed by the expression within parentheses, \$(*expression*). Variable processing functions add the ability to evaluate and manipulate variables within variable expressions. Variable expressions can contain other variable expressions.



**Important:** Variables and variable expressions are interpreted only within MISQL, MIVAR, and MIERROR tags, and within the COND attribute of the MIBLOCK tag.

---

## Variable Processing Functions

The following functions can be performed on Web DataBlade module variables.

Function	Returns
\$(+,val1,val2,...,valn)	Returns the sum of the numbers val1,val2,...,valn.
\$(-,val1,val2,...,valn)	Returns the result of subtracting the numbers val2 through valn from val1.
\$(*,val1,val2,...,valn)	Returns the result of multiplying the numbers val1,val2,...,valn.
\$(/,val1,val2,...,valn)	Returns the result of dividing the number val1 by val2,...,valn.

(1 of 5)

Function	Returns
<code>\$ (=, val1, val2)</code>	If the numbers <i>val1</i> and <i>val2</i> are equal, 1 is returned; otherwise 0 is returned.
<code>\$ (&lt;, val1, val2)</code>	If the number <i>val1</i> is less than <i>val2</i> , 1 is returned; otherwise 0 is returned.
<code>\$ (&gt;, val1, val2)</code>	If the number <i>val1</i> is greater than <i>val2</i> , 1 is returned; otherwise 0 is returned.
<code>\$ (!=, val1, val2)</code>	If the numbers <i>val1</i> and <i>val2</i> are not equal, 1 is returned; otherwise 0 is returned.
<code>\$ (&lt;=, val1, val2)</code>	If the number <i>val1</i> is less than or equal to <i>val2</i> , 1 is returned; otherwise 0 is returned.
<code>\$ (&gt;=, val1, val2)</code>	If the number <i>val1</i> is greater than or equal to <i>val2</i> , 1 is returned; otherwise 0 is returned.
<code>\$ (AND, val1, val2, ..., valn)</code>	Returns the logical AND of the integers <i>val1</i> through <i>valn</i> . Processing halts when a false condition is reached.
<code>\$ (EC, string1, string2)</code>	If <i>string1</i> and <i>string2</i> are identical, regardless of case, 1 is returned; otherwise 0 is returned.
<code>\$ (EQ, string1, string2)</code>	If <i>string1</i> and <i>string2</i> are identical, including case, 1 is returned; otherwise 0 is returned.
<code>\$ (FIX, value)</code>	Truncates the real number <i>value</i> to an integer by discarding any fractional part.
<code>\$ (HTTPHEADER, name, value)</code>	Adds the HTTP header <i>name</i> with the <i>value</i> to an AppPage. See <a href="#">“Adding HTTP Headers to AppPages” on page 8-3</a> for more information.

(2 of 5)



Function	Returns
<code>\$(IF,expr,dottrue)</code>	If <i>expr</i> is nonzero, <i>dottrue</i> is evaluated and returned.
<code>\$(IF,expr,dottrue,dofalse)</code>	If <i>expr</i> is nonzero, <i>dottrue</i> is evaluated and returned. Otherwise, <i>dofalse</i> is evaluated and returned. The branch not chosen by <i>expr</i> is not evaluated.
<code>\$(INDEX,which,string)</code>	<i>string</i> is assumed to contain one or more values delimited by the comma. The numeric value <i>which</i> selects one of these values to be extracted. Numbering of the items in <i>string</i> begins with 0.
<code>\$(ISINT,value)</code>	If <i>value</i> is an integer, 1 is returned; otherwise 0 is returned. (A number that is of equal value to an integer, such as 1.0, evaluates to 1.)
<code>\$(ISNUM,value)</code>	If <i>value</i> is numeric, 1 is returned; otherwise 0 is returned.
<code>\$(LOWER,string)</code>	Returns <i>string</i> converted to lowercase letters.
<code>\$(NC,string1,string2)</code>	If <i>string1</i> and <i>string2</i> are not identical, regardless of case, 1 is returned; otherwise 0 is returned.
<code>\$(NE,string1,string2)</code>	If <i>string1</i> and <i>string2</i> are not identical, including case, 1 is returned; otherwise 0 is returned.
<code>\$(NOT,value)</code>	Returns the logical negation of <i>value</i> .

(3 of 5)

Function	Returns
<code>\$(NTH,which,arg0,arg1,...,argN)</code>	Evaluates and returns the argument selected by <i>which</i> . If <i>which</i> is 0, <i>arg0</i> is returned, and so on. Note the difference between <code>\$(NTH)</code> and <code>\$(INDEX)</code> ; <code>\$(NTH)</code> returns one of a series of arguments to the function while <code>\$(INDEX)</code> extracts a value from a comma-delimited string passed as a single argument. Arguments not selected by <i>which</i> are not evaluated.
<code>\$(NXST,varname)</code>	If variable <i>varname</i> does not exist (has not been assigned a numeric or string value), 1 is returned; otherwise 0 is returned.
<code>\$(OR,val1,val2,...,valn)</code>	Returns the logical OR of the integers <i>val1</i> through <i>valn</i> . Processing halts when a true condition is reached.
<code>\$(POSITION,string1,string2)</code>	Returns the starting position of <i>string2</i> within <i>string1</i> . If <i>string2</i> is not found, 0 is returned.
<code>\$(REPLACE,string1,string2,string3)</code>	Replaces all instances of <i>string2</i> with <i>string3</i> within <i>string1</i> .
<code>\$(SEPARATE,varvector,string)</code>	Separates items in the variable vector <i>varvector</i> with the string value <i>string</i> .
<code>\$(SETVAR,varname,value)</code>	Sets the variable <i>varname</i> to the numeric or string <i>value</i> .
<code>\$(STRFILL,string,ncopies)</code>	Returns the result of concatenating <i>ncopies</i> number of copies of <i>string</i> .
<code>\$(STRLEN,string)</code>	Returns the length of <i>string</i> .
<code>\$(SUBSTR,string,start,length)</code>	Returns the substring of <i>string</i> starting at character <i>start</i> and extending for <i>length</i> characters. Characters in the string are numbered from 1. If <i>length</i> is omitted, the entire remaining length of the string is returned.

Function	Returns
<code>\$(TRACMSG,string)</code>	Writes the message <i>string</i> to a trace file. See <a href="#">“Enabling WebExplode Tracing” on page A-7</a> for more information.
<code>\$(TRIM,string)</code>	Removes leading and trailing white space from <i>string</i> .
<code>\$(UNHTML,string)</code>	Returns <i>string</i> with special HTML characters replaced with their entity reference for display by a Web browser. See <a href="#">“WebUnHTML” on page 7-12</a> for a description of this functionality implemented as a server function.
<code>\$(UNSETVAR,varname)</code>	Unsets the variable <i>varname</i> . No error is generated if <i>varname</i> is not set.
<code>\$(UPPER,string)</code>	Returns <i>string</i> converted to uppercase letters.
<code>\$(URLDECODE,string)</code>	Returns <i>string</i> with all hexadecimal values replaced with their nonalphanumeric ASCII characters. See <a href="#">“WebURLDecode” on page 7-14</a> for a description of this functionality implemented as a server function.
<code>\$(URLENCODE,string)</code>	Returns <i>string</i> with all nonalphanumeric ASCII characters replaced with their hexadecimal values. See <a href="#">“WebURLEncode” on page 7-15</a> for a description of this functionality implemented as a server function.
<code>\$(XOR,val1,val2,...,valn)</code>	Returns the logical XOR of the integers <i>val1</i> through <i>valn</i> .
<code>\$(XST,varname)</code>	If variable <i>varname</i> exists (has been assigned a numeric or string value), 1 is returned; otherwise 0 is returned.

(5 of 5)



**Important:** Arithmetic functions accept either floating-point or integer arguments and perform all calculations in floating-point. Arithmetic functions that allow more than two arguments allow a maximum of 10.

**Tip:** Spaces are significant in the evaluation of variable expressions. For example, the variable expression `$(EQ,$var1,$var2)` is not equivalent to `$(EQ, $var1,$var2)` because the latter expression has a space before the string `$var1`.

---

## Using Variable Expressions in AppPages

The following sections show a variety of uses for variable processing functions to create simple and complex variable expressions.

### Using Arithmetic Functions in Variable Expressions

The following **varexp1** AppPage is an example of variable processing within an MIVAR tag:

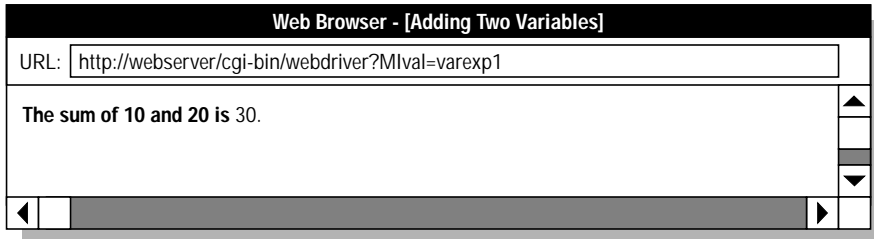
```
<HTML>
<HEAD><TITLE>Adding Two Variables</TITLE></HEAD>
<BODY>
  <?MIVAR NAME=NUMA>10<?/MIVAR>
  <?MIVAR NAME=NUMB>20<?/MIVAR>
  <?MIVAR><B>The sum of $NUMA and $NUMB is</B>
  $(+,$NUMA,$NUMB).
<?/MIVAR>
</BODY>
</HTML>
```

The following is sample output returned to the client:

```
<B>The sum of 10 and 20 is</B> 30.
```

The following is sample Web browser output.

**Figure 5-1**  
*Adding Two Variables*



## Using SEPARATE and REPLACE in Variable Expressions

Use the SEPARATE variable processing function to separate elements in a variable vector. A variable vector consists of multiple variables with the same name, passed into the AppPage using check boxes or the MULTIPLE attribute of selection lists.

Use the REPLACE variable processing function to specify a string to be replaced within text. For example, you must replace single quotes with two single quotes so that single quotes can be inserted into the database. If this replacement is not made and the text being inserted into the database contains single quotes, the INSERT statement is not built correctly.

The following **table\_prog** AppPage uses both the SEPARATE and REPLACE variable processing functions. The columns of the **employees** table are displayed as a check box list. Check one or more columns of the **employees** table to be retrieved, then submit the form. The form is posted to the same **table\_prog** AppPage. On the second call to the AppPage, the SQL statement that retrieves the checked columns of the **employees** table is built, using the SEPARATE variable processing function to place commas between the selected columns in the SELECT statement. The REPLACE variable is then used to replace the commas separating items in the variable vector with TH tags to create an HTML table row. Finally, the output is displayed in an HTML table.

```
<HTML>
<HEAD><TITLE> Select from Table</TITLE></HEAD>
<BODY>
<!-- Show columns of employees table in a form --->
<!-- with multi-value check box. Turn checked --->
<!-- columns into a comma-separated list. --->
<?MIVAR NAME=$column_headers> </MIVAR>
<HR>
<STRONG>Select Columns from Employees Table</STRONG><BR>
<?MIVAR><FORM METHOD=POST ACTION="$WEB_HOME"></MIVAR>
<?MISQL SQL="select a.colname, colno from syscolumns a, systables b
  where a.tabid = b.tabid and b.tabname = 'employees' order by colno;">
<INPUT TYPE=CHECKBOX NAME=column_list VALUE="$1">$1</MISQL>
<INPUT TYPE=HIDDEN NAME=Mival VALUE="table_prog">
<INPUT TYPE=SUBMIT VALUE="Get Rows"><HR>
</FORM>
<!-- On the second time through the form, --->
<!-- retrieve the selected columns from the --->
<!-- database, display in table format. --->
<?MIVAR COND=$(NXST,$column_list) NAME=$column_list></MIVAR>
<?MIBLOCK COND=$(NOT,$(EQ,$column_list,))>
  <?MIVAR NAME=$select_list>$(SEPARATE,$column_list,",")</MIVAR>
  <?MIVAR NAME=$column_headers>$(REPLACE,$select_list,",", "<TH><TH>")</MIVAR>
  <TABLE BORDER>
    <TR><TH><?MIVAR>$column_headers</MIVAR></TH></TR>
    <?MISQL SQL="select $select_list from employees order by 1;">
    <TR><TD>$*</TD></TR>
  </MISQL>
  </TABLE>
</MIBLOCK>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 5-2**  
Select from Table

Web Browser - [Select from Table]

URL:

Select Columns from Employees Table:

☒ first\_name ☒ last\_name ☒ title ☐ onsite ☐ department

first_name	last_name	title
Beth	Hume	Product Manager
Betty	Pen	Senior Line Worker
Craig	Wallace	Line Worker
Gonzo	Babbage	Product Manager
Kermit	French	Event Co-ordinator
Sarah	Dun	Event Co-ordinator
Simon	Smith	Senior Salesman
Wilma	Jones	Salesman

## Using Variable Expressions to Format Output Conditionally

You can also use variable processing functions to format output conditionally. The following **varexp2** AppPage illustrates how a variable expression can be used to process the results of a SELECT statement. This AppPage queries the **employees** and **departments** tables and displays the employees by department. The department name is not output when the name has not changed from the previous row retrieved:

```
<HTML>
<HEAD><TITLE>Conditional Output</TITLE></HEAD>
<BODY>
<B>Display employee names by department: </B>
<?MIVAR NAME=LAST><?/MIVAR>
<TABLE BORDER=1>
<?MISQL SQL="select b.name, a.first_name, a.last_name from
      employees a , departments b where a.department = b.name
      order by b.name, a.last_name;">
<TR>
<TD>$(IF,$(NE,$1,$LAST),$1)</TD>
<TD> $2 $3</TD> $(SETVAR,$LAST,$1)
</TR>
<?/MISQL>
</TABLE>
</BODY>
</HTML>
```



The following is sample Web browser output.

**Figure 5-3**  
Conditional Output

Display employee names by department:	
manufacturing	Betty Pen
	Craig Wallace
marketing	Gonzo Babbage
	Sarah Dun
	Kermit French
	Beth Hume
sales	Wilma Jones
	Simon Smith

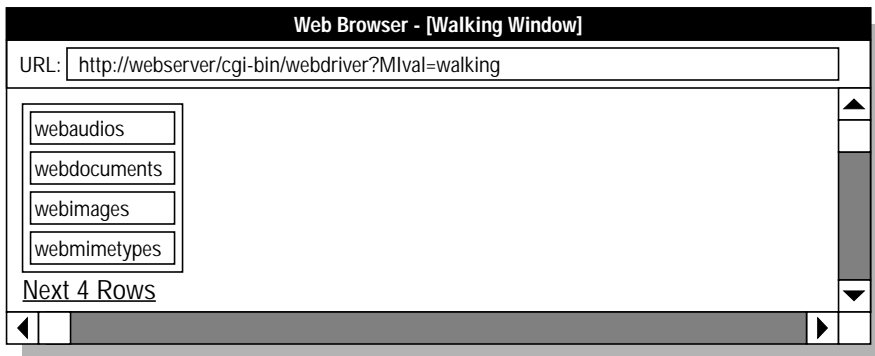
The following **walking** AppPage also uses variable processing functions to format output conditionally. This example queries the **systables** table and displays only the rows that are within the current data window. The display of a row is suppressed when the row is not within the current data window.

