IBM Informix Web DataBlade Module

Administrator's Guide

Version 4.13
December 2001
Part No. 000-8674

Note:
Before using this information and the product it supports, read the information in the appendix entitled "Notices."

Table of Contents

Chapter 2 Getting Started

Chapter 3 Configuring Webdriver

Chapter 5 Using the Apache Webdriver

Chapter 6 Using the ISAPI Webdriver

Introduction

In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

The *IBM Informix Web DataBlade Module Administrator's Guide* describes how to administer Web applications that use the IBM Informix Web DataBlade module to dynamically retrieve data from an Informix database. The manual describes topics such as how to configure the Web DataBlade module for your database server; how to configure the NSAPI, Apache, CGI, and ISAPI Webdrivers; how to implement security in your Web applications; and how to increase the performance of your Web 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:

- Chapter 1, "Overview of the Web DataBlade Module," provides an overview of the architecture and features of the Web DataBlade module.
- Chapter 2, "Getting Started," provides information on how to initially set up the Web DataBlade module for your database.
- Chapter 3, "Configuring Webdriver," describes how to configure Webdriver, the format of the Webdriver configuration file (**web.cnf**), and how to use the Web DataBlade Module Administration Tool to create and update Webdriver mappings and Webdriver configurations.
- Chapter 4, "Using the NSAPI Webdriver," describes how to configure and use the NSAPI Webdriver.
- Chapter 5, "Using the Apache Webdriver," describes how to configure and use the Apache Webdriver.
- Chapter 6, "Using the ISAPI Webdriver," describes how to configure and use the ISAPI Webdriver.
- Chapter 7, "Using the CGI Webdriver," describes how to configure and use the CGI Webdriver.
- Chapter 8, "Implementing Security," describes how to implement security in a Web DataBlade module application by using AppPage-level security, database access password encryption, and large object security.
- Chapter 9, "Improving Performance," describes ways of improving the performance of your Web DataBlade module applications by using AppPage caching and large object caching.
- Chapter 10, "Globalizing Your Web DataBlade Module Application," describes how to write and execute AppPages and database content that is multibyte in format.
- Chapter 11, "Deploying Web DataBlade Module Applications," describes how to deploy a Web DataBlade module application from a development environment to a production environment.
- Chapter 12, "Debugging and Troubleshooting," describes how to debug and troubleshoot your Web DataBlade module application.

- Chapter 13, "Web DataBlade Module Utilities," describes the Web DataBlade module utilities.
- Appendix A, "Web DataBlade Module System Tables," describes system tables of the Web DataBlade module.
- Appendix B, "Web DataBlade Module Variables," lists all of the Webdriver variables.

A Notices appendix describes IBM products, features, and services; a glossary of relevant terms follows, and an index directs you to areas of particular interest.

Types of Users

This manual is written for users who administer Web DataBlade module applications and the databases that store the applications.

Software Dependencies

To use the Web DataBlade module, you must use IBM Informix Dynamic Server as your database server. Check the release notes for specific version compatibility. The release notes also list the Web servers that have been certified for this release of the Web DataBlade module.

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a GLS (Global Language Support) locale.

The examples in this manual are written with the assumption that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for date, time, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters, such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see Chapter 10, "Globalizing Your Web DataBlade Module Application," as well as the *IBM Informix Guide to GLS Functionality*.

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 discussed:

- 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 primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
italics ***italics*** *italics*	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface ***boldface***	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
monospace *monospace*	Information that the product displays and information that you enter appear in a monospace typeface.

(1 of 2)

Convention	Meaning
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.
→	This symbol indicates a menu item. For example, "Choose **Tools→Options**" means choose the **Options** item from the **Tools** menu.

(2 of 2)

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.*

Case-Sensitive Text

Variable names used in the Web DataBlade module are case sensitive, are preceded by a dollar sign ($), and consist of alphanumeric and underscore characters. Variables that begin with an underscore are reserved for system use.

Case-Insensitive Text

Tags 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. These tags are known as *AppPage tags*.

The AppPage tags use the SGML processing instruction tag format, <?*tag_info*>, <?/*tag_info*>. An SGML processor ignores tags that it does not recognize, including AppPage tags. Like other SGML processing tags, the AppPage tags and attributes are not case sensitive. You can use uppercase letters, lowercase letters, or any combination of the two.

The text and many of the examples in this manual show function and data type names in mixed lettercasing (uppercase and lowercase). Because IBM Informix Dynamic Server is case insensitive, you do not need to enter function names exactly as shown; you can use uppercase letters, lowercase letters, or any combination of the two.

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 three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
	***Warning:***	Identifies paragraphs that contain vital instructions, cautions, or critical information
	***Important:***	Identifies paragraphs that contain significant information about the feature or operation that is being described
	***Tip:***	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Platform Icons

Platform icons identify paragraphs that contain platform-specific information.

Icon	Description
	Identifies information that is specific to Windows operating systems
	Identifies information that is specific to UNIX operating systems

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 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

This section describes the Web DataBlade module documentation available from Informix:

- Printed documentation
- Online documentation

Printed Documentation

The following Informix manuals are part of the Web DataBlade module documentation set and provide more information about the DataBlade module:

- The *IBM Informix Web DataBlade Module Application Developer's Guide* describes how to develop Web-enabled database applications that dynamically retrieve data from the Informix database.
- The *IBM Informix Web DataBlade Module Administrator's Guide* describes how to administer Web applications that use the Web DataBlade module to dynamically retrieve data from an Informix database. The manual describes topics such as how to configure the Web DataBlade module for your database server; how to configure the NSAPI, Apache, CGI, and ISAPI Webdrivers; how to implement security in your Web applications; and how to increase the performance of your Web applications.