```
<HTML>
<HEAD><TITLE>Walking Window</TITLE></HEAD>
<BODY>
<!-- Initialization -->
<?MIVAR NAME=WINSIZE DEFAULT=4>$WINSIZE</MIVAR>
<?MIVAR NAME=BEGIN DEFAULT=0>$START</MIVAR>
<!-- Definition of Ranges ---->
<?MIVAR NAME=BEGIN>$(IF,$(<,$BEGIN,0),0,$BEGIN)</MIVAR>
<?MIVAR NAME=END>$(+,$BEGIN,$WINSIZE)</MIVAR>
<!-- Execution -->
<TABLE BORDER>
<?MISQL SQL="select tabname from systables where tabname like 'web%'
order by tabname;">
    $(IF,$(AND,$(<,$BEGIN,$MI_CURRENTROW),$(>=,$END,$MI_CURRENTROW)),
    <TR><TD>$1</TD></TR>)
</MISQL>
</TABLE>
<BR>
<!-- Set up next range -->
<?MIBLOCK COND=$(<=,$WINSIZE,$BEGIN)>
    <?MIVAR>
    <A HREF=$WEB_HOME?Mival=walking&START=
    $(-,$BEGIN,$WINSIZE)&WINSIZE=$WINSIZE>
    Previous $WINSIZE Rows</A>
    </MIVAR>
</MIBLOCK>
<?MIBLOCK COND=$(<,$END,$MI_ROWCOUNT)>
    <?MIVAR>
    <A HREF=$WEB_HOME?Mival=walking&START=$END&WINSIZE=$WINSIZE>
    Next $WINSIZE Rows</A>
    </MIVAR>
</MIBLOCK>
</BODY>
</HTML>
```

For more information on the **MI\_CURRENTROW** variable used in this AppPage, see [“Processing Variables”](#) on page 4-11.

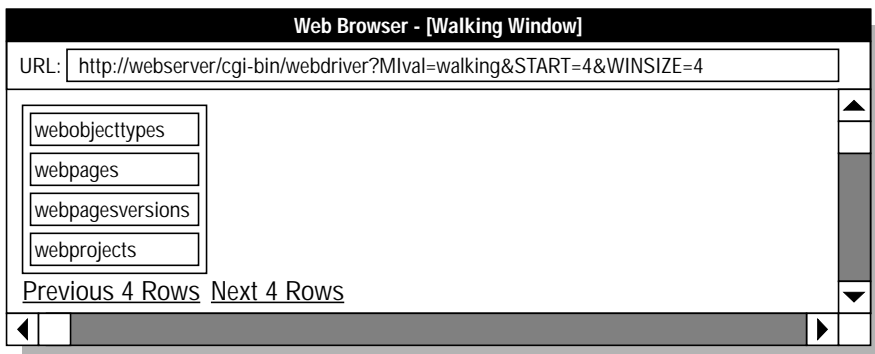
The following shows the Web browser output for the first set of rows retrieved.

**Figure 5-4**  
*Walking Window 1*



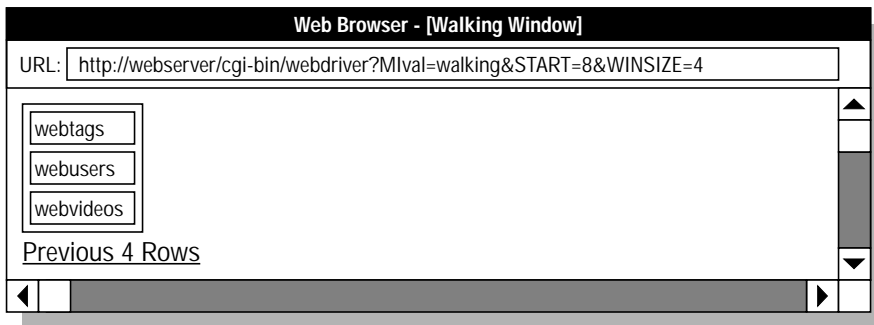
The following shows the Web browser output for the next set of rows retrieved.

**Figure 5-5**  
*Walking Window 2*



The following shows the Web browser output for the final set of rows retrieved.

**Figure 5-6**  
*Walking Window 3*



## Special Characters in Variable Expressions

Quoted strings can be used to suppress evaluation of sequences of characters that otherwise would be interpreted as part of a variable expression. In the following example, double quotes are placed around the string Hello, Citizen to prevent the comma in the string from being treated as a parameter separator:

```
<?MIVAR>$(SUBSTR,"Hello, Citizen",5)<?/MIVAR>
```

Double quotes may be included in quoted strings by placing two adjacent double quotes.

Additionally, since a blank space terminates an attribute assignment, you must place double quotes around any variable expression containing a space, as in the following example:

```
<?MIBLOCK COND="$(EQ,x y,$var1)">Values are equal.<?/MIBLOCK>
```

Since a greater than ( > ) terminates a tag, you must also place double quotes around any variable expression containing greater than:

```
<?MIBLOCK COND="$(>,x,y)">X is greater than Y.<?/MIBLOCK>
```

---

# Using Dynamic Tags in AppPages

Invoking Dynamic Tags in AppPages . . . . .	6-3
Using System Dynamic Tags . . . . .	6-4
CHECKBOXLIST . . . . .	6-4
RADIOLIST . . . . .	6-6
SELECTLIST. . . . .	6-9
Creating User Dynamic Tags . . . . .	6-13
Special Characters in Dynamic Tags . . . . .	6-15



**D**ynamic tags allow AppPage segments to be shared among multiple AppPages, reducing maintenance requirements and centralizing the source of updates to Web applications. Dynamic tags allow you to specify standard components that are used in multiple AppPages, such as headers and footers that appear on every AppPage in your Web application. Since dynamic tags are expanded with the Web DataBlade module tags by **WebExplode** function, changes made to a dynamic tag are automatically applied to all AppPages that include the dynamic tag. In addition to allowing you to create user dynamic tags, the Web DataBlade module provides system dynamic tags.

---

## Invoking Dynamic Tags in AppPages

Specify a dynamic tag using the SGML-processing instruction format `<?tag_info>`. Specify parameters to dynamic tags as tag attributes.

The following example contains the **display\_image** dynamic tag:

```
These are the employees in department 20:<HR>
<CENTER>
<?display_image NAME=$emp_name DEPT=20>
</CENTER>
```

The preceding **display\_image** dynamic tag has two attributes, NAME and DEPT. Dynamic tags accept variables, variable expressions, and constants as parameter values. The COND attribute to Web DataBlade module tags, described in [“COND Attribute” on page 4-18](#), is also a valid attribute for a dynamic tag. The COND attribute specifies a condition that is evaluated before the tag is processed. If the condition is true, the tag is processed.

When dynamic tags are encountered in an AppPage, the body of the dynamic tag is substituted for the tag identifier. If you specify a tag within your AppPage that is not defined in the **webTags** table, no error is generated, and the tag is returned unaffected in the **WebExplode** function output. The names of the Web DataBlade module tags (MISQL, MIVAR, MIBLOCK, and MIERROR) take precedence over the names of dynamic tags.

## Using System Dynamic Tags

The CHECKBOXLIST, RADIOLIST, and SELECTLIST system dynamic tags simplify the creation of check box lists, radio button lists, and selection lists, respectively. You can also create your own user dynamic tags, as described in [“Creating User Dynamic Tags” on page 6-13](#).

### CHECKBOXLIST

The CHECKBOXLIST system dynamic tag creates an HTML check box list based on the attributes you specify. CHECKBOXLIST has the following tag attributes.

Attribute	Mandatory?	Description
NAME	Yes	Specifies the value of the NAME attribute of the check boxes in the check box list.
SQL	Yes	Specifies the SQL statement that will return a list of items to compose the check box list.
CHECKED	No	Specifies the SQL statement that will return a list of items that are initially checked.
CHECKONE	No	Specifies the value of a single item initially checked.
PRE	No	Specifies text that precedes every check box field.
POST	No	Specifies text that follows every check box field. Default is  .



The following example illustrates the use of CHECKBOXLIST to display information about employees based on the **employees** table schema:

```
create table employees
(
  first_name      varchar(40),
  last_name       varchar(40),
  title           varchar(40),
  onsite           boolean,
  department      varchar(40));
```

The following is the **checkboxlist** AppPage:

```
<HTML>
<HEAD> <TITLE> CHECKBOXLIST Example </TITLE></HEAD>
<BODY>
<?MIBLOCK COND=$(XST,$action)>
  <!-- Block to perform update when submitting form --!>
  <?MIVAR NAME=where SEPARATE=', '>$names</MIVAR>
  <?MIVAR NAME=sql_statement1>update employees set onsite='t'
    where first_name in ('$where');</MIVAR>
  <?MISQL SQL="$sql_statement1"></MISQL>
  <?MIVAR NAME=sql_statement2>update employees set onsite='f'
    where first_name not in ('$where');</MIVAR>
  <?MISQL SQL="$sql_statement2"></MISQL>
<?/MIBLOCK>
<H3> Employees that work onsite: </H3>
<FORM METHOD=POST ACTION=<?MIVAR>$WEB_HOME</MIVAR>>
<!-- Hidden Fields --!>
<INPUT TYPE=hidden NAME=action VALUE=on>
<INPUT TYPE=hidden NAME=Mival VALUE=checkboxlist>
<!-- SQL to generate check box list --!>
<?CHECKBOXLIST NAME=names SQL="select first_name from employees order by
  first_name" CHECKED="select first_name from employees where onsite='t'">
<P>
Control-click names to toggle on and off. Then choose Submit.
<P>
<INPUT TYPE=SUBMIT VALUE="Submit">
<INPUT TYPE=RESET VALUE="Reset">
</FORM>
<HR>
<?MIVAR COND=$(XST,$action)>
  SQL executed: <I>$sql_statement2</I></MIVAR>
<P>
<?MIVAR COND=$(XST,$action)>
  SQL executed: <I>$sql_statement1</I></MIVAR>
<P>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 6-1**  
CHECKBOXLIST

**Web Browser - [CHECKBOXLIST Example]**

URL:

**Employees that work onsite:**

☐ Beth  
☒ Betty  
☐ Craig  
☒ Gonzo  
☐ Kermit  
☐ Sarah  
☐ Simon  
☐ Wilma

Control-click names to toggle on and off. Then choose Submit.

---

SQL executed: *update employees set onsite='f' where first\_name not in ('Betty', 'Gonzo');*

SQL executed: *update employees set onsite='t' where first\_name in ('Betty', 'Gonzo');*

## RADIOLIST

The RADIOLIST system dynamic tag creates an HTML radio button list based on the attributes you specify. RADIOLIST has the following tag attributes.

Attribute	Mandatory?	Description
NAME	Yes	Specifies the value of the NAME attribute of the radio buttons in the radio button list.
SQL	Yes	Specifies the SQL statement that will return a list of items to compose the radio button list.
CHECKED	No	Specifies the SQL statement that will return a single item that is initially checked.
CHECKONE	No	Specifies the value of a single item initially checked.
PRE	No	Specifies text that precedes every radio button field.
POST	No	Specifies text that follows every radio button field. Default is  .



**Tip:** By definition, a radio button list can have only one item selected at a time.

The following example illustrates the use of RADIOLIST to display information about employees based on the **employees** table schema:

```
create table employees
(
  first_name      varchar(40),
  last_name       varchar(40),
  title           varchar(40),
  onsite          boolean,
  department      varchar(40));
```

The following is the **radiolist** AppPage:

```
<HTML>
<HEAD> <TITLE> RADIOLIST Example </TITLE></HEAD>
<BODY>
<?MIBLOCK COND=$(XST,$action)>
  <!-- Block to perform update when submitting form -->
  <H3> Details for Employee <?Mivar>$name</MIVAR>: </H3>
  <?MIVAR NAME=sql_statement> select * from employees
    where first_name = '$name';</MIVAR>
  <?MISQL SQL="$sql_statement">
  <B> Name: </B> $1 $2 <BR>
  <B> Title: </B> $3 <BR>
  <B> Onsite: </B> $4 <BR>
  <B> Department: </B> $5 <BR>
  </MISQL>
</MIBLOCK>
<FORM METHOD=POST ACTION=<?MIVAR>$WEB_HOME</MIVAR>>
<H3> Choose an Employee </H3>
<!-- Hidden Fields -->
<INPUT TYPE=hidden NAME=action VALUE=on>
<INPUT TYPE=hidden NAME=Mival VALUE=radiolist>
<?MIVAR NAME=name DEFAULT="">$name</MIVAR>
<!-- SQL to generate radio button list -->
<?RADIOLIST NAME=name SQL="select first_name from employees
  order by first_name" CHECKONE="Betty">
<P>
Select a name. Then choose Submit.
<P>
<INPUT TYPE=SUBMIT VALUE="Submit">
<INPUT TYPE=RESET VALUE="Reset">
</FORM>
<HR>
<?MIVAR COND=$(XST,$action)>
  SQL executed: <I>$sql_statement</I></MIVAR>
<P>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 6-2**  
RADIOLIST

Web Browser - [RADIOLIST Example]

URL:

Details for Employee Betty:

Name: Betty Pen  
Title: Senior Line Worker  
Onsite: t  
Department: manufacturing

**Choose an Employee**

☐ Beth  
☒ Betty  
☐ Craig  
☐ Gonzo  
☐ Kermit  
☐ Sarah  
☐ Simon  
☐ Wilma

Select a name. Then choose Submit.

SELECTLIST

The SELECTLIST system dynamic tag creates an HTML selection list based on the attributes you specify. SELECTLIST has the following tag attributes.

Attribute	Mandatory?	Description
NAME	Yes	Specifies the value of the NAME attribute of the items in the selection list.
SQL	Yes	Specifies the SQL statement that will return a list of items to compose the selection list.
MULTIPLE	No	If specified, users can make multiple selections.
SELECTED	No	Specifies the SQL statement that will return a list of items that are initially selected.
SELECTONE	No	Specifies the value of a single item initially selected.
SIZE	No	Specifies the number of visible choices.

The following example illustrates the use of SELECTLIST to display information about employees based on the **employees** table schema:

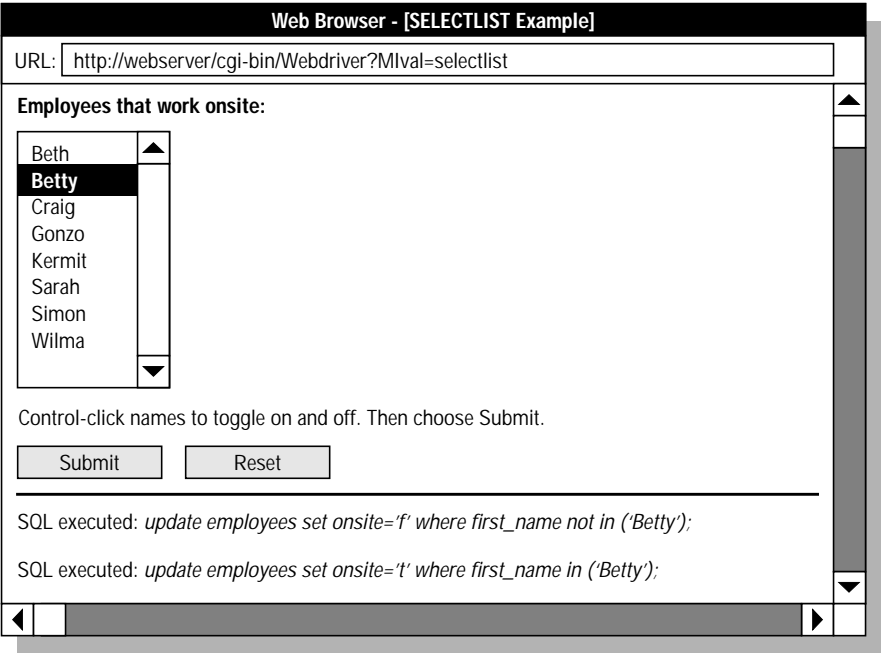
```
create table employees
(
first_name      varchar(40),
last_name       varchar(40),
title           varchar(40),
onsite          boolean,
department      varchar(40));
```

The following is the **selectlist** AppPage:

```
<HTML>
<HEAD> <TITLE> SELECTLIST Example </TITLE></HEAD>
<BODY>
<?MIBLOCK COND=$(XST,$action)>
  <!-- Block to perform update when submitting form ---!>
  <?MIVAR NAME=where SEPARATE=', '>$names</MIVAR>
  <?MIVAR NAME=sql_statement1>update employees set
    onsite='t' where first_name in ('$where');</MIVAR>
  <?MISQL SQL="$sql_statement1"></MISQL>
  <?MIVAR NAME=sql_statement2>update employees set
    onsite='f' where first_name not in ('$where');</MIVAR>
  <?MISQL SQL="$sql_statement2"></MISQL>
</MIBLOCK>
<H3> Employees that work onsite: </H3>
<FORM METHOD=POST ACTION=<?MIVAR>$WEB_HOME</MIVAR>>
<!-- Hidden Fields ---!>
<INPUT TYPE=hidden NAME=action VALUE=on>
<INPUT TYPE=hidden NAME=Mival VALUE=selectlist>
<!-- SQL to generate selection list ---!>
<?SELECTLIST NAME=names SIZE=8 MULTIPLE
  SQL="select first_name from employees order by first_name"
  SELECTED="select first_name from employees where onsite='t'">
<P>
Control-click names to toggle on and off. Then choose Submit.
<P>
<INPUT TYPE=SUBMIT VALUE="Submit">
<INPUT TYPE=RESET VALUE="Reset">
</FORM>
<HR>
<?MIVAR COND=$(XST,$action)>
  SQL executed: <I>$sql_statement2</I></MIVAR>
<P>
<?MIVAR COND=$(XST,$action)>
  SQL executed: <I>$sql_statement1</I></MIVAR>
<P>
</BODY>
</HTML>
```

The following is sample Web browser output.

Figure 6-3  
SELECTLIST





## Creating User Dynamic Tags

User dynamic tag definitions, like system dynamic tag definitions, are stored in the **webTags** table in the database. The **webTags** schema is shown in the following table.

Column	Description
ID	Unique identifier for the dynamic tag.
parameters	List of parameters to the dynamic tag.
class	Class of dynamic tag. For example, you can have <code>beginning</code> , <code>expert</code> , or any other class name. System dynamic tags have the class name <code>system</code> , and cannot be modified in AppPage Builder. Default class is <code>user</code> .
description	Description of the dynamic tag.
content	Body of the dynamic tag.

Parameters in the **parameters** column are separated by an ampersand ( & ). You can assign a default value to a parameter by specifying the parameter and its value as a name/value pair, for example, `param1=value1`. A parameter that does not need a default value is specified by the parameter followed by an equal sign ( = ) with no value following, for example, `param1=`.

Parameters are delimited by a commercial at ( @ ) before and after the parameter name within the body of the dynamic tag.

When a dynamic tag is inserted or updated in the **webTags** table, the tag is verified to check that all of the parameters in the **content** column (delimited by the @ character) are also listed in the **parameters** column. When a dynamic tag is encountered in an AppPage, the tag is verified to check that all parameters requiring a value are assigned a value.

You can add, edit, and delete dynamic tags (other than those of class `system`) in AppPage Builder. Following the definition of the dynamic tag, the tag can be invoked in any AppPage.

The following IMG dynamic tag, which invokes the standard HTML IMG tag based on information retrieved in a SELECT statement, displays the image identified by the mandatory SRC parameter. HEIGHT, WIDTH, and ALIGN are all optional parameters.

Figure 6-4  
Dynamic Tag

Web Browser - [APB - Add Dynamic Tag]

URL:

Tag ID:

Class:

Parameters:

Description:

Dynamic Tag:

For example, if your AppPage contains the dynamic tag <?IMG SRC=apb\_logo>, sample output to the client is:

```
<IMG SRC=/cgi-bin/webdriver?LO=lo_handle&MItypeObj=image/gif ALT="APB Logo"
HEIGHT=40
WIDTH=335>
```

If your AppPage contains the dynamic tag

<?IMG SRC=apb\_logo ALIGN=CENTER>, sample output to the client is:

```
<IMG SRC=/cgi-bin/webdriver?LO=lo_handle&MItpeObj=image/gif ALT="APB Logo"
HEIGHT=40
WIDTH=335
ALIGN=CENTER>
```

If your AppPage contains the dynamic tag

<?IMG SRC=apb\_logo COND=\$(XST,\$DISPLAY)>, the IMG tag is generated only if the DISPLAY variable has been assigned a value within that AppPage.

**Warning:** The body of a dynamic tag can contain another dynamic tag. Do not call the same dynamic tag recursively, or you might consume all your system resources.



## Special Characters in Dynamic Tags

You must replace the @ character with its entity reference if the character occurs within your dynamic tag content.

| Character | Entity Reference |
|-----------|------------------|
| @         | &#64;            |



---

# Using Server Functions in AppPages

|                        |      |
|------------------------|------|
| WebExplode . . . . .   | 7-4  |
| WebLint . . . . .      | 7-7  |
| WebRelease . . . . .   | 7-10 |
| WebUnHTML . . . . .    | 7-12 |
| WebURLDecode . . . . . | 7-14 |
| WebURLEncode . . . . . | 7-15 |



**T**he **WebExplode** server function executes Web DataBlade module tags within AppPages to dynamically retrieve database content. Additional Web DataBlade module server functions provide features that are commonly required by Web application designers. The following server functions are described in this chapter:

- **WebExplode** returns the specified AppPage with all the Web DataBlade module tags expanded and SQL results dynamically retrieved.
- **WebLint** reports syntax errors in Web DataBlade module tags in the specified AppPage.
- **WebRelease** returns the release identifier of the Web DataBlade module.
- **WebUnHTML** returns the specified text with special HTML characters replaced for display by a Web browser.
- **WebURLDecode** returns the specified text with hexadecimal values replaced with their nonalphanumeric ASCII characters.
- **WebURLEncode** returns the specified text with nonalphanumeric ASCII characters replaced with their hexadecimal values.

You can write additional server functions to simplify Web application design.



## WebExplode

The **WebExplode** function expands Web DataBlade module tags within an AppPage and retrieves SQL results dynamically.

***Tip:** Because **WebExplode** is a server function, all SQL statements within an AppPage are executed as a single transaction block.*

## Syntax

```
WebExplode(HTML, HTML)
```

## Arguments

The first HTML argument is an AppPage.

The second HTML argument specifies any variables passed to **WebExplode** by the calling application as name/value pairs (for example, name1=value1&name2=value2...).

***Important:** When you call **WebExplode**, all variable assignments are inherited from the parent process (usually Webdriver). Variables are global in scope. Therefore, if you override the assignment of a variable in the second argument of your call to **WebExplode** or within the AppPage you execute, that variable assignment is retained until you reassign it elsewhere.*



## Returns

**WebExplode** returns an AppPage as HTML.

## Example

The following example illustrates the use of the **WebExplode** function.



## To create an AppPage table and retrieve data dynamically using WebExplode

### 1. Create the **web\_apps** table to store AppPages:

```
create table web_apps
(
  app_id      varchar(40) NOT NULL,
  app_desc    varchar(64),
  app_frm     html,
  primary key (app_id)
);
```

### 2. Create the **employees** table to store employee data:

```
create table employees
(
  first_name   varchar(40),
  last_name    varchar(40),
  title        varchar(40),
  onsite        boolean,
  department   varchar(40));
```

### 3. Load data into the **employees** table.

### 4. Insert an AppPage into the **web\_apps** table. The **emp\_list** AppPage contains an MISQL tag that retrieves data from the **employees** table:

```
insert into web_apps values
(
  'emp_list',
  'employee listing',
  '<HTML>
<HEAD><TITLE>Employee List</TITLE></HEAD>
<BODY>
<H2>Current list of employees and job titles:</H2>
<?MISQL SQL="select first_name, last_name, title from
  employees;">
<B>$1 $2</B> $3 <BR>
<?/MISQL>
</BODY>
</HTML>'
);
```

5. When you retrieve the AppPage using **WebExplode**, the query within the MISQL tag is executed, and the results are formatted as HTML according to the specifications in the MISQL tag:

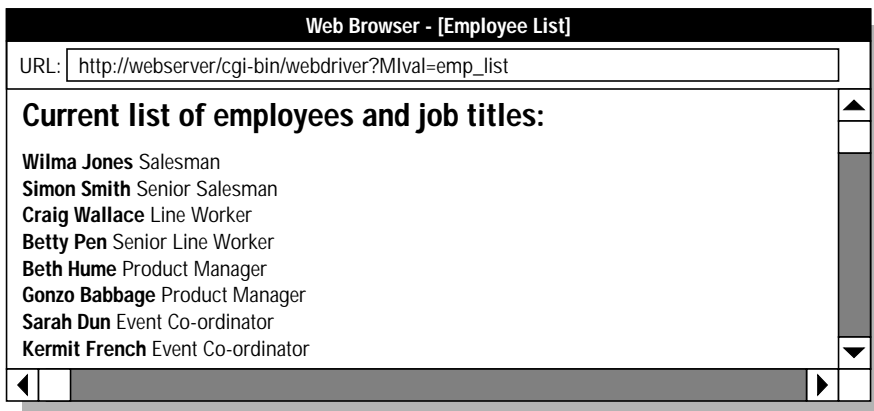
```
select WebExplode(app_frm, '') from web_apps where
app_id = 'emp_list';
```

**WebExplode** returns the following HTML:

```
<HTML>
<HEAD><TITLE>Employee List</TITLE></HEAD>
<BODY>
<H2>Current list of employees and job titles:</H2>
<B>Gonzo Babbage</B> Product Manager <BR>
<B>Betty Pen</B> Senior Line Worker <BR>
<B>Craig Wallace</B> Line Worker <BR>
<B>Sarah Dun</B> Event Co-ordinator <BR>
<B>Kermit French</B> Event Co-ordinator <BR>
<B>Wilma Jones</B> Salesman <BR>
<B>Simon Smith</B> Senior Salesman <BR>
<B>Beth Hume</B> Product Manager <BR>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 7-1**  
WebExplode





## WebLint

The **WebLint** function scans an AppPage and reports syntax errors within Web DataBlade module tags.

*Tip: **WebLint** does not evaluate dynamic tags.*

## Syntax

```
WebLint(HTML, INTEGER)
```

## Arguments

The first HTML argument is an AppPage.

The second INTEGER argument represents the level of checking to be performed. Levels of checking are described in the following table.

| Level | Description  |
|-------|--|
| 0     | Returns PASS or FAIL. Checking stops as soon as an error is encountered.   |
| 1     | Returns PASS or error text describing the first error encountered.   |
| 2     | Returns PASS or error text describing all errors encountered.  |
| 3     | Same error processing as level 2, with additional checks on variables. Issues a warning if a value is not assigned to a variable within the AppPage. |

## Returns

**WebLint** returns error messages as LVARCHAR.

## Example

The following SELECT statement executes **WebLint** against the **title** AppPage in the **webPages** table:

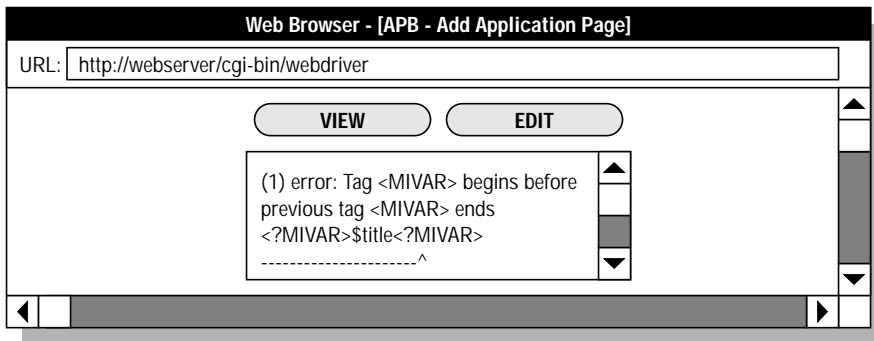
```
select WebLint(object, 1) from webPages where ID = 'title';
```

If the **title** AppPage contains the following HTML, with a missing slash ( / ) in the end MIVAR tag,

```
<TITLE>
<?MIVAR>$title<?MIVAR>
</TITLE>
```

the following error message is displayed by **WebLint** when the level of checking is greater than 0.

**Figure 7-2**  
WebLint



**Tip:** You can attempt to execute an AppPage even if **WebLint** reports errors in the AppPage.

You can execute **WebLint** against a file that contains an AppPage directly from the operating system prompt. Execute the **weblint** command from the **\$INFORMIXDIR/extend/web.3.30.UC1/utlis** directory, or add the **\$INFORMIXDIR/extend/web.3.30.UC1/utlis** directory to your path. Then enter the following command:

```
weblint [level] < AppPage_file
```

**or:**

```
cat AppPage_file | weblint [level]
```

---

## WebRelease

The **WebRelease** function returns the version of the Web DataBlade module.

### Syntax

```
WebRelease()
```

### Arguments

None

### Returns

**WebRelease** returns the version number and date of the Web DataBlade module as LVARCHAR.

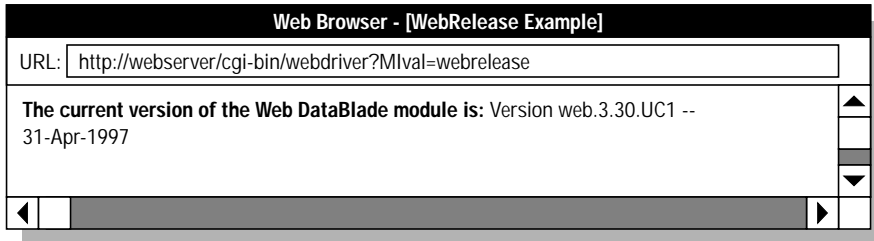
### Example

The following **webrelease** AppPage calls the **WebRelease** function to display the version number and date of the Web DataBlade module:

```
<HTML>
<HEAD><TITLE>WebRelease Example</TITLE></HEAD>
<BODY>
<B>The current version of the Web DataBlade module is:</B>
<?MISQL SQL="execute function WebRelease();">$1<?/MISQL>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 7-3**  
WebRelease



---

## WebUnHTML

The **WebUnHTML** function returns HTML with special HTML characters replaced with their entity reference for display by a Web browser. **WebUnHTML** scans the AppPage, and makes the following replacements.

| Character | Entity Reference |
|-----------|------------------|
| <         | &lt;             |
| >         | &gt;             |
| "         | &quot;           |
| &         | &amp;            |

This substitution allows the HTML tag information to be displayed by a Web browser. If this action is not taken, the browser uses these characters in its attempt to render the HTML tags as formatting information.

## Syntax

```
WebUnHTML (HTML)
```

## Arguments

The argument is HTML.

## Returns

**WebUnHTML** returns HTML, with HTML codes replaced, as HTML.



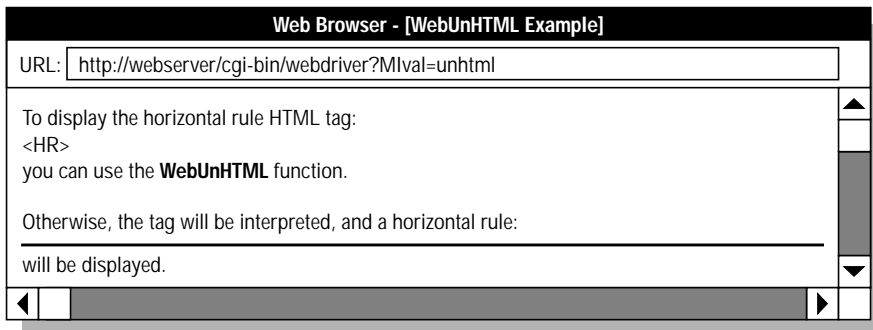
## Example

The following **unhtml** AppPage uses **WebUnHTML** to display HTML tags within the AppPage:

```
<HTML>
<HEAD><TITLE>WebUnHTML Example</TITLE></HEAD>
<BODY>
To display the horizontal rule HTML tag: <BR>
<?MISQL SQL="execute function
WebUnHTML('<HR>');">$1<?/MISQL>
<BR>
you can use the <B>WebUnHTML</B> function. <BR> <BR>
Otherwise, the tag will be interpreted, and a horizontal rule:
<HR>
will be displayed.
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 7-4**  
WebUnHTML



---

## WebURLDecode

The **WebURLDecode** function returns HTML with hexadecimal values replaced with their nonalphanumeric ASCII characters, and plus signs ( + ) replaced with spaces.