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

- Data Director for Web is a set of Windows tools that allows you to develop and manage Informix-based Web sites and that provides an interface to the Web DataBlade module. For detailed information about Data Director for Web, refer to the *IBM Informix Data Director for Web User's Guide*.
- Before you can use the Web DataBlade module, you must install and configure IBM Informix Dynamic Server. The *Administrator's Guide* for your database server provides information about how to configure IBM Informix Dynamic Server and also contains information about how it interacts with DataBlade modules.
- Once you have installed the Web DataBlade module, you must use BladeManager to register it into the database where the DataBlade module will be used. See the *DataBlade Module Installation and Registration Guide* for details on registering DataBlade modules.
- If you have never used Structured Query Language (SQL), read the *IBM Informix Guide to SQL: Tutorial*. It provides a tutorial on SQL as it is implemented by IBM Informix products. It also describes the fundamental ideas and terminology for planning and implementing an object-relational database.
- A companion volume to the *Tutorial*, the *IBM Informix Guide to SQL: Reference*, includes details of the IBM Informix system catalog tables, describes IBM Informix and common environment variables that you should set, and describes the column data types that IBM Informix database servers support.
- An additional companion volume to the *Reference*, the *IBM Informix Guide to SQL: Syntax*, provides a detailed description of all the SQL statements supported by IBM Informix products. This guide also provides a detailed description of Stored Procedure Language (SPL) statements.
- The *DB-Access User's Manual* describes how to invoke the DB-Access utility to access, modify, and retrieve information from IBM Informix database servers.
- The *Performance Guide* for your database server provides information on how to improve the performance of your SQL queries.

- If you plan to develop your own DataBlade modules using the Web DataBlade module as a foundation, read the *DataBlade Developers Kit User's Guide*. This manual describes how to develop DataBlade modules using BladeSmith, BladePack, and BladeManager.
- When errors occur, you can look them up by number and learn their cause and solution in the *IBM Informix Error Messages* manual. If you prefer, you can look up the error messages in the online message file described in the introduction to the *IBM Informix Error Messages* manual.

Online Documentation

The online documentation for the Web DataBlade module includes release notes and documentation notes.

Release Notes and Documentation Notes

In addition to printed documentation, the following sections describe the online files that supplement the information in this manual. Examine these files before you begin using the Web DataBlade module. They contain vital information about application and performance issues.

UNIX

On UNIX platforms, the following online files appear in the **$INFORMIXDIR/extend/web.*version*** directory, where *version* refers to the current version of the Web DataBlade module.

Online File	Purpose
WEBDOC.TXT	Describes features that are not covered in the manual or that have been modified since publication
WEBREL.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.

♦

Windows

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix** from the Task Bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals, along with information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of IBM Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

♦

IBM Welcomes Your Comments

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

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

Send electronic mail to:

doc@informix.com

We appreciate your suggestions.

Chapter

1

Overview of the Web DataBlade Module

In This Chapter

This chapter provides an overview of the IBM Informix Web DataBlade module. It includes the following topics:

What Is the Web DataBlade Module?

The Web DataBlade module is a collection of SQL functions, data types, tags, and client applications that enables you to create Web applications that dynamically retrieve data from an Informix 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-specific tags (also called *AppPage 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.

AppPages are themselves stored in the database. A Web application that uses the Web DataBlade module, therefore, first retrieves the AppPage from the database and then passes the AppPage through an SQL function that interprets the special AppPage tags and functions, typically to retrieve or update data from database tables and to format the results.

Product Architecture

The Web DataBlade module consists of three main components:

- Webdriver
- The **WebExplode()** function
- Tags and attributes

These components are described in the following sections. The section "Architecture Diagram" on page 1-6 provides an illustration of the architecture of the Web DataBlade module and how the main components work together.

Webdriver

Webdriver is a database client application that builds the SQL queries that execute the **WebExplode()** function to retrieve AppPages from your database. Webdriver returns the HTML that results from calls to the **WebExplode()** function to the Web server.

The Web DataBlade module includes four implementations of Webdriver:

- **NSAPI Webdriver**. This implementation of Webdriver is written with the Netscape Server API and is used only with Netscape Web servers.
- **Apache Webdriver**. This implementation of Webdriver is written with the Apache API and is used only with Apache Web servers.
- **ISAPI Webdriver**. This implementation of Webdriver is written with the Microsoft Internet Information Server API and is used only with Microsoft Internet Information Web servers.
- **CGI Webdriver**. This implementation of Webdriver is a standard CGI program that can be executed by any Web server.

For optimal performance, use the implementation of Webdriver written for your specific Web server. Use the CGI Webdriver only for Web servers that do not have their own implementation of Webdriver.

Important: *This guide uses the term "Webdriver," without a preceding qualifier, to refer to Webdriver functionality that is present in all implementations of Webdriver. The guide uses a qualified term, such as "NSAPI Webdriver," to refer to a specific implementation of Webdriver.*

The WebExplode() Function

The **WebExplode()** function is an SQL function that builds dynamic HTML pages based on data stored in your database. The **WebExplode()** function parses AppPages that contain AppPage tags within HTML and dynamically builds and executes the SQL statements and processing instructions embedded in the AppPage tags. The **WebExplode()** function formats the results of these SQL statements and processing instructions and returns the resulting HTML page to the client application, Webdriver. The SQL statements and processing instructions are specified using SGML-compliant processing tags.