### Syntax

```
WebURLDecode(HTML)
```

### Arguments

The argument is HTML.

### Returns

**WebURLDecode** returns URL-encoded HTML as HTML.

### Example

Since **WebExplode** decodes information passed in URLs, you do not normally need to decode the URL yourself.

## WebURLEncode

The **WebURLEncode** function returns HTML with nonalphanumeric ASCII characters replaced with their hexadecimal values, and spaces replaced with a plus sign ( + ).

### Syntax

```
WebURLEncode(HTML)
```

### Arguments

The argument is HTML.

### Returns

**WebURLEncode** returns HTML as URL-encoded HTML.

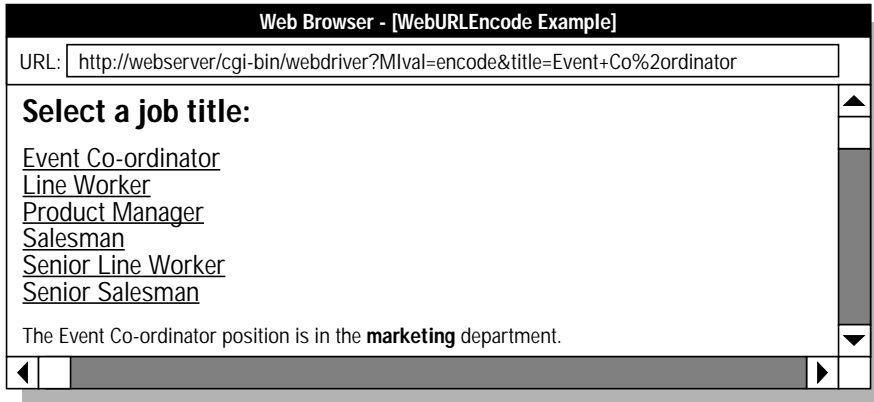
### Example

The following **encode** AppPage uses the **WebURLEncode** function to encode job titles, which may contain spaces, for use in URLs. If you do not encode text within links, and the text contains spaces, the links will not function properly.

```
<HTML>
<HEAD> <TITLE>WebURLEncode Example</TITLE> </HEAD>
<BODY>
<H2>Select a job title:</H2>
<?MISQL SQL="select distinct title, WebURLEncode(title)
      from employees order by title;">
<A HREF=$WEB_HOME?Mival=encode&title=$2>$1</A><BR>
<?/MISQL>
<?MIBLOCK COND=$(XST,$title)>
<?MISQL SQL="select distinct department from employees
      where title='$title';">
<BR>The $title position is in the <B>$1</B> department.<BR>
<?/MISQL>
<?/MIBLOCK>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 7-5**  
WebURLEncode



---

# Using Advanced Webdriver Features

|  |      |
|--|------|
| Adding HTTP Headers to AppPages. . . . .                         | 8-3  |
| Retrieving Non-HTML Pages . . . . .                              | 8-4  |
| Using Cookies . . . . .  | 8-4  |
| Setting Cookies . . . . .  | 8-5  |
| Converting Cookies into Web DataBlade Module Variables . . . . . | 8-5  |
| Uploading Client Files. . . . .                                  | 8-7  |
| Passing Image Map Coordinates . . . . .                          | 8-10 |
| IMG Tag . . . . .  | 8-11 |
| FORM Tag . . . . .   | 8-12 |
| Configuring Webdriver Connection Settings . . . . .              | 8-14 |



**I**n addition to providing the CGI or NSAPI interface to a Web server, Webdriver:

- enables you to process errors for error messages returned by the **WebExplode** function. Webdriver displays errors generated by the MIERROR tag and by INFORMIX-Universal Server, as described in [“Processing Errors with Webdriver” on page 4-25](#).
- enables you to retrieve large objects, as described in [“Retrieving Large Objects” on page 2-13](#).
- enables you to cache large objects, as described in [“Caching Large Objects Using Webdriver” on page 9-3](#).
- enables you to add HTTP headers to AppPages to retrieve non-HTML pages and use cookies.
- enables you to upload client files for Web browsers that support the ENCTYPE attribute of the FORM tag.
- enables you to pass image map coordinates.
- enables you to configure connection parameters.

The final four of these list items are described in this chapter.

---

## Adding HTTP Headers to AppPages

Webdriver enables you to use HTTP headers in your AppPages to:

- retrieve non-HTML pages.
- use cookies.

## Retrieving Non-HTML Pages

To change the content type of an AppPage, add an HTTP header to the AppPage to replace the default `text/html` content type header. Use the following syntax within a variable expression to set the content type:

```
$(HTTPHEADER,content-type,mimetype/subtype)
```

In all cases, Webdriver adds a `content-length` header to the page, because only Webdriver can determine the size of the page.

The following is a sample plain text page, stored in the Micol column of your Web application table:

```
This is a plain text page.  
<?MIVAR>$(HTTPHEADER,content-type,text/plain)<?/MIVAR>  
It is displayed without rendering any HTML tags,  
so that characters such as "<HR>" appear normally,  
and are not treated as markup tags.
```

The resulting HTTP response to the Web browser is:

```
Content-length: 222  
Content-type: text/plain
```

```
This is a plain text page.
```

```
It is displayed without rendering any HTML tags,  
so that characters such as "<HR>" appear normally,  
and are not treated as markup tags.
```

**Tip:** The *HTTPHEADER* variable expression can be placed anywhere within the AppPage.

## Using Cookies

Cookies are a general mechanism used by Web-server-side connections (such as Webdriver) to store and retrieve information on the client side of the connection (such as your Web browser). You can set cookies in your AppPages and then convert cookies into Web DataBlade module variables.





## ***Setting Cookies***

You can set cookies on any AppPage by adding an HTTP header to the AppPage, for example:

```
$(HTTPHEADER,set-cookie,name=value)
```

You can set additional attributes in the second parameter to the HTTPHEADER variable expression, for example:

```
$(HTTPHEADER,set-cookie,name=value; expires=DATE; path=PATH; domain=DOMAIN_NAME)
```

For more information on cookies, refer to the following URL:

```
http://home.netscape.com/newsref/std/cookie\_spec.html
```

## ***Converting Cookies into Web DataBlade Module Variables***

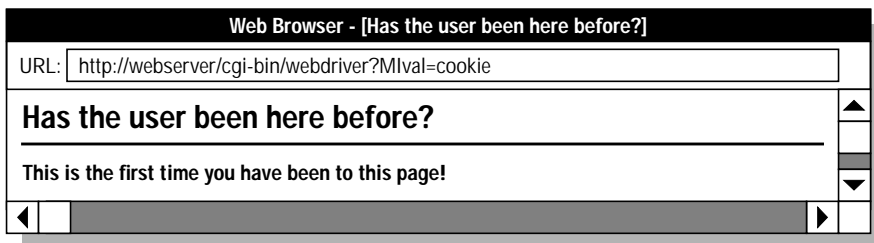
When a cookie is set for a Web browser, the cookie is passed back to the Web server for each request made by that same Web browser. The Web DataBlade module automatically takes any cookies it receives and converts them into Web DataBlade module variables. The Web server environment variable HTTP\_COOKIE is detected by Webdriver and parsed into variables so that the HTTP\_COOKIE variable is never seen in an AppPage.

The following **cookie** AppPage determines whether or not the Web browser has retrieved this AppPage previously. The first time the Web browser retrieves the **cookie** AppPage, the AppPage sends a cookie to the Web browser, which the Web browser keeps even if it retrieves other HTML pages before retrieving this AppPage again. On any subsequent retrieval of this AppPage, the Welcome Back! message is displayed.

```
<HTML>
<HEAD><TITLE>Has the user been here before?</TITLE></HEAD>
<BODY>
<H2>Has the user been here before?</H2><HR>
<!-- See if the flag variable has been set -->
<?MIBLOCK COND=$(XST,$flag)>
    <!-- Flag variable has been set -->
    <B>Welcome Back! You have been here before!</B>
<?/MIBLOCK>
<?MIBLOCK COND=$(NXST,$flag)>
    <!-- Flag variable has NOT been set -->
    <!-- Set a cookie -->
    <?MIVAR>$(HTTPHeader,set-cookie,flag=yes)<?/MIVAR>
    <B>This is the first time you have been to this page!</B>
<?/MIBLOCK>
</BODY>
</HTML>
```

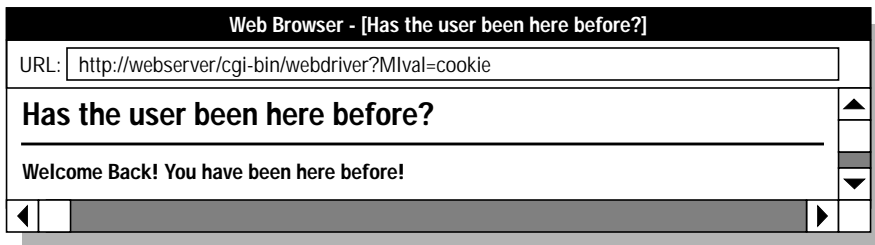
The following is sample Web browser output the first time the **cookie** AppPage is retrieved.

**Figure 8-1**  
*Cookie - First Request*



The following is sample Web browser output on any subsequent retrieval of the **cookie** AppPage.

**Figure 8-2**  
Cookie - Subsequent Request



## Uploading Client Files

If you use a Web browser that supports client file upload, you can use Webdriver to upload files from your client machine.



**Important:** *At the time that this guide was printed, only the Netscape 3.0 browser supported client file upload (using the ENCTYPE attribute of the FORM tag). However, this file upload capability has been proposed as part of future HTML standards.*

The following is an example HTML form that retrieves an image file into the **input\_image1** variable and is processed by the **process\_file** AppPage:

```
<FORM ENCTYPE=multipart/form-data METHOD=POST> ACTION=<?MIVAR>$WEB_HOME<?/MIVAR>>
<INPUT TYPE=TEXT NAME=file_name>
<INPUT TYPE=FILE NAME=input_image1>
<INPUT TYPE=SUBMIT VALUE="Send File">
<INPUT TYPE=HIDDEN NAME=Mlval VALUE=process_file>
</FORM>
```

Set the following **web.cnf** file variable to upload client files.

| Variable        | Mandatory? | Content  |
|-----------------|------------|--|
| MI_WEBUPLOADDIR | No         | Directory on the Web server machine in which uploaded files are placed. Default is /tmp. |

Set MI\_WEBUPLOADDIR to the directory on the Web server machine where the uploaded files are to be placed. In the preceding example, if MI\_WEBUPLOADDIR is set to /local/Web/uploads, the file **/local/Web/uploads/input\_image1.PID** (where *PID* is the process ID for the Webdriver process) is created when the form is submitted. If MI\_WEBUPLOADDIR is not set, the uploaded files are placed in the **/tmp** directory by default. After Webdriver finishes processing the AppPage, the uploaded file is removed from the MI\_WEBUPLOADDIR directory.

When the form is submitted, you can access the following variables in the AppPage that processes the form.

| Variable Name          | Description   |
|------------------------|---|
| <i>input_file</i>      | Full pathname of the uploaded file on the Web server machine. |
| <i>input_file_name</i> | Full pathname of the client file.                             |
| <i>input_file_type</i> | MIME type of the uploaded file (may be unknown).              |

In the preceding example, if the client file is named **D:\images\input\_image.gif**, the following variables are accessible in the **process\_file** AppPage.

| Variable Name     | Assignment                          |
|-------------------|-------------------------------------|
| input_image1      | /local/Web/uploads/input_image1.PID |
| input_image1_name | D:\images\input_image.gif           |
| input_image1_type | image/gif                           |

Use the **FileToBlob** function to create a large object from the uploaded image. For more information about large objects, see [Informix Guide to SQL: Reference](#).

If Webdriver is unable to write the file to the directory specified by MI\_WEBUPLOADDIR, the value of the file variable is set to MI\_ERROR.

The following example illustrates the use of client file upload where uploaded files are stored in the **uploads** table with the schema:

```
CREATE TABLE uploads
(
  name          varchar(40),
  object_type   varchar(40),
  object        blob,
  local_file    varchar(100))
put object in (mysblobs);
```

The following is the **upload\_file** AppPage:

```
<HTML>
<HEAD><TITLE>File Upload Form</TITLE></HEAD>
<BODY>
<HR>
<FORM ENCTYPE=multipart/form-data METHOD=POST ACTION=<?MIVAR>$WEB_HOME</MIVAR>>
<INPUT TYPE=HIDDEN NAME=Mival VALUE=upload_file>
<INPUT TYPE=HIDDEN NAME=action VALUE=on>
<TABLE>
<TR><TD>Name: </TD><TD><INPUT NAME=name SIZE=40 TYPE=TEXT>
</TD></TR>
<TR><TD>File: </TD><TD><INPUT NAME=upload SIZE=40 TYPE=FILE>
</TD></TR>
</TABLE>
<HR>
<INPUT TYPE=SUBMIT VALUE="Insert New Object">
</FORM>
<?MIBLOCK COND=$(XST,$action)>
  <HR>
  <?MIVAR NAME=sql_statement>
  INSERT into uploads VALUES
  ('$name', '$upload_type', FileToBlob('$upload','client','uploads','object'),
  '$upload_name');
  </MIVAR>
  <?MYSQL SQL="$sql_statement">
  Inserted $MI_ROWCOUNT new objects.<P><?/MYSQL>
  <?MIVAR>The SQL executed was <I>$sql_statement</I>.<P><?/MIVAR>
<?/MIBLOCK>
<B> Here are all of the uploaded objects:</B>
<TABLE>
<?MYSQL SQL="select name, object, object_type, local_file from uploads;">
<TR><TD><A HREF="$WEB_HOME?LO=$2&type=$3">$1</A>
</TD><TD>$4</TD></TR>
<?/MYSQL>
</TABLE>
</BODY>
</HTML>
```

The following is sample Web browser output.

**Figure 8-3**  
File Upload Form

Web Browser - [File Upload Form]

URL: http://webserver/cgi-bin/webdriver

Name: graylogo

File: C:\LOGOS\GRAYLOGO.gif

Browse...

Insert New Object

Inserted 1 new objects.

The SQL executed was *INSERT into uploads VALUES ('graylogo', 'image/gif', FileToBlob('/tmp/upload.008987', 'client', 'uploads', 'object'), 'C:\LOGOS\GRAYLOGO.gif');*

Here are all of the uploaded objects:

bookmark C:\NETSCAPE\Program\Bookmark.htm

graylogo C:\LOGOS\GRAYLOGO.gif

## Passing Image Map Coordinates

Set the following variable to enable image map coordinates to be passed to AppPages.

| Variable | Mandatory? | Content   |
|----------|------------|---|
| MImap    | Yes        | Set to on or off. When on, the URL is treated as an image map, and the values are passed as x- and y-coordinates. Default is off. |

**Important:** MImap must be set in the URL that invokes the AppPage. MImap must not be set in the web.cnf file.



There are two methods for passing image map coordinates within a Web DataBlade module application. You can pass coordinates with Webdriver by:

- using the ISMAP attribute of the IMG tag.
- using an HTML form.

## IMG Tag

To pass *x*- and *y*-coordinates through Webdriver, set the MImap variable to `on` in the URL that calls the AppPage to which the coordinates are passed. This prevents the coordinates from being overridden in the URL when you use the ISMAP attribute of the IMG tag to create an image map.

When MImap is set to `on` in PATH\_INFO (the portion of a URL consisting of name/value pairs following the path name and preceding the `?`), Webdriver parses the QUERY\_STRING (the portion of a URL following the `?`) into two variables, called *x\_value* and *y\_value*, which hold the values from the image map. An example image map URL is:

```
http://myhost:port/cgi-bin/webdriver/MImap=on&MVal=image_example?100,13
```

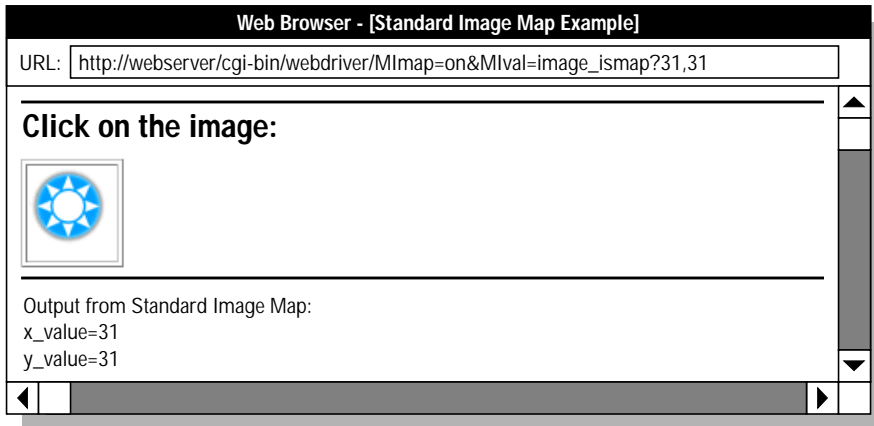
You can then access *x\_value* and *y\_value* (in this example, 100 and 13, respectively) in the same way that you access other variables. The following **image\_ismap** AppPage illustrates the use of image maps with the IMG tag:

```
<HTML>
<HEAD><TITLE>Standard Image Map Example</TITLE></HEAD>
<BODY>
<H2>Click on the image:</H2>
<TABLE BORDER>
<TR><TD VALIGN="top">
<!-- Display the image as an image map -->
<A HREF= "<?MIVAR>$WEB_HOME<?/MIVAR>/MImap=on&MVal=image_ismap">
<IMG BORDER=0
SRC="<?MIVAR>$WEB_HOME<?/MIVAR>?MValObj=sun&MTypeObj=image/gif" ISMAP></A>
</TD></TR></TABLE><HR>
<!-- Show resulting coordinates from the image -->
<!-- Output values x_value and y_value -->
<!-- if the standard image is clicked. -->
<?MIBLOCK COND=$(XST,$x_value)>
Output from Standard Image Map:<BR>
<?MIVAR>x_value = $x_value<?/MIVAR><BR>
<?MIVAR>y_value = $y_value<?/MIVAR>
<?/MIBLOCK>
</BODY>
</HTML>
```



The following shows sample Web browser output.

**Figure 8-4**  
*Standard Image Map*



## FORM Tag

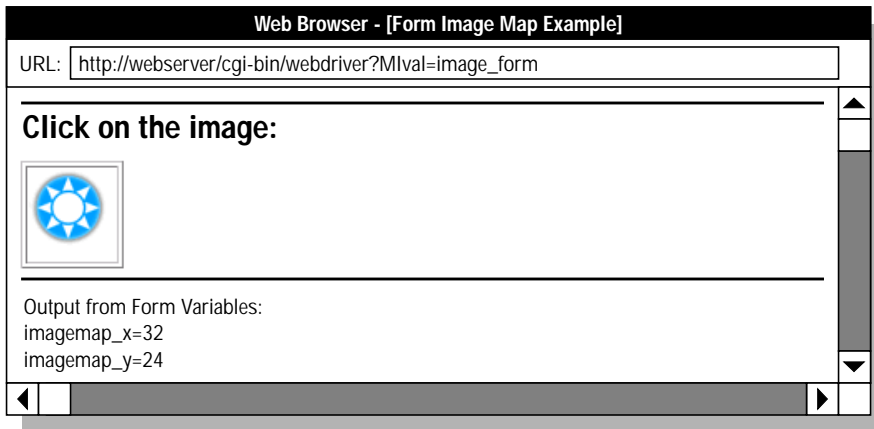
To pass *x*- and *y*-coordinates in HTML forms, the **WebExplode** function converts the period ( . ) to underscore ( \_ ) in variable names. You can then access the *variable\_x* and *variable\_y* variables within your AppPages.

The following **image\_form** AppPage illustrates the use of image maps with an HTML form:

```
<HTML>
<HEAD><TITLE>Form Image Map Example</TITLE></HEAD>
<BODY>
<H2>Click on the image:</H2>
<TABLE BORDER>
<TR><TD VALIGN="top">
<!-- Display the image as an input for a form -->
<FORM METHOD="POST" ACTION="<?MIVAR>$WEB_HOME</MIVAR>">
<INPUT TYPE=HIDDEN NAME=Mival VALUE="image_form">
<INPUT NAME="imagemap" TYPE="image" BORDER=0
      SRC="<?MIVAR>$WEB_HOME</MIVAR>?MivalObj=sun&MitypeObj=image/gif">
</FORM>
</TD></TR>
</TABLE>
<!-- Output imagemap_x and imagemap_y if a form --->
<HR>
<?MIBLOCK COND=$(XST,$imagemap_x)>
  Output from Form Variables:<BR>
  <?MIVAR>imagemap_x = $imagemap_x</MIVAR><BR>
  <?MIVAR>imagemap_y = $imagemap_y</MIVAR>
<?/MIBLOCK>
</BODY>
</HTML>
```

The following shows sample Web browser output.

**Figure 8-5**  
Form Image Map



## Configuring Webdriver Connection Settings

The following table lists **web.cnf** file settings for Webdriver connections to INFORMIX-Universal Server.

| Variable         | Mandatory? | Content   |
|------------------|------------|---|
| MI_WEBQRYTIMEOUT | Yes        | Webdriver interrupts a running query that has not completed in MI_WEBQRYTIMEOUT seconds.  |
| MI_WEBKEEPALIVE  | Yes        | Every MI_WEBKEEPALIVE seconds, Webdriver checks the Web browser connection. If the browser is no longer connected because a STOP/CANCEL has been sent by the browser, the running query is interrupted and the Web server daemon is freed to execute the next query request (NSAPI only). |

The following is an example **web.cnf** file with Webdriver connection settings:

```
# web.cnf file
INFORMIXDIR      /local1/web/sqlldist
INFORMIXSERVER   IUS9_web
MI_USER          webdba
MI_PASSWORD      secret
MI_DATABASE      webdb
Mival?           apb
Minam            ID
Micol            object
Mitab            webPages
WEB_HOME         /nsapi/secure/
# Connection settings
MI_WEBKEEPALIVE  5
MI_WEBQRYTIMEOUT 30
```



---

# Caching Large Objects Using Webdriver

|  |     |
|--|-----|
| Configuring Webdriver Large Object Caching . . . . .       | 9-4 |
| Configuring Webdriver Large Object Cache Cleanup . . . . . | 9-6 |



# W

ebdriver enables caching of large objects to improve performance. Caching reduces the number of database requests required to retrieve large objects.

If your application contains many static large objects, you can improve performance by eliminating some database requests and retrieving large objects directly from the disk cache for Webdriver. When you enable caching, Webdriver creates a disk copy of the large object the first time it is retrieved from the database. Subsequent requests for that large object retrieve the large object from the disk cache. Since large objects cannot be updated (a large object must be deleted and reinserted, and therefore has a new large object handle), it is not possible to retrieve stale objects.

## Configuring Webdriver Large Object Caching

To enable Webdriver large object caching, set the following variables in your **web.cnf** file.

| Variable         | Mandatory? | Content  |
|------------------|------------|--|
| MI_WEBCACHEDIR   | Yes        | Directory on the Web server machine in which cached large objects are placed. If not set, large objects are not cached.                      |
| MI_WEBCACHESUB   | No         | Number of subdirectories per database created under the directory specified by MI_WEBCACHEDIR. Default is one subdirectory per database.     |
| MI_WEBCACHELIFE  | No         | Length of time that a large object remains in cache. Specify MI_WEBCACHELIFE in units of seconds (s or S), hours (h or H), or days (d or D). |
| MI_WEBCACHECRON  | No         | Length of time between execution of cache cleanup. Specify MI_WEBCACHECRON in units of seconds (s or S), hours (h or H), or days (d or D).   |
| MI_WEBCACHEMAXLO | No         | Maximum size in bytes of large objects to be cached. Default is 64 K.  |

When you set the MI\_WEBCACHEDIR variable, Webdriver places a large object in its disk cache the first time the large object is retrieved. Subsequent retrievals of that large object are made from Webdriver's disk cache.

Depending on the setting for the MI\_WEBCACHESUB variable, one or more subdirectories of MI\_WEBCACHEDIR are created automatically for each database for which large objects are cached. The default number of subdirectories for cached large objects is one for each database.





If you expect many large objects to be cached, you can create more than one subdirectory per database under the MI\_WEBACHEDIR directory by specifying a higher number for MI\_WEBACHESUB. The subdirectories created are named **database\_name0**, **database\_name1**, and so on. There is no limit to the number of large objects that can be placed in these subdirectories, other than any operating system limitations.

**Important:** *If you modify the setting for MI\_WEBACHESUB, the algorithm used to locate large objects in the database subdirectories changes. Therefore you should remove all large objects from the subdirectories if you change the value for MI\_WEBACHESUB. The algorithm for creating the database subdirectories is such that the subdirectories might not be created in sequential numeric order.*

The following is an example **web.cnf** file with Webdriver large object caching enabled:

```
## web.cnf
INFORMIXDIR      /local1/web/sqlldist
INFORMIXSERVER   IUS9_web
MI_USER          webdba
MI_PASSWORD      secret
MI_DATABASE      webdb
Mival?          apb
Minam            ID
Micol            object
Mitab            webPages
WEB_HOME         /cgi-bin/webdriver

# Set cache directory
MI_WEBACHEDIR    /bigdisk/L0cache
# Create 30 subdirectories. Estimate as many as 6000 large
# objects with approximately 200 objects in each subdirectory.
MI_WEBACHESUB    30
# Set up cache life and clock daemon.
# Objects remain in cache unless they
# have not been accessed for 2 days.
# Clock daemon runs every 2 hours.
MI_WEBACHELIFE   2d
MI_WEBACHECRON   2h
# Increase maximum size of large objects
# that can be stored in cache to 1 M.
MI_WEBACHEMAXLO  1024000
```

Webdriver names a cached large object using a compressed version of the large object handle. For example, an image stored in Webdriver's disk cache is named as follows:

```
/bigdisk/L0cache/webdb0/na6b7c8d9m2m2114k2g9qlp686f626a65637  
44g1814m1ft48h9bde74ga2b9dia2begeg
```

---

## Configuring Webdriver Large Object Cache Cleanup

Set the MI\_WEBCACHELIFE variable to the length of time you want large objects to remain in the disk cache after their last access. Specify MI\_WEBCACHELIFE in units of seconds (s or S), hours (h or H), or days (d or D). After this time, when a cache cleanup is performed, the large objects are deleted. When you set the MI\_WEBCACHECRON variable, a cache cleanup is performed every MI\_WEBCACHECRON unit of time. Specify MI\_WEBCACHECRON in units of seconds (s or S), hours (h or H), or days (d or D).

To force a cache cleanup, enter the following command from the operating system level, in the directory that contains the Webdriver executable file and the appropriate **web.cnf** file:

```
webdriver cache_purge
```

---

# Using NSAPI Webdriver Features

|  |       |
|--|-------|
| NSAPI Webdriver Architecture . . . . .                           | 10-3  |
| Configuring the Netscape Web Server . . . . .                    | 10-3  |
| Configuring Startup Configuration Information . . . . .          | 10-4  |
| Configuring Object Management . . . . .                          | 10-4  |
| Starting and Stopping the NSAPI Webdriver Processes . . . . .    | 10-6  |
| Invoking the NSAPI Webdriver . . . . .                           | 10-6  |
| Implementing Security with the NSAPI Webdriver . . . . .         | 10-7  |
| Passing Image Map Coordinates with the NSAPI Webdriver . . . . . | 10-10 |
| Error Logging with the NSAPI Webdriver . . . . .                 | 10-10 |





**T**he NSAPI Webdriver uses the Netscape API interface rather than the CGI interface to connect to the database and execute AppPages. The NSAPI Webdriver offers the ability to:

- use security features built into the Netscape Web server to control access to AppPages.
- eliminate CGI process overhead.

***Important:** The NSAPI Webdriver is implemented using the Netscape Server API. Check the on-line documentation for supported Web server versions.*

---

## NSAPI Webdriver Architecture

The NSAPI interface for the NSAPI Webdriver replaces the CGI interface used in the CGI implementation of Webdriver.

When the Netscape Web server is started, the NSAPI Webdriver shared object is loaded using information specified in the Web server configuration files, described in the following section. If user authorization is enabled, the database connection is also used for user validation and security.

---

## Configuring the Netscape Web Server

The NSAPI Webdriver obtains configuration information about the Web application environment from the **web.cnf** file and from the **obj.conf** Netscape Web server configuration file, located in the server root configuration directory. The **obj.conf** file contains startup configuration information and object management for the Netscape Enterprise and FastTrack servers.



**Important:** Back up the existing **obj.conf** file before you modify it. For information on Netscape configuration files, see the *Netscape Web server Installation and Reference Guide*. When you edit the **obj.conf** file for Netscape Enterprise and FastTrack servers, the Netscape Administration Server detects the changes, and generates a JavaScript Alert stating Warning: Manual edits not loaded. Some configuration files have been edited by hand. Use the 'Apply' button on the upper right side of the screen to load the latest configuration files. **Click the OK button, and then click the Apply button to load the changes. The Netscape Administration Server should now function normally.**

## Configuring Startup Configuration Information

The **obj.conf** file contains startup configuration information for the Netscape Enterprise and FastTrack servers. The following directives must be added to the beginning of the **obj.conf** file to register Webdriver as an NSAPI function:

```
Init fn="load-modules"  
shlib="path_to_INFORMIXDIR/extend/web.3.30.UC1/netscape/driversnsapi20.so" \  
      funcs="informix_auth,informix_explode,informix_init"  
Init fn="informix_init"
```

In the preceding Web server configuration file entry, the first Init directive registers the functions used by the NSAPI Webdriver and points to the shared object that contains these functions. Specify the full path to the directory where the shared object resides in the shlib parameter. The second Init directive indicates that the Web server initializes the NSAPI Webdriver with the **informix\_init** NSAPI function upon startup.

## Configuring Object Management

The **obj.conf** file also contains mapping information for the execution of NSAPI functions during the processing of requests by the Web server. Two additions must be made to this file for each Webdriver object:

- A NameTrans directive that specifies the path and the name for the object
- An Object setting for the object named in the NameTrans directive

In the **default** Object setting, where every line begins with NameTrans, add a directive to configure an NSAPI Webdriver directory for each Webdriver object. The directives are evaluated in sequential order; therefore, to ensure that the correct URL mapping is chosen, the NSAPI Webdriver NameTrans directives should precede other directives:

```
NameTrans fn="pfx2dir" from="URL-mapped_NSAPI_path" dir="NSAPI_directory"
name="webdriver1"
```

The preceding NameTrans directive indicates to the Web server that any URL including the path specified in the from parameter follows the directives for the object specified by the name parameter. The name maps to the Object setting for the named object, which you add to the end of the **obj.conf** file. The following shows an example setting for the **webdriver1** object:

```
<Object name="webdriver1">
Service method=(GET|POST) fn="informix_explode" filename="/path_to/web.cnf"
</Object>
```

The Service directive indicates the call to the **informix\_explode** NSAPI function, which makes the call to the **WebExplode** function. The value of the filename parameter is the path to the **web.cnf** file for this object.

To use Web server authorization, add AuthTrans and PathCheck directives to your Object setting. The **webdriver2** object has a NameTrans directive that is similar to the preceding example:

```
NameTrans fn="pfx2dir" from="URL-mapped_NSAPI_path" dir="NSAPI_directory"
name="webdriver2"
```

The object specification for the **webdriver2** object contains additional information about how user authorization is performed:

```
<Object name="webdriver2">
AuthTrans fn="basic-auth" auth-type="basic" userdb="/path_to/web.cnf"
userfn="informix_auth"
PathCheck fn="require-auth" auth-type="basic" realm="Secure"
Service method=(GET|POST) fn="informix_explode" filename="/path_to/web.cnf"
</Object>
```

The PathCheck directive indicates that authorization is required. The AuthTrans directive indicates that the **informix\_auth** NSAPI function performs the authorization based on the user access information from the **web.cnf** file specified in the userdb parameter. The **web.cnf** file should be the same as the one specified in the filename parameter of the Service directive. See the [“Implementing Security with the NSAPI Webdriver” on page 10-7](#) for information on using Netscape Web server authorization with the NSAPI Webdriver.

---

## Starting and Stopping the NSAPI Webdriver Processes

The NSAPI Webdriver processes are started when the first database request is made. The processes are restarted when the number of requests specified in the ProcessLife **magnus.conf** file parameter is reached.

---

## Invoking the NSAPI Webdriver

Because Webdriver is an NSAPI function, it is invoked by including the path in the from parameter of the NameTrans directive in a URL. This URL is different from the URL associated with the CGI Webdriver. The value for Mival follows the URL-mapped path to the NSAPI Webdriver directory. Both of the following configurations are allowed:

```
/NSAPI/test  
/NSAPI/test.html
```

In the preceding URLs, the from parameter of the NameTrans directive is **/NSAPI** and Mival is **test**. The **.html** extension is ignored. Any additional parameters can be added to the URL in the QUERY\_STRING (following the **?**):

```
/NSAPI/test.html?var1=1
```

or

```
/NSAPI/test?var1=1
```



The value of Mival in the QUERY\_STRING overrides the Mival value in the path. This provides compatibility with the CGI Webdriver. The following URLs are equivalent:

```
/NSAPI/ignore?Mival=apb2
/NSAPI/apb2
/NSAPI/apb2.html
```

In your AppPages, specify a URL as in the following example:

```
<?MIVAR>$WEB_HOME<?/MIVAR>Mival=apb2
```

In the NSAPI case, an example value for WEB\_HOME is /NSAPI/ignore. In the CGI case, an example value for WEB\_HOME is /cgi/webdriver. Therefore, you can migrate between the NSAPI and the CGI implementations of Webdriver simply by changing the value of WEB\_HOME. For more information about using WEB\_HOME to create dynamic links between AppPages, see [“Linking AppPages” on page 2-10](#).

---

## Implementing Security with the NSAPI Webdriver

To use the security features of the Netscape Web server, you must specify variables in the **web.cnf** file in addition to those required for the CGI Webdriver. For more information about the **web.cnf** file, see [“Configuring Webdriver” on page 2-3](#). Set the following **web.cnf** variables to configure Web server authorization.

| Variable     | Mandatory? | Content  |
|--------------|------------|--|
| Mluserable   | Yes        | Name of the table that contains user access information.   |
| Mlusername   | Yes        | Name of the VARCHAR column in the user access table that contains the name of the database user.     |
| Mluserpasswd | Yes        | Name of the VARCHAR column of the user access table that contains the password of the database user. |

(1 of 2)



| Variable       | Mandatory? | Content  |
|----------------|------------|--|
| MIuserlevel    | Yes        | Name of the INTEGER column of the user access table that contains the access level of the database user. |
| MIpagelevel    | Yes        | Name of the INTEGER column of the AppPage table (MITab) that contains the access level of the AppPage.   |
| MI_WEBREDIRECT | No         | URL to redirect users to if they do not have access to the AppPage they attempt to retrieve.             |

(2 of 2)

**Important:** If *MIpagelevel* is not set in the **web.cnf** file, no authorization check is performed.

The following is an example **web.cnf** file with user access settings based on the **webPages** and **webUsers** tables, described in the schema for AppPage Builder in [Appendix B](#):

```
# web.cnf file
INFORMIXDIR           /local1/web/sqlldist
INFORMIXSERVER        IUS9_web
MI_USER               webdba
MI_PASSWORD           secret
MI_DATABASE            webdb
Mival?                apb
Minam                 ID
Micol                 object
MITab                 webPages
WEB_HOME              /nsapi/secure/
# NSAPI user authorization variables
MIpagelevel           read_level
MIusertable            webUsers
MIusername             name
MIuserpasswd           password
MIuserlevel            security_level
MI_WEBREDIRECT         http://host/cgi-bin/errors
```

If the access level of the retrieved AppPage is less than or equal to the access level of the authenticated user, the user can see the AppPage. The access level of the user, obtained from the value of the MIUserlevel column of the MIUserTable table, is stored in the MI\_WEBACCESSLEVEL variable. Webdriver does not allow this variable to be overridden in the URL. Additionally, the Web server stores the name of the remote user in the REMOTE\_USER Web server environment variable. You can access the value of REMOTE\_USER within your AppPages. Webdriver does not allow this variable to be overridden in the URL.

If a database access is set up in the **obj.conf** file with no authorization (no AuthTrans directive), only those AppPages with an access level of 0 are accessible without authorization. AppPages with higher access levels still require authorization. When authorization is not available or access is denied, you can redirect the browser to another URL by setting the MI\_WEBREDIRECT variable to that URL. If MI\_WEBREDIRECT is not set and a user attempts to access an AppPage with an access level higher than 0, an access error is raised.

In the following example, both authorized and unauthorized users access the same AppPages, with two different views of the same application.

The following are the NameTrans and Object setting entries in the **obj.conf** file for secure users:

```
NameTrans from="/secure" fn="pfx2dir" dir="/secure" name="secure"

<Object name="secure">
AuthTrans userfn="informix_auth" userdb="/path_to_secure/web.cnf" fn="basic-
auth" auth-type="basic"
PathCheck realm="Secure" fn="require-auth" auth-type="basic"
Service method="(GET|POST)" filename="/path_to_secure/web.cnf"
fn="informix_explode"
</Object>
```

The following are the NameTrans and Object setting entries in the **obj.conf** file for guest users:

```
NameTrans from="/guest" fn="pfx2dir" dir="/guest" name="guest"

<Object name="guest">
Service method="(GET|POST)" filename="/path_to_guest/web.cnf"
fn="informix_explode"
</Object>
```

See “[Configuring Object Management](#)” on page 10-4 for a description of the preceding **obj.conf** file entries. Access to AppPages with a security level of 0 is available to both unauthorized and authorized users. However, any AppPages with an access level greater than 0 can only be accessed by authorized users with a access level (MIuserlevel) equal to or higher than the access level for the AppPage (MIpagelevel). Both unauthorized and authorized users can access AppPages from the same MItab table.

For the preceding **obj.conf** file entries, an example of a guest URL is:

```
http://myhost:port/guest/pagename.html
```

An example of a secure URL is:

```
http://myhost:port/secure/pagename.html
```

Customize the layout and information on the AppPages based on both the REMOTE\_USER and the MI\_WEBACCESSLEVEL variables to display different information to guest and secure users.

---

## Passing Image Map Coordinates with the NSAPI Webdriver

For details on passing image map coordinates with Webdriver in standard image maps and forms, see “[Passing Image Map Coordinates](#)” on page 8-11. An example image map URL is:

```
http://myhost:port/cgi-bin/webdriver/MImap=on&MIVAL=image_example?100,13
```

**Important:** You must specify MImap as the first variable in PATH\_INFO. Anything preceding MImap is ignored by the NSAPI Webdriver.



---

## Error Logging with the NSAPI Webdriver

The NSAPI Webdriver logs error messages to the Netscape Web server error log file. The location of this file is specified by the ErrorLog entry in the **magnus.conf** file.

---

# Debugging Web DataBlade Module Applications

The Web DataBlade module is one of many components of your Web-enabled applications. Other components include your Web browser, your Web server, and the INFORMIX-Universal Server database. You can use a variety of techniques to resolve problems with creation, configuration, or execution of Web applications.

To determine which component of your installation is failing, retrieve your AppPage directly by running Webdriver interactively, bypassing your Web browser and Web server.

To debug Web applications as you are developing them, use RAW mode to get more information about your Webdriver environment or use the **WebLint** function to find syntax errors within Web DataBlade module tags.

To obtain more information when an error occurs, enable Web DataBlade module tracing and check the appropriate log files.

---

## Running Webdriver Interactively

To debug configuration issues, you must first determine which component of your installation is failing. If you are unsure whether it is the Web server or Webdriver that is failing when you attempt to retrieve an AppPage, call Webdriver interactively, bypassing the Web browser and Web server.

### To run Webdriver interactively

1. Log in as the owner of your Web server (HTTPD).

2. Move to the directory in which the Webdriver executable file is located. For example:  

```
cd /disk6/netscape/ns-home/cgi
```
3. Set any variables in your environment that you would normally set in the calling URL. For example:  

```
setenv Mival testit
```
4. Invoke Webdriver:  

```
webdriver
```

The following is sample output:

```
Content-type: text/html
Content-length: 3703
<HTML>
.....
```

An appropriate error message is generated if the AppPage cannot be retrieved. For example, if you specify the name of an AppPage that does not exist,

```
setenv Mival wrong
```

the error is printed within the following Webdriver output:

```
webdriver

Content-type: text/html
Content-length: 225
<HTML> <HEAD><TITLE> Error Message</TITLE>
<HEAD> <BODY>
<H2>HTTP/1.0 404 Not Found</H2>
<B>Error from Informix:</B><HR>
  The resource you requested was not found.<P>
  Zero rows were returned from the server
</BODY> </HTML>
```

---

## Using RAW Mode with Webdriver

Webdriver allows you to enable RAW mode to aid in developing and debugging Web applications. When you enable RAW mode, you can:

- display the AppPage as stored in the database, without expanding the Web DataBlade module tags.
- display variables and identify where variable assignments are made.

Set the following **web.cnf** file variable to enable RAW mode.

| Variable          | Mandatory? | Content                      |
|-------------------|------------|------------------------------|
| MI_WEBRAWPASSWORD | Yes        | Password to enable RAW mode. |

The following is an example **web.cnf** file with MI\_WEBRAWPASSWORD set:

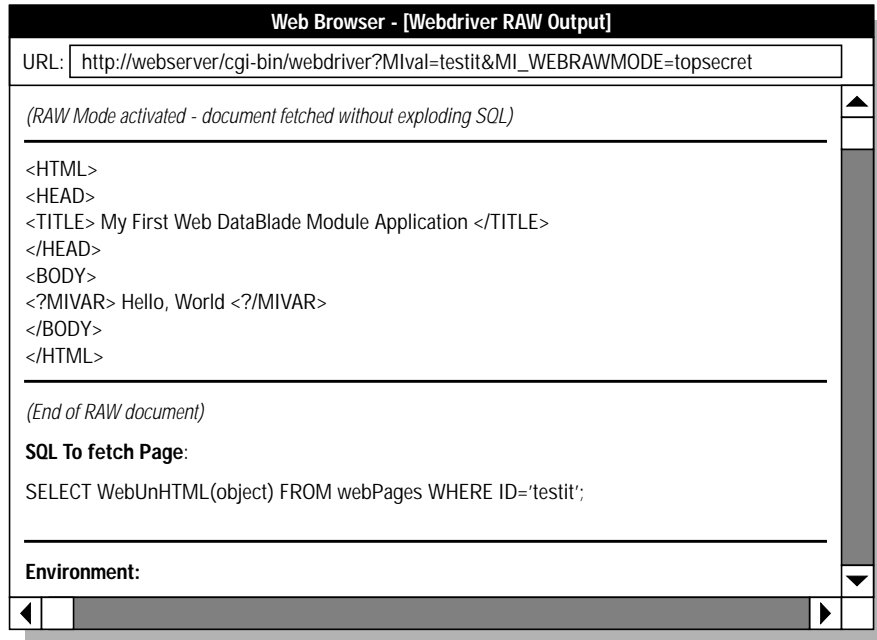
```
# web.cnf file
INFORMIXDIR      /local1/web/sqlldist
INFORMIXSERVER   IUS9_web
MI_USER          webdba
MI_PASSWORD      secret
MI_DATABASE      webdb
MIval?           apb
MInam            ID
MIcol            object
MITab            webPages
WEB_HOME         /cgi-bin/webdriver
MI_WEBRAWPASSWORD topsecret
```

You can retrieve the unexpanded AppPage by specifying `MI_WEBRAWMODE=value_of_MI_WEBRAWPASSWORD` in a URL. Webdriver returns the unexpanded AppPage as stored in the database, including the Web DataBlade module tags. RAW mode also displays all variables and where they were assigned. The following URL retrieves the **testit** AppPage in RAW mode:

```
http://myhost:port/cgi-bin/webdriver?MIval=testit&MI_WEBRAWMODE=topsecret
```

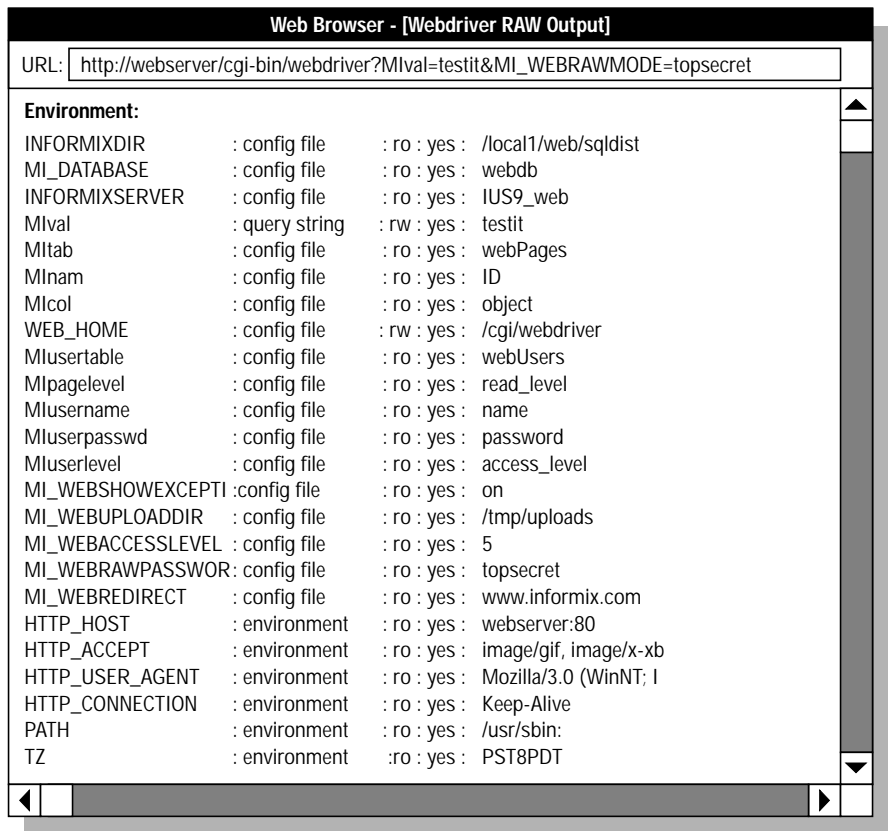
The following screens show sample RAW mode output.

**Figure A-1**  
Webdriver RAW Mode Output 1





**Figure A-2**  
Webdriver RAW Mode Output 2



The following is a sample of the environment variable information shown in the preceding figure:

```
MIval      : query string : rw : yes : testit
```

The first field in the environment variable list indicates the name of the variable, in this case, `Mlval`. The second value identifies where the variable has been set, with the following meanings.

| Environment Setting | Where Set?                           |
|---------------------|--------------------------------------|
| config file         | <b>web.cnf</b> file                  |
| query string        | QUERY_STRING environment variable    |
| path info           | PATH_INFO environment variable       |
| cookie              | Web browser cookie                   |
| post                | METHOD=POST in the calling HTML form |
| uri                 | NSAPI URL (NSAPI only)               |
| environment         | Web server environment (CGI only)    |
| rq->vars            | Web server environment (NSAPI only)  |
| rq->headers         | Web server environment (NSAPI only)  |
| rq->reqpb           | Web server environment (NSAPI only)  |
| sn->client          | Web server environment (NSAPI only)  |

The third field indicates the mode of the variable. The mode is set to either `rw` or `ro`, where `rw` indicates that the value can be overridden, and `ro` indicates that the value cannot be overridden.

The fourth field can be set to `yes` or `no`, and indicates whether or not the variable is passed to the Webdriver call to **WebExplode**.

The final field is the current value of the variable.

## Using WebLint

Use the **WebLint** function to find syntax errors within Web DataBlade module tags. **WebLint** is described in [“WebLint” on page 7-7](#).

## Enabling WebExplode Tracing

Set the following variables to enable logging of **WebExplode** function trace information.

| Variable       | Mandatory? | Content   |
|----------------|------------|---|
| MI_WEBEXPLEVEL | Yes        | Enables <b>WebExplode</b> function tracing.           |
| MI_WEBEXPLOG   | No         | File to which <b>WebExplode</b> messages are written. |

The following table lists the MI\_WEBEXPLEVEL trace settings.

| Trace Value | Information Displayed                        |
|-------------|--|
| 1           | Logs information within HTML comments.       |
| 2           | Logs conditional information.                |
| 4           | Logs tag attribute information.              |
| 8           | Logs variable information.                   |
| 16          | Logs the processing results of dynamic tags. |
| 32          | Logs the SQL statement being executed.       |

When you enable tracing, the information is written to the trace file specified by the MI\_WEBEXPLOG variable. If you do not set MI\_WEBEXPLOG (or if the server cannot write to the specified file), the server creates a file in the **/tmp** directory with a **.trc** file extension. You can also write your own message to the trace file using the TRACMSG variable processing function. For example, you can log errors to the trace file within your AppPages as follows:

```
<?MIVAR>$(TRACMSG, You encountered the error: $MI_ERRORMSG)<?/MIVAR>
```



The trace value is additive; therefore, you can turn on multiple settings simultaneously. For example, if you set `MI_WEBEXPLEVEL` to 6, trace information is generated for both tags and conditionals. Add 1 to the value of `MI_WEBEXPLEVEL` to generate output within your AppPages as HTML comments. The output is still written to the trace file.

**Important:** If you add 1 to the value of `MI_WEBEXPLEVEL` to generate output within your AppPages as HTML comments, the additional text within your HTML might change how the Web browser renders your AppPage.

## Enabling Webdriver Tracing

Set the following variables to enable logging of Webdriver trace information.

| Variable                    | Mandatory? | Content   |
|-----------------------------|------------|---|
| <code>MI_WEBDRVLEVEL</code> | Yes        | Enables Webdriver tracing.                                    |
| <code>MI_WEBDRVLOG</code>   | No         | Name of the log file to which Webdriver messages are written. |

The following table lists the `MI_WEBDRVLEVEL` trace settings.

| Trace Value | Information Displayed   |
|-------------|---|
| 1           | Logs all pblocks (NSAPI only). Pblocks contain the name/value pairs passed from the Web browser to the Netscape Web server. |
| 2           | Logs callbacks including errors and other messages, such as HTTP headers, from INFORMIX-Universal Server.                   |
| 4           | Logs Webdriver query requests to INFORMIX-Universal Server, such as calls to <b>WebExplode</b> or authorization requests.   |
| 8           | Logs large object requests.   |
| 16          | Logs AppPage headers.   |

(1 of 2)

| Trace Value | Information Displayed  |
|-------------|--|
| 32          | Logs large object headers.   |
| 64          | Logs client file upload information.   |
| 128         | RESERVED   |
| 256         | Logs session variables.  |
| 512         | Logs information similar to the information logged by the NSAPI driver (CGI only). |
| 1024        | Logs connection pool information.  |

(2 of 2)

The trace value is additive; therefore, you can turn on multiple settings simultaneously. When you enable tracing, the information is written to the trace file specified by the MI\_WEBDRVLOG variable. If the file does not exist, it is created. If the file exists, additional messages are appended to it.

## Checking Log Files

Be sure to check your Web server error log files for any additional information when you encounter an error.



# AppPage Builder Schema

AppPage Builder (APB) provides a flexible and extensible base for developing Web applications with the Web DataBlade module. APB has built-in support for common multimedia objects, such as images, audio clips, video clips, and documents. In addition, you can extend the APB schema to add support for additional object types. This appendix describes the APB schema and lists the schema definition for APB.

---

## Schema Description

The following sections describe key components of the APB schema.

### Projects

A Web site consists of many different objects, such as AppPages, images, sound bites, and video clips. These objects are associated with a particular Web application, called a *project*. The information specific to a given project is stored in the **webProjects** table:

```
create table webProjects
(
    project                varchar(40) NOT NULL,
    description            varchar(250),
    owner                  varchar(40),
    primary key            (project)
);
```

Each object is assigned to one of the projects in the **webProjects** table.

### Object Types

In addition to being associated with a project, the objects stored in the database are assigned an *object type*. An object type, such as Image, Audio, or Video, defines the contents of the object. The **webObjectTypes** table stores valid object types:

```
create table webObjectTypes
(
    object_type          varchar(40) NOT NULL,
    page_suffix          varchar(40) NOT NULL,
    primary key          (object_type)
);
```

The information in the **page\_suffix** column indicates which APB AppPages are used to manage that object type. For example, the Image object type has a page suffix of Image, and objects of that type are maintained by the **apb\_add\_Image**, **apb\_edit\_Image**, and **apb\_delete\_Image** AppPages.

### MIME Types

There can be a number of different object formats within each object type. For example, there are MPEG, Quicktime, and AVI video formats. The **webMimeType** table provides a mapping from each object type to its valid MIME types:

```
create table webMimeType
(
    mime_name            varchar(40) NOT NULL,
    mime_type            varchar(40) NOT NULL,
    extension            varchar(14) NOT NULL,
    object_type          varchar(40) NOT NULL,
    references webObjectTypes,
    primary key          (object_type, mime_type)
);
```

### Adding Object Types

You can extend the APB schema by adding new multimedia object types.



**To add new multimedia object types**

1. Create the appropriate entries in the **webObjectTypes** and **webMimeType** tables. To create a new object type, enter the name and the page suffix to identify the AppPages that manage objects of that type. To add support for a new MIME type, select the object type with which it is associated and specify the name and MIME type. The **Add Object Type** and **Add MIME Type** options are available from the **APB Admin Menu**.
2. Define a new table to store information for the new object type.
3. Write three AppPages to enable users to add, edit, and delete the new object type. These AppPages can be based on similar existing AppPages, such as **apb\_add\_Audio**, **apb\_edit\_Audio**, and **apb\_delete\_Audio**.

## Schema Definition

The following is the schema definition for APB:

```
CREATE TABLE webProjects
(
    project          varchar(40) NOT NULL,
    description      varchar(250),
    owner            varchar(40),
    primary key      (project)
);

CREATE TABLE webObjectTypes
(
    object_type      varchar(40) NOT NULL,
    page_suffix      varchar(40) NOT NULL,
    primary key      (object_type)
);

CREATE TABLE webMimeTypes
(
    mime_name        varchar(40) NOT NULL,
    mime_type        varchar(40) NOT NULL,
    extension        varchar(14) NOT NULL,
    object_type      varchar(40) NOT NULL,
    references webObjectTypes,
    primary key      (object_type, mime_type)
);

CREATE TABLE webPages
(
    ID               varchar(40) NOT NULL,
    project          varchar(40) NOT NULL,
    references webProjects,
    object_type      varchar(40) NOT NULL,
    mime_type        varchar(40) NOT NULL,
    description      varchar(250),
    author           varchar(40),
    page_version     integer,
    last_changed     datetime year to second,
    last_changed_by  varchar(40),
    write_level      integer,
    read_level       integer,
    object           HTML,
    foreign key      (object_type, mime_type)
    references webMimeTypes,
    primary key      (ID)
);
```

```

CREATE TABLE webPagesVersions
(
    ID                varchar(40) NOT NULL,
    project            varchar(40) NOT NULL
    references webProjects,
    object_type        varchar(40) NOT NULL,
    mime_type          varchar(40) NOT NULL,
    description        varchar(250),
    author             varchar(40),
    page_version       integer,
    last_changed       datetime year to second,
    last_changed_by    varchar(40),
    write_level        integer,
    read_level         integer,
    object             HTML,
    foreign key        (object_type,mime_type)
    references webMimeTypes,
    primary key        (ID,page_version)
);

CREATE TABLE webImages
(
    ID                varchar(40) NOT NULL,
    project            varchar(40) NOT NULL
    references webProjects,
    object_type        varchar(40) NOT NULL,
    mime_type          varchar(40) NOT NULL,
    description        varchar(250),
    height             integer,
    width              integer,
    object             BLOB,
    foreign key        (object_type,mime_type)
    references webMimeTypes,
    primary key        (ID)
) PUT object IN ($2);

CREATE TABLE webAudios
(
    ID                varchar(40) NOT NULL,
    project            varchar(40) NOT NULL
    references webProjects,
    object_type        varchar(40) NOT NULL,
    mime_type          varchar(40) NOT NULL,
    description        varchar(250),
    length             decimal(10,2),
    object             BLOB,
    foreign key        (object_type,mime_type)
    references webMimeTypes,
    primary key        (ID)
) PUT object IN ($2);

```

```
CREATE TABLE webVideos
(
    ID                varchar(40) NOT NULL,
    project            varchar(40) NOT NULL
    references webProjects,
    object_type        varchar(40) NOT NULL,
    mime_type          varchar(40) NOT NULL,
    description         varchar(250),
    height             integer,
    width              integer,
    length             decimal(10,2),
    object             BLOB,
    foreign key        (object_type,mime_type)
    references webMimeTypes,
    primary key        (ID)
) PUT object IN ($2);

CREATE TABLE webDocuments
(
    ID                varchar(40) NOT NULL,
    project            varchar(40) NOT NULL
    references webProjects,
    object_type        varchar(40) NOT NULL,
    mime_type          varchar(40) NOT NULL,
    description         varchar(250),
    object             BLOB,
    foreign key        (object_type,mime_type)
    references webMimeTypes,
    primary key        (ID)
) PUT object IN ($2);