Tags and Attributes

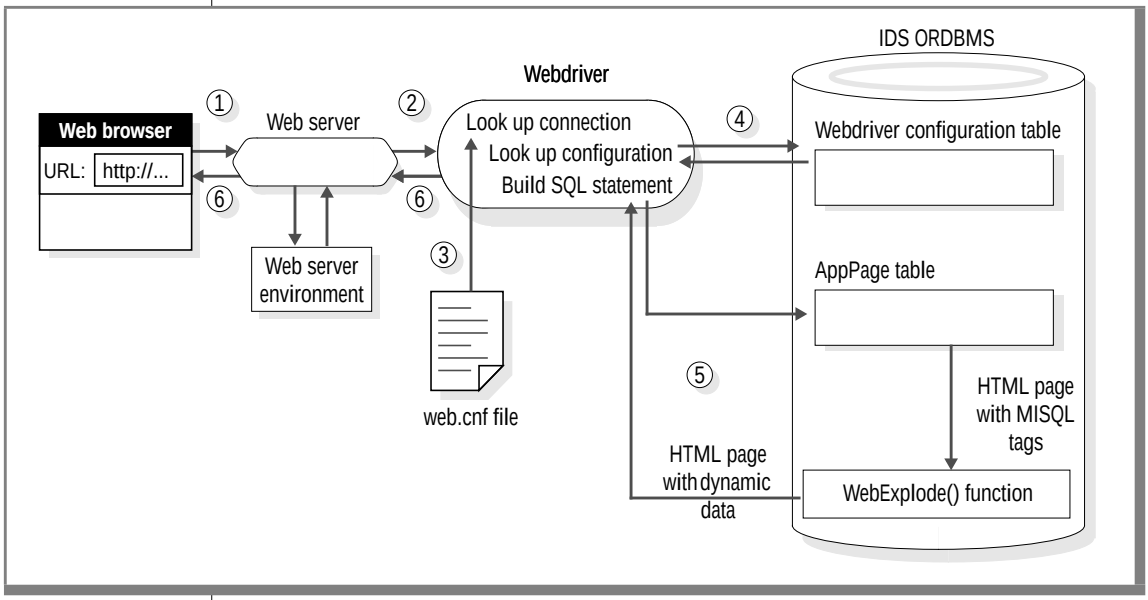
The Web DataBlade module includes a built-in set of SGML-compliant tags and attributes that enable SQL statements to be executed dynamically within AppPages. These tags are referred to as *AppPage tags*.

For example, the MISQL tag allows you to execute an SQL statement, such as SELECT, and format the results of the statement in your AppPage. The MISQL tag has its own attributes, such as SQL, COND, and ERR.

Architecture Diagram

The following diagram illustrates the architecture of the Web DataBlade module. The sequence of events starts with a user typing a URL in a browser and ends with the AppPage rendered in the browser.

Figure 1-1
Web DataBlade Module Architecture



1. A user enters a URL with a Webdriver request and the name of an AppPage in a browser, as shown in the following example:

   ```
   http://ariel:8080/hr_map/?MIval=/welcome.html
   ```

   The browser makes a request to the Web server.

2. The Web server uses its configuration files and information from its environment to determine how to invoke Webdriver. Depending on the type of Webdriver that has been configured for your Web DataBlade module installation, the Web server can execute a CGI program (CGI Webdriver), call a Netscape API shared object (NSAPI Webdriver), call an Apache API object (Apache Webdriver), and so on.

3. Webdriver refers to the **web.cnf** file on the operating system for information on how to connect to an Informix database server, the database to which to connect, the user to connect to the database as, and the Webdriver configuration to use once a connection has been made to the database. Webdriver establishes a connection to the appropriate database with this information.
4. Once Webdriver has established a connection to a database, it looks up the Webdriver configuration in the **WebConfigs** system table. The Webdriver configuration describes, among other things, the AppPage table that contains the AppPage the user requested in the URL originally entered in the browser.
5. Using this schema-related information, Webdriver builds a SELECT statement to retrieve the requested AppPage from the Web application table. The SELECT statement executes the **WebExplode()** function on the AppPage at the same time that it retrieves the AppPage. The **WebExplode()** function expands the AppPage tags within the AppPage and formats the results, resulting in a standard HTML page.
6. Finally, Webdriver returns this HTML page to the Web server, which in turn returns the HTML page to be rendered by the Web browser.

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

Enterprise Replication

The IBM Informix Web DataBlade module contains enterprise replication (ER) support for the HTML data type. The necessary support functions are automatically created in your database when you register the Web DataBlade module on a Version 9.30, or later, server.

Please refer to the *Guide to IBM Informix Enterprise Replication* for information on how to design your replication system as well as how to administer and manage data replication throughout your enterprise.

If you are using a tool like Application Page Builder 2.0 (**APB20**), you must install the tool in each of the replicated sites and set up the tables that contain the HTML content (the wbPages) as well as the User Dynamic Tags and other supporting tables for replication.

If you are using the Administrator's Tool (**adminTool**), you must install the **adminTool** in each of the replicated sites and you must manually update the **webconfigs** table with the **adminTool** at each site.

Important: *It is your responsibility to maintain consistent system information throughout your enterprise. This means that you must ensure that all copies of the tables that your applications utilize are synchronized at all times.*

Converting from a 9.2x Server

If you register this version of the Web DataBlade module in a database on a 9.2x server and then subsequently convert your server to Version 9.30 or later and want to use ER, you must use BladeManager and repeat the registration of the Web DataBlade module in your database, in order to enable ER support. Repeat the registration procedure as follows:

```
% blademgr

myserver> register web.version mydatabase
```

Where *version* is the current version of the Web DataBlade module and *mydatabase* is the database in which you want to register it.

Reverting to a 9.2x Server

If you have registered the Web DataBlade module on a Version 9.30 server and you want to revert your server to Version 9.2x, you must run the **revert93to92.sql** script to drop the Web DataBlade module ER support functions, as follows:

```
cd $INFORMIXDIR/extend/web.version
dbaccess my_database revert93to92.sql
```

Run the **revert93to92.sql** script on each database in which you have registered the Web DataBlade module.

Product Features

The Web DataBlade module includes the following features:

- AppPage tags identify the elements of an HTML page and specify the structure and formatting for that page; they 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 simplify the creation of check box lists, radio button lists, and selection lists.
  - You can also create *user dynamic tags*. A user dynamic tag is a tag that you create and add to the database.

- Webdriver allows you to customize Web applications using information from its configuration file, the Webdriver configurations stored in the database, the Web server environment, URLs, HTML forms, and your own Web application variables, without additional CGI programming. Webdriver also allows you to track persistent session variables between AppPages.
- *AppPage Builder (APB)*, a development tool that is packaged with the Web DataBlade module, provides a user interface to create and update AppPages and to manage multimedia database content.

  APB is itself a Web DataBlade module application made up of linked AppPages.

  APB uses the same database schema as IBM Informix Data Director for Web. Data Director for Web is a set of Windows tools that allows you to develop and manage Informix-based Web sites and that provides an interface to the Web DataBlade module. For detailed information about Data Director for Web, refer to the *IBM Informix Data Director for Web User's Guide*.
- The NSAPI, Apache, and ISAPI implementations of Webdriver allow you to use the proprietary features of the Netscape Web server, Microsoft Internet Information Server, and Apache Web Server, respectively, and eliminate CGI process overhead.
- The Web DataBlade Module Administration Tool, a Web DataBlade module application, provides a browser-based interface to create and update Webdriver mappings and configurations.
- A subset of the examples in this guide and the *IBM Informix Web DataBlade Module Application Developer's Guide* are available in the directory ***INFORMIXDIR*/extend/web.*version*/examples**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Important: *In some examples in this guide, long lines of code wrap to accommodate the fonts used in the guide rather than at the most logical places for the code. Therefore, it is not recommended that you follow these examples exactly when you write your code.*

Chapter

2

Getting Started

In This Chapter

This chapter describes the tasks you must perform to configure the IBM Informix Web DataBlade module for your database server. It includes the following topics:

This chapter is written with the assumption that you have some knowledge of the following topics:

- **Web server configuration and terminology**. Refer to your Web server documentation for detailed information on this topic.
- **Web DataBlade module overview and terminology**. Refer to Chapter 1, "Overview of the Web DataBlade Module," for detailed information on this topic.

Overview of Web DataBlade Module Configuration

The first time you configure the Web DataBlade module for your Informix database server, you must configure *both* the Web server and database components. Configuring Web server components means updating Web server configuration files to communicate with the database server. Configuring database components means installing Web DataBlade module tools into your database, such as the Web DataBlade Module Administration Tool and AppPage Builder.

To configure the Web DataBlade module for the first time

1. Perform all the tasks described in the section "Preconfiguration Tasks," following.
2. Perform the procedure described in "Configuring the Web DataBlade Module for Your Database Server" on page 2-6.

After the initial configuration, you can configure additional databases to use the Web DataBlade module. For detailed information on this topic, refer to "Configuring Additional Databases" on page 2-17.

Preconfiguration Tasks

Before you can configure the Web DataBlade module for your database server, you must perform the following tasks:

1. Install the Web DataBlade module on your database server.

   Refer to the *DataBlade Module Installation and Registration Guide* and the release notes for the Web DataBlade module for detailed information on installing DataBlade modules.
2. Install and configure either the IBM Informix Client Software Developer's Kit or IBM Informix Connect on the Web server computer.

   Refer to the *IBM Informix Client Products Installation Guide for UNIX* or the *IBM Informix Client Products Installation Guide for Microsoft Windows Environments* for detailed information on installing Informix client products.

3. If your Web server computer is different from your database computer, configure the client / server communications between the two computers. This configuration typically involves updating the **$INFORMIXDIR/etc/sqlhosts** and **/etc/services** files on the Web server computer.

   Refer to the *Administrator's Guide* for your database server for detailed information on configuring client / server communications.

4. Create an sbspace of at least 50 MB with the **onspaces** utility. You can create either the default sbspace pointed to by the **SBSPACENAME** parameter of the **ONCONFIG** file or create a different sbspace.

   Be sure you enable logging for the sbspace.

   The following example creates an sbspace named **sbsp1** with an initial offset of 0, a size of 100 MB, and logging turned on:

   ```
   onspaces -c -S sbsp1 -g 2 -p /SBspace/sbsp1 -o 0 -s 100000 -Df
     "LOGGING=ON"
   ```

   Refer to the *Administrator's Guide* for your database server for detailed information on creating sbspaces with the **onspaces** utility and a description of the **SBSPACENAME** parameter.

5. Create a database with logging enabled. If you are going to configure an existing database to use the Web DataBlade module, be sure that logging has been enabled for the database.

   For example, the SQL statement to create the **hr_db** database with logging enabled is shown in the following example:

   ```
   CREATE DATABASE hr_db WITH LOG;
   ```

   Refer to the *IBM Informix Guide to SQL: Syntax* for detailed information on creating databases.

6. If desired, set the Informix environment variables DBDATE and DBCENTURY to customize your environment.

7. Register the Web DataBlade module in your database with BladeManager.

   The following example shows how to use the BladeManager command line interface to register the **web.4.13.UC1** DataBlade module into the **hr_db** database:

   ```
   register web.4.13.UC1 hr_db
   ```

   Refer to the *DataBlade Module Installation and Registration Guide* for detailed information on registering DataBlade modules with BladeManager.

8. Install and configure a Web server on your computer.

   Do not use a Web server installation that has been configured for Versions 3.32 or previous of the Web DataBlade module. Instead, create a new Web server installation.

   Refer to your Web server documentation for information on installing and configuring a Web server.

Important: *If you are configuring the Web DataBlade module for a Windows NT database server, be sure you perform the preceding tasks as a user who has "Informix-Admin" and "Administrator" privileges.*

Configuring the Web DataBlade Module for Your Database Server

Once you have performed the tasks in "Preconfiguration Tasks" on page 2-4, you can configure the Web DataBlade module for your database server. The **websetup** utility performs most of the configuration work.

The **websetup** utility configures the following Web server components:

- The **web.cnf** file
- Web server configuration files

The **websetup** utility also configures the following database components:

- Web DataBlade Module Administration Tool
- AppPage Builder (optionally)

Tip: *For Windows NT, the term "**websetup** utility" refers to the **setup.exe** utility in the directory INFORMIXDIR**extend****web.version****websetup**, where INFORMIXDIR is the main Informix directory and **version** is the current version of the Web DataBlade module installed on your computer.*

Who Should Run the websetup Utility?

Because the **websetup** utility configures both Web server and database components, you need ownership permission on both servers to complete the procedure in "Configuring the Web DataBlade Module: Servers on Same Computer" on page 2-8.

If the same user owns both the Web server and the database, run the **websetup** utility as that user.

If different users own the Web server and database, run the **websetup** utility as the **root** user on UNIX or a user with **Administrator** privileges on Windows NT. The **websetup** utility will then have full permission to execute its various tasks as the appropriate user: the owner of the Web server or the owner of the database.

If, however, you are unable to become the **root** user on your computer, you must run the **websetup** utility twice, once for each type of user, as described in the following procedure:

1. As the owner of the Web server, run the **websetup** utility and select option 3, **Configure Web server components only**.

   The utility guides you through the Web server component configuration.

2. As the database owner, run the **websetup** utility and select option 2, **Configure Database components only**.

   The utility guides you through the database component configuration.

3. Follow the instructions, provided by the **websetup** utility after configuring database components, to correctly update the **web.cnf** file.

If your Web server and database server are on the same computer, follow the instructions in "Configuring the Web DataBlade Module: Servers on Same Computer," following.

If your Web server and database server are on different computers that are both UNIX platforms, follow the instructions in "Configuring the Web DataBlade Module: Servers on Different Computers" on page 2-12.

Configuring the Web DataBlade Module: Servers on Same Computer

This section is written with the assumption that you are configuring the Web DataBlade module for a new database and not upgrading from a 3.32 or earlier version of the Web DataBlade module. If your database is currently registered with a 3.3 or earlier version of the Web DataBlade module and you want to upgrade to version 4.0 or later, refer to the release notes for upgrade instructions.

This section provides two procedures for configuring the Web DataBlade module for your database server; use the first procedure if your database server is on the UNIX platform and use the second procedure on page 2-11 if your database server is on the Windows NT platform.

To configure the Web DataBlade module for your UNIX database server

1. Change to the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

   For example, if the main Informix directory is **/local1/ifmx** and the current version of the Web DataBlade module installed on your computer is **4.13.UC1**, execute the following UNIX command:

   ```
   cd /local1/ifmx/extend/web.4.13.UC1/install
   ```

2. Run the **websetup** utility as the owner of the database and the Web server, or as the **root** user.

   Select option 1, **Configure Web Server and Database Components**. The utility guides you through the setup process.

   The **websetup** utility asks you a variety of questions about your environment, whether you want to install database components such as AppPage Builder, where you want to store the **web.cnf** file, and so on. The **websetup** utility configures the Web DataBlade module for your database server according to your answers.

   You can obtain helpful information about many of the screens by entering the letter h.

   Refer to "Who Should Run the websetup Utility?" on page 2-7 for information about the permissions you need to run the **websetup** utility.

3. If you use, or are planning to use, the MIEXEC AppPage tag to execute Perl programs in your AppPages, add a WEB virtual processor to your database server by adding an appropriate **VPCLASS** parameter of the **ONCONFIG** file.

   For detailed information on this step, refer to "Adding and Starting the WEB Virtual Processor" on page 2-19.

4. If you are using the Apache, ISAPI, or CGI Webdriver, refer to the appropriate chapters in this guide for instructions on how to complete the configuration of these Webdriver implementations. These chapters are:

   - Chapter 5, "Using the Apache Webdriver"
   - Chapter 6, "Using the ISAPI Webdriver"
   - Chapter 7, "Using the CGI Webdriver"

   The **websetup** utility automatically configures the NSAPI Webdriver.

5. Invoke the Web DataBlade Module Administration Tool in your browser by specifying the URL that the **websetup** utility provides at the end of its execution.

   Typically, this URL looks something like the following example:

   ```
   http://domain:port/dbname/admin/
   ```

   The *domain* variable refers to the name of the Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of the database you have just configured for the Web DataBlade module.

   Use the Web DataBlade Module Administration Tool to create new Webdriver mappings and Webdriver configurations. For example, you might want to add a Webdriver mapping to invoke AppPage Builder to begin developing Web applications.

   For detailed information on adding Webdriver mappings and configurations, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

To configure the Web DataBlade module for your Windows NT database server

1. Set the Windows NT system environment variable **INFORMIXDIR** to the full pathname of the main Informix directory.
2. Change to the directory ***INFORMIXDIR*\extend\web.*version*\websetup**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

   For example, if the main Informix directory is **c:\local1\ifmx** and the current version of the Web DataBlade module installed on your computer is **4.13.UC1**, execute the following command at the Windows NT command prompt:

   ```
   cd c:\local1\ifmx\extend\web.4.13.UC1\websetup
   ```
3. Run the **setup.exe** command as the owner of the database and the Microsoft Internet Information server, or as a user who has **Administrator** privileges. This launches the **websetup** utility.

   Select option 1, **Configure Web Server and Database Components**. The utility guides you through the setup process.

   The **websetup** utility asks you a variety of questions about your environment, whether you want to install database components such as AppPage Builder, where you want to store the **web.cnf** file, and so on. The **websetup** utility configures the Web DataBlade module for your database server according to your answers.

   Refer to "Who Should Run the websetup Utility?" on page 2-7 for information about the permissions you need to run the **websetup** utility.
4. If you use, or are planning to use, the MIEXEC AppPage tag to execute Perl programs in your AppPages, add a WEB virtual processor to your database server by adding an appropriate **VPCLASS** parameter of the **ONCONFIG** file.

   For detailed information on this step, refer to "Adding and Starting the WEB Virtual Processor" on page 2-19.
5. Follow the instructions in Chapter 6, "Using the ISAPI Webdriver," to complete the configuration of the ISAPI Webdriver.

6. Invoke the Web DataBlade Module Administration Tool in your browser by specifying the URL that the **websetup** utility provides at the end of its execution.

   Typically, this URL looks something like the following example:

   ```
   http://domain:port/dbname/admin/
   ```

   The *domain* variable refers to the name of the Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of the database you have just configured for the Web DataBlade module.

   Use the Web DataBlade Module Administration Tool to create new Webdriver mappings and Webdriver configurations. For example, you might want to add a Webdriver mapping to invoke AppPage Builder to begin developing Web applications.

   For detailed information on adding Webdriver mappings and configurations, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

Configuring the Web DataBlade Module: Servers on Different Computers

This section describes how to configure the Web DataBlade module when the Web server and database server are on different computers. This procedure is for UNIX platforms only.

1. Log onto the database server computer as the owner of the database for which you are going to configure the Web DataBlade module.
2. Configure the database components by following the instructions in the section "Configuring Database Components," following.
3. Log onto the Web server computer as the user **informix**.
4. Configure the Web server components by following the instructions in the section "Configuring Web Server Components" on page 2-15.

5. Invoke the Web DataBlade Module Administration Tool in your browser by specifying the URL that the **websetup** utility provides at the end of its execution.

   Typically, this URL looks something like the following example:

   ```
   http://domain:port/dbname/admin/
   ```

   In this example, *domain* is the name of the Web server computer.

   The variable *port* is the port number of the Web server process.

   The variable *dbname* is the name of the database you have just configured for the Web DataBlade module.

6. Use the Web DataBlade Module Administration Tool to create new Webdriver mappings and Webdriver configurations. For example, you might want to add a Webdriver mapping to invoke AppPage Builder to begin developing Web applications.

For detailed information on adding Webdriver mappings and configurations, refer to Chapter 3, "Configuring Webdriver."

Configuring Database Components

This section describes step 2 of the procedure in previous section; it explains how to configure the database components. Perform these steps on the database server computer.

1. Copy the sample Webdriver configuration file, **web.cnf.example**, from the **$INFORMIXDIR/extend/web.*version*/install** directory to the **/tmp** directory (*version* is the version of the Web DataBlade module that you are using, for example, 4.13.UC1).

   Rename the **web.cnf.example** file to **web.cnf**.

2. Go to the directory **$INFORMIXDIR/extend/web.*version*/install**.

3. Run the **websetup** utility and select option 2, Configure Database Components Only. The utility guides you through the setup process.

   When the utility asks you for the directory that contains the **web.cnf** file, enter: `/tmp`.

   The **websetup** utility asks you questions about your environment and whether you want to install database components such as App-Page Builder. The utility configures the Web DataBlade module for your database server according to your answers. You can obtain helpful information about many of the screens by entering the letter `h`.

4. Delete the file **/tmp/web.cnf**.

5. Go to the directory **$INFORMIXDIR/extend/web.*version*/utils**.

6. Run the **webconfig** utility with the following command:

   ```
   webconfig -addmap -p /dbname/admin -n admin
     -d dbname -u user -o /tmp/temp_map
   ```

   In this command, *dbname* is the name of the database for which you are configuring the Web DataBlade module and *user* is the owner of the database.

   The **-o temp_map** option of the **webconfig** utility creates a file in the current directory called **/tmp/temp_map** that contains a Webdriver mapping. You import this file later when you are configuring the Web server components.

7. Go to the directory **$INFORMIXDIR/extend/web.*version***.

8. Execute the UNIX **tar** command to create a TAR file that contains the necessary files for configuring the Web server components on the Web server computer, as shown:

   ```
   tar cvf /tmp/WebBundle.tar install netscape apache utils
     doc
   ```

   The **tar** command creates a TAR file called **/tmp/WebBundle.tar** that contains the directories: **install**, **netscape**, **apache**, **utils**, and **doc**.

9. If you use, or are planning to use, the MIEXEC AppPage tag to execute Perl programs in your AppPages, add a WEB virtual processor to your database server by adding an appropriate VPCLASS parameter of the ONCONFIG file.

   For detailed information about this step, refer to "Adding and Starting the WEB Virtual Processor" on page 2-19.

Configuring Web Server Components

This section describes step 4 of the procedure in "Configuring the Web DataBlade Module: Servers on Different Computers" on page 2-12; it explains how to configure the Web server components. Perform these steps on the Web server computer.

1. Copy the files **/tmp/temp_map** and **/tmp/WebBundle.tar** (that you created while configuring the database components on the database server computer) to the **/tmp** directory on the Web server computer.
2. Go to the main Informix directory (**$INFORMIXDIR**) on your Web server computer that contains the Informix client files.
3. Create a directory called **extend/web.*version***. For example, if the current version of the Web DataBlade module is 4.13.UC1, execute the following command:

   ```
   mkdir extend/web.4.13.UC1
   ```
4. Go to this new directory.
5. Using the **tar** command, extract the files from the /**tmp/WebBundle.tar file** into this directory. For example:

   ```
   tar -xvf /tmp/WebBundle.tar
   ```
6. Go to the **install** directory that is created by the previous step, as in:

   ```
   cd install
   ```
7. Run the **websetup** utility and select option 3, Configure Web Server Components Only. The utility guides you through the setup process.

   The **websetup** utility asks you questions about your environment and the location of your Web server. The utility configures the Web server components according to your answers.

   Be sure to note the directory in which the **websetup** utility stores the **web.cnf** Webdriver configuration file.

   You can obtain helpful information about the screens by entering the letter h.
8. Set the environment variable MI_WEBCONFIG to the full pathname of the **web.cnf** file created by the **websetup** utility.
9. Go to the directory **$INFORMIXDIR/extend/web.*version*/utils**.

10. Run the **webconfig** utility with the **-i** option as shown:

    ```
    webconfig -addmap -i /tmp/temp_map
    ```

    This imports the **/tmp/temp_map** file that you copied to the **/tmp** directory into the **web.cnf** file.

11. If you are using the Apache, ISAPI, or CGI Webdriver, refer to the appropriate chapters in this guide for instructions on how to complete the configuration of these Webdriver implementations:

    - Chapter 5, "Using the Apache Webdriver"
    - Chapter 6, "Using the ISAPI Webdriver"
    - Chapter 7, "Using the CGI Webdriver"

Configuring Additional Databases

This section describes how you can configure the Web DataBlade module for additional databases.

This section is written with the assumption that you have already performed the initial configuration of the Web DataBlade module for your database server, as described in "Configuring the Web DataBlade Module for Your Database Server" on page 2-6.

To configure an additional database to use the Web DataBlade module

1. Create a database with logging enabled. If you are going to configure an existing database to use the Web DataBlade module, be sure that logging has been enabled for the database.

   For example, the SQL statement to create the **hr_db** database with logging enabled is shown in the following example:

   ```
   CREATE DATABASE hr_db WITH LOG;
   ```

   Refer to the *IBM Informix Guide to SQL: Syntax* for detailed information on creating databases.

2. Register the Web DataBlade module in your database with BladeManager.

   The following example shows how to use the BladeManager command line interface to register the **web.4.13.UC1** DataBlade module into the **hr_db** database:

   ```
   register web.4.13.UC1 hr_db
   ```

   Refer to the *DataBlade Module Installation and Registration Guide* for detailed information on registering DataBlade modules with BladeManager.

3. Change to the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

   For example, if the main Informix directory is **/local1/ifmx** and the current version of the Web DataBlade module installed on your computer is **4.13.UC1**, execute the following UNIX command:

   ```
   cd /local1/ifmx/extend/web.4.13.UC1/install
   ```

4. Run the **websetup** utility.

   Select option 2, **Configure Database Components Only**. The utility guides you through the setup process.

   The **websetup** utility asks you questions about your environment, the location of the **web.cnf** file, and whether you want to install database components such as AppPage Builder. The **websetup** utility configures the Web DataBlade module for your database according to your answers.

   You can obtain helpful information about the screens by entering the letter h.

5. Add a */dbname/*admin URL prefix to your Web server, where *dbname* refers to the name of the database you are configuring.

   The following chapters, describing the NSAPI, Apache, ISAPI, and CGI Webdrivers, provide information on adding URL prefixes to the corresponding Web server:

   - Chapter 4, "Using the NSAPI Webdriver"
   - Chapter 5, "Using the Apache Webdriver"
   - Chapter 6, "Using the ISAPI Webdriver"
   - Chapter 7, "Using the CGI Webdriver"

6. Stop and restart your Web server.

7. Invoke the Web DataBlade Module Administration Tool in your browser by specifying the URL that the **websetup** utility provides at the end of its execution.

   Typically, this URL looks something like the following example:

   ```
   http://domain:port/dbname/admin/
   ```

   The *domain* variable refers to the name of the Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of the database you have just configured for the Web DataBlade module.

   Use the Web DataBlade Module Administration Tool to create new Webdriver mappings and Webdriver configurations. For example, you might want to add a Webdriver mapping to invoke AppPage Builder to begin developing Web applications.

   For detailed information on adding Webdriver mappings and configurations, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

Adding and Starting the WEB Virtual Processor

The Web DataBlade module uses a special virtual processor called WEB to execute the internal database server function that performs the work of the MIEXEC tag. Therefore, if you use, or are planning to use, the MIEXEC tag to execute Perl programs in your AppPages, you must update the **ONCONFIG** file and add a **VPCLASS** parameter to start the WEB virtual processor.

To add and start the WEB virtual processor

1. Add the following line to the **ONCONFIG** file:

   ```
   VPCLASS WEB,num=1,noyield
   ```

   The **num** parameter specifies the number of WEB virtual processors that the database server starts. Although you can add as many as you want, keep in mind that the more virtual processors you start, the more resources they use. Refer to the *Administrator's Guide* for your database server for information on the optimal number of virtual processors.

   The **noyield** parameter specifies that the WEB virtual processor runs to completion before it processes the next request. You *must* include the **noyield** parameter when you specify the WEB virtual processor.

2. Restart the database server.

   The database server reads the new **ONCONFIG** file and automatically starts a WEB virtual processor when needed.

3. Verify that the WEB virtual processor has been registered with the database server by executing the **onstat** command at the UNIX shell prompt, as shown in the following example:

   ```
   INFORMIXDIR/bin/onstat -g glo
   ```

   INFORMIXDIR refers to the main Informix directory.

   Under the heading **Virtual processor summary**, you should see an entry for the WEB virtual processor.

For detailed information about virtual processors, the **ONCONFIG** file, the **VPCLASS** parameter of the **ONCONFIG** file, the **onstat** command, and stopping and starting the database server, refer to the *Administrator's Guide* for your database server.

Chapter

3

Configuring Webdriver

In This Chapter

This chapter describes how to configure Webdriver. It includes the following topics:

Overview of Webdriver

Webdriver is one of the main components of the Web DataBlade module. It provides the interface between a Web server and an Informix database. When a user invokes an *AppPage* by either specifying a URL in a browser or clicking on a link in an HTML page, Webdriver dynamically retrieves the AppPage, as well as results of SQL statements, from the database and passes the resulting HTML page to the Web server. AppPages are HTML pages that include Web DataBlade module specific tags (also called AppPage tags) and functions that dynamically execute the SQL statements you specify and format the results. To design a Web application, you create AppPages and specify the flow between them.

A Web server calls Webdriver through an NSAPI, ISAPI, Apache, or CGI interface. Webdriver obtains configuration information about the Web application environment from the following sources:

- The Webdriver configuration file, called **web.cnf** by default
- The Webdriver configuration, stored in the **WebConfigs** system table in the database
- The Web browser
- The Web server environment

Important: *This chapter uses the term "Webdriver," without a preceding qualifier, to refer to Webdriver functionality that is present in all implementations of Webdriver. The chapter uses a qualified term, such as "NSAPI Webdriver," to refer to a specific implementation of Webdriver.*

Webdriver Variables

Webdriver obtains configuration information about Web DataBlade module applications from its own set of variables. There are two types of Webdriver variables: those that reside in the **web.cnf** file and those that reside in the database.

Webdriver Variables in the web.cnf File

Webdriver uses the Webdriver variables that reside in the **web.cnf** file to connect to a database, set global Webdriver variables, and set Informix environment variables, such as **INFORMIXDIR**, and **INFORMIXSERVER**. The Webdriver variables that are used to connect to a particular database are collectively known as *Webdriver mappings*.

Important: *Do not set the Informix environment variables DBDATE and DBCENTURY in your* ***web.cnf*** *file. Their settings will be ignored. Instead, set them in your environment before you register the DataBlade module in your database.*

The names of Webdriver mappings are reflected in certain *URL prefixes* defined for a Web server. URL prefixes are the URLs that client applications send to the Web server to invoke HTML pages, execute CGI programs, call Web server plug-ins, and so on.

As part of the Web DataBlade module configuration for your database, you configure the Web server so that the URL prefixes that match the names of Webdriver mappings invoke Webdriver. This means that when a user types in a browser a URL prefix that matches the name of a Webdriver mapping, the Web server calls Webdriver, which in turn makes a connection to a database and invokes the appropriate AppPage. For this reason, Webdriver mappings can be thought of as the *glue* between a Web server and the database server.

The Web server finds the location of the **web.cnf** file using the **MI_WEBCONFIG** variable, typically set in the script that starts the Web server.

Webdriver Variables in the Database

Webdriver uses the variables that reside in the database to enable Webdriver features, such as AppPage caching, and to retrieve user-defined variable definitions.

When Webdriver consults a Webdriver mapping to make a connection to a database, the Webdriver mapping also specifies the *Webdriver configuration* to use. A Webdriver configuration is the name given to a set of Webdriver variables and their values that reside in the database. A Webdriver configuration includes the Webdriver variables to enable Webdriver features and user-defined variables.

When you install the Web DataBlade Module Administration Tool, described in "The Web DataBlade Module Administration Tool" on page 3-9, into your database, the tool automatically creates the **apb2**, **ddw**, and **admin** Webdriver configurations. The **apb2** Webdriver configuration, for example, defines the Webdriver variables to invoke the AppPages that make up AppPage Builder (APB).

Important: *The Web DataBlade Module Administration Tool also creates a Webdriver configuration called* ***apb****. The* ***apb*** *Webdriver configuration is used to invoke an older version of APB, used in Versions 3.3 and earlier of the Web DataBlade module. This older version of APB uses a different database schema to store AppPages than that of the new version of APB included in Version 4.0 and later of the Web DataBlade module. The previous version of APB uses the Webdriver variables* ***MItab****,* ***MIcol****, and* ***MInam*** *to specify the AppPage table schema; the new version of APB uses the* ***