CREATE TABLE webUsers
(
    name              varchar(40) NOT NULL,
    password           varchar(40) NOT NULL,
    security_level     integer    NOT NULL,
    default_project    varchar(40) NOT NULL
    references webProjects,
    def_object_type    varchar(40) NOT NULL
    references webObjectTypes,
    textarea_width     integer    default 80,
    textarea_height    integer    default 20,
    page_versions      char       default 't',
    web_lint            integer    default 2,
    primary key        (name)
)
```

# Web DataBlade Module Variables

This appendix lists all the variables that can be set for Webdriver and the **WebExplode** function. Variables listed as *mandatory* must be set for a particular feature to be enabled.

Web DataBlade module variables begin with MI. Variables that represent elements of your database schema (such as MI`tab`, MI`col`, and so on) or are set in a URL (such as MI`map`) are of the form MI`varname`. All other variables are of the form MI\_`varname`.

Set the following variables to retrieve an AppPage using Webdriver.

| Variable    | Mandatory? | Content  |
|-------------|------------|--|
| MItab       | Yes        | Name of the table in which the AppPages for the Web application are stored.                                    |
| MIcol       | Yes        | Name of the HTML column that contains the AppPage in the Web application table.                                |
| MInam       | Yes        | Name of the VARCHAR column that identifies the appropriate row of the Web application table.                   |
| MIval       | Yes        | Value to check against the MInam column to identify the row you want to retrieve in the Web application table. |
| WEB_HOME    | No         | URL-mapped path to Webdriver.  |
| MI_USER     | Yes        | Name of the database user.   |
| MI_PASSWORD | Yes        | Password for the user specified by MI_USER.  |
| MI_DATABASE | Yes        | Name of the database to connect to when the user makes a Webdriver request.                                    |

For more information on retrieving AppPages using Webdriver, see [“Configuring Webdriver” on page 2-3](#).

Set the following variables to enable AppPage-level authorization using Webdriver.

| Variable          | Mandatory? | Content  |
|-------------------|------------|--|
| MIpagelevel       | Yes        | Name of the INTEGER column of the AppPage table (MITab) that contains the access level of the AppPage. |
| MI_WEBACCESSLEVEL | Yes        | Access level of all users for this <b>web.cnf</b> file.  |
| MI_WEBREDIRECT    | No         | URL to redirect users to if they do not have access to the AppPage they attempt to retrieve.           |

For more information on implementing AppPage-level authorization, see [“Implementing AppPage-Level Security” on page 2-7](#).

Set the following variables to retrieve a large object by uniquely identifying the object within a table.

| Variable  | Mandatory? | Content  |
|-----------|------------|--|
| MITabObj  | Yes        | Name of a table in which Web application large objects are stored.   |
| MIcolObj  | Yes        | Name of the BLOB column that contains large objects in the MITabObj table.                                 |
| MInamObj  | Yes        | Name of the VARCHAR column that identifies the appropriate row of the MITabObj table.                      |
| MIvalObj  | Yes        | Value to check against the MInamObj column to identify the row you want to retrieve in the MITabObj table. |
| MItypeObj | Yes        | MIME type and subtype used to export the large object.   |

For more information on retrieving large objects by name, see [“Retrieving Large Objects by Name” on page 2-13.](#)

Set the following variables to retrieve large objects by obtaining large object handles (LO handles) dynamically from a SELECT statement.

| Variable  | Mandatory? | Content  |
|-----------|------------|--|
| LO        | Yes        | Large object handle.                                   |
| MltypeObj | Yes        | MIME type and subtype used to export the large object. |

For more information on retrieving large objects by LO handle, see [“Retrieving Large Objects by LO Handle” on page 2-15.](#)

Set the following variable to display database exception messages using Webdriver.

| Variable             | Mandatory? | Content  |
|----------------------|------------|--|
| MI_WEBSHOWEXCEPTIONS | No         | Set to <b>on</b> or <b>off</b> . When <b>on</b> , Webdriver displays the database exception returned by <b>WebExplore</b> . When <b>off</b> , Webdriver displays an HTTP/1.0 500 Server error. Default is <b>off</b> . |

For more information on displaying database exception messages, see [“Processing Errors with Webdriver” on page 4-25.](#)

Set the following variable to upload client files using Webdriver.

| Variable        | Mandatory? | Content  |
|-----------------|------------|--|
| MI_WEBUPLOADDIR | No         | Directory on the Web server machine in which uploaded files are placed. Default is /tmp. |

For more information, see [“Uploading Client Files” on page 8-7.](#)





Set the following variable to pass image map coordinates using Webdriver.

| Variable | Mandatory? | Content   |
|----------|------------|---|
| MImap    | Yes        | Set to <code>on</code> or <code>off</code> . When <code>on</code> , the URL is treated as an image map and the values are passed as <code>x</code> - and <code>y</code> -coordinates. Default is <code>off</code> . |

**Important:** *MImap must be set in the URL that invokes the AppPage. MImap must not be set in the `web.cnf` file.*

For more information, see [“Passing Image Map Coordinates” on page 8-11](#).

Set the following variables to configure Webdriver connections to INFORMIX-Universal Server.

| Variable         | Mandatory? | Content   |
|------------------|------------|---|
| MI_WEBQRYTIMEOUT | No         | Webdriver interrupts a running query that has not completed in MI_WEBQRYTIMEOUT seconds.  |
| MI_WEBKEEPALIVE  | No         | Every MI_WEBKEEPALIVE seconds, Webdriver checks the Web browser connection. If the browser is no longer connected because a STOP/CANCEL has been sent by the browser, the running query is interrupted and the Web server daemon is freed to execute the next query request (NSAPI only). |

For more information, see [“Configuring Webdriver Connection Settings” on page 8-15](#).

Set the following variables to enable caching of large objects using Webdriver.

| Variable         | Mandatory? | Content  |
|------------------|------------|--|
| MI_WEBCACHEDIR   | Yes        | Directory on the Web server machine in which cached large objects are placed. If not set, large objects are not cached.                      |
| MI_WEBCACHESUB   | No         | Number of subdirectories per database created under the directory specified by MI_WEBCACHEDIR. Default is one subdirectory per database.     |
| MI_WEBCACHELIFE  | No         | Length of time that a large object remains in cache. Specify MI_WEBCACHELIFE in units of seconds (s or S), hours (h or H), or days (d or D). |
| MI_WEBCACHECRON  | No         | Length of time between execution of cache cleanup. Specify MI_WEBCACHECRON in units of seconds (s or S), hours (h or H), or days (d or D).   |
| MI_WEBCACHEMAXLO | No         | Maximum size of objects to be cached. Default is 64 K.   |

For more information, see [“Caching Large Objects Using Webdriver” on page 9-3.](#)

Set the following variable to enable RAW mode using Webdriver.

| Variable          | Mandatory? | Content                      |
|-------------------|------------|------------------------------|
| MI_WEBRAWPASSWORD | Yes        | Password to enable RAW mode. |

For more information, see [“Using RAW Mode with Webdriver” on page 0-2.](#)

Set the following variables to enable logging of **WebExplode** function trace information.

| Variable       | Mandatory? | Content   |
|----------------|------------|---|
| MI_WEBEXPLEVEL | Yes        | Enables <b>WebExplode</b> function tracing.           |
| MI_WEBEXPLOG   | No         | File to which <b>WebExplode</b> messages are written. |

For more information, see [“Enabling WebExplode Tracing” on page 0-7](#).

Set the following variables to enable logging of Webdriver trace information.

| Variable       | Mandatory? | Content   |
|----------------|------------|---|
| MI_WEBDRVLEVEL | Yes        | Enables Webdriver tracing.                                    |
| MI_WEBDRVLOG   | No         | Name of the log file to which Webdriver messages are written. |

For more information, see [“Enabling Webdriver Tracing” on page 0-8](#).



# Index

## A

ANCHOR tag 2-10  
 AND variable processing  
   function 5-4  
 AppPage Builder 3-3, B-1  
 AppPages  
   creating with AppPage  
     Builder 3-6  
   definition of 1-3  
 Arithmetic variable processing  
   functions 5-3, 5-8

## C

CHECKBOXLIST system dynamic  
   tag 6-4  
 Client file upload 3-3, 8-7  
 Column variables 4-7  
 Commercial at (@) in dynamic  
   tags 6-13, 6-15  
 COND attribute  
   of dynamic tag 6-3  
   of MIBLOCK tag 4-17  
 Conditional output using variable  
   expressions 5-12  
 Conditional statements 4-17  
 Cookies 8-4

## D

Debugging Webdriver A-1, A-2,  
   A-8  
 DEFAULT attribute of MIVAR  
   tag 4-15

Double quotes  
   in variable expressions 5-16  
   in Web DataBlade module  
     tags 4-28  
 Dynamic tags 6-3

## E

EC variable processing function 5-4  
 ENCTYPE attribute of FORM  
   tag 3-3, 8-7  
 Entity reference  
   for " 4-28  
   for @ 6-15  
 EQ variable processing  
   function 5-4  
 ERR attribute of MIERORR  
   tag 4-21  
 Error handling  
   MI\_ERRORCODE variable 4-11  
   MI\_ERRORMSG variable 4-11  
   MI\_ERRORSTATE variable 4-11  
   using ERR attribute 4-21  
   using generic error handler 4-22  
   using Webdriver 4-25

## F

File upload 3-3, 8-7  
 FIX variable processing  
   function 5-4  
 FORM tag  
   linking AppPages 2-10  
   uploading files in an HTML  
     form 3-3, 8-7  
   using image maps 8-12

Formatting characters in Web  
 DataBlade module tags 4-28

Functions

- arithmetic 5-3, 5-8
- server 7-3
- string 5-3
- variable processing 5-3

## H

HTTP headers 8-3

HTTPHEADER variable processing  
 function 5-4, 8-3

## I

IF variable processing function 5-5

Image maps 8-10, 10-10

INDEX variable processing  
 function 5-5

ISINT variable processing  
 function 5-5

ISNUM variable processing  
 function 5-5

## L

Large objects

- caching with Webdriver 9-3
- retrieving with  
 Webdriver 2-13 to 2-17
- uploading with Webdriver 8-8

LO variable 2-15

LOWER variable processing  
 function 5-5

## M

Mandatory environment  
 variables 2-3

MAXROWS attribute of MISQL  
 tag 4-14

MIBLOCK tag 4-17 to 4-19

Mlcol variable 2-4

MlcolObj variable 2-13

MIERROR tag 4-19 to 4-27

Mlmap variable 8-10

MIME types 3-8, 8-3, B-3

MInam variable 2-4

MInamObj variable 2-13

Mlpagelevel variable 2-8, 10-8

MISQL tag 4-5 to 4-14

- formatting the SQL  
 results 4-7 to 4-14

Mltab variable 2-4

MltabObj variable 2-13

MltypeObj variable 2-13, 2-15

Mluserlevel variable 10-8

Mlusername variable 10-7

Mluserpasswd variable 10-7

Mlinsertable variable 10-7

Mlval variable 2-4

MlvalObj variable 2-13

MIVAR tag 4-14 to 4-16

MI\_COLUMNCOUNT  
 variable 4-11

MI\_CURRENTROW variable 4-11

MI\_DATABASE variable 2-4

MI\_ERRORCODE variable 4-11

MI\_ERRORMSG variable 4-11

MI\_ERRORSTATE variable 4-11

MI\_NOVALUE variable 4-13

MI\_NULL variable 4-13

MI\_PASSWORD variable 2-4

MI\_ROWCOUNT variable 4-11

MI\_SQL variable 4-11

MI\_USER variable 2-4

MI\_WEBACCESSLEVEL  
 variable 2-8, 10-9

MI\_WEBCACHECRON  
 variable 9-4

MI\_WEBCACHEDIR variable 9-4

MI\_WEBCACHELIFE variable 9-4

MI\_WEBCACHEMAXLO  
 variable 9-4

MI\_WEBCACHESUB variable 9-4

MI\_WEBDRVLEVEL variable A-8

MI\_WEBDRVLOG variable A-8

MI\_WEBEXPLEVEL variable A-7

MI\_WEBEXPLOG variable A-7

MI\_WEBKEEPALIVE variable 8-14

MI\_WEBQRYTIMEOUT  
 variable 8-14

MI\_WEBBRAWMODE variable A-3

MI\_WEBBRAWPASSWORD  
 variable A-3

MI\_WEBREDIRECT variable 2-8,  
 10-8

MI\_WEBSHOWEXCEPTIONS  
 variable 4-25

MI\_WEBUPLOADDIR variable 8-7

## N

NAME attribute

- of MISQL tag 4-6
- of MIVAR tag 4-15

NC variable processing  
 function 5-5

NE variable processing  
 function 5-5

NOT variable processing  
 function 5-5

NSAPI Webdriver 10-3

- configuring 10-3
- implementing security 10-7
- invoking 10-6
- passing image map  
 coordinates 10-10
- starting 10-6

NTH variable processing  
 function 5-6

NXST variable processing  
 function 5-6

## O

Object types 3-8, B-2

OR variable processing  
 function 5-6

## P

PATH\_INFO Web server  
 environment variable 2-6, 8-11

POSITION variable processing  
 function 5-6

Processing variables 4-10

Projects

- adding 3-8, B-1
- editing 3-8

## Q

QUERY\_STRING Web server  
environment variable 2-6, 8-11

Quotes  
in variable expressions 5-16  
in Web DataBlade module  
tags 4-28

## R

RADIOLIST system dynamic  
tag 6-6

RAW mode A-2

REMOTE\_USER Web server  
environment variable 2-9, 10-9

REPLACE variable processing  
function 5-6, 5-9

Row variables 4-7

## S

Scope of variables 4-4, 7-4

Security using NSAPI  
Webdriver 10-7

SELECTLIST system dynamic  
tag 6-9

SEPARATE variable processing  
function 5-6, 5-9

Server functions 7-3

SETVAR variable processing  
function 5-6

SGML tags 1-4, 4-3, 6-3

Special characters  
in dynamic tags 6-15  
in variable expressions 5-16  
in Web DataBlade module  
tags 4-28

SQL attribute of MYSQL tag 4-6

STRFILL variable processing  
function 5-6

String variable processing  
functions 5-3

STRLEN variable processing  
function 5-6

SUBSTR variable processing  
function 5-6

System dynamic tags  
CHECKBOXLIST 6-4  
RADIOLIST 6-6  
SELECTLIST 6-9

System variables 4-7 to 4-14

## T

TAG attribute of MIERROR  
tag 4-20

Tags  
CHECKBOXLIST 6-4  
dynamic 6-3  
MIBLOCK 4-17 to 4-19  
MIERROR 4-19 to 4-27  
MYSQL 4-5 to 4-14  
MIVAR 4-14 to 4-16  
RADIOLIST 6-6  
SELECTLIST 6-9  
SGML 1-4, 4-3, 6-3  
system dynamic 6-4  
tracing A-7  
user dynamic 6-13  
Web DataBlade module 1-4, 4-3

TRACMSG variable processing  
function 4-22, 5-7, A-7

Tracing Web DataBlade module  
tags A-7

Tracing Webdriver A-8

TRIM variable processing  
function 5-7

Troubleshooting Webdriver A-1,  
A-2, A-8

## U

UNHTML variable processing  
function 5-7  
*See also* WebUnHTML function.

UNSETVAR variable processing  
function 5-7

UPPER variable processing  
function 5-7

URLDECODE variable processing  
function 5-7  
*See also* WebURLDecode function.

URLENCODE variable processing  
function 5-7  
*See also* WebURLEncode function.

User dynamic tags 6-13

## V

Variable expressions 4-17, 5-3, 5-8

Variable processing functions  
AND 5-4  
arithmetic 5-3, 5-8  
conditional output 5-12  
EC 5-4  
EQ 5-4  
FIX 5-4  
HTTPHEADER 5-4, 8-3  
IF 5-5  
INDEX 5-5  
ISINT 5-5  
ISNUM 5-5  
LOWER 5-5  
NC 5-5  
NE 5-5  
NOT 5-5  
NTH 5-6  
NXST 5-6  
OR 5-6  
POSITION 5-6  
REPLACE 5-6, 5-9  
SEPARATE 5-6, 5-9  
SETVAR 5-6  
STRFILL 5-6  
string 5-3  
STRLEN 5-6  
SUBSTR 5-6  
TRACMSG 4-22, 5-7, A-7  
TRIM 5-7  
UNHTML 5-7  
UNSETVAR 5-7  
UPPER 5-7  
URLDECODE 5-7  
URLENCODE 5-7  
XOR 5-7  
XST 5-7

## Variables

- appendix containing C-1
- case sensitivity 4-4
- column 4-7
- conditional expression 4-17
- default values 2-3
- mandatory 2-3
- MI\_NOVALUE 4-13
- MI\_NULL 4-13
- naming 4-4
- processing 4-10
- row 4-7
- scope 4-4, 7-4
- system 4-7 to 4-14
- user-defined 2-3
- Web DataBlade module 4-4
- Web server environment 2-3
- where interpreted 4-4
- where set 4-4

## W

- Webdriver 1-3
  - adding HTTP headers 8-3
  - caching large objects 9-3
  - call to WebExplode function 2-6
  - configuring 2-3 to 2-6
  - debugging A-1, A-2, A-8
  - environment variables 2-6
  - error handling 4-25
  - interactively running A-1
  - invoking 2-7
  - passing image map
    - coordinates 8-10
  - retrieving large
    - objects 2-13 to 2-17
  - tracing A-8
  - troubleshooting A-1, A-2, A-8
  - uploading files in an HTML
    - form 8-7
  - using NSAPI features 10-3
  - using RAW mode A-2
- WebExplode function 1-4, 2-6, 4-22, 7-4
- WebLint function 3-8, 7-7
- webMimeTypes table B-2
- webObjectTypes table B-2
- webProjects table B-1

- WebRelease function 7-10
- webTags table 6-13
- WebUnHTML function 7-12
  - See also* UNHTML variable
  - processing function.
- WebURLDecode function 7-14
  - See also* URLENCODE variable
  - processing function.
- WebURLEncode function 7-15
  - See also* URLENCODE variable
  - processing function.
- web.cnf file 2-3
- WEB\_HOME variable 2-4, 2-10

## X

- XOR variable processing
  - function 5-7
- XST variable processing
  - function 5-7

## Symbols

- " character
  - in variable expressions 5-16
  - in Web DataBlade module
    - tags 4-28
- @ in dynamic tags 6-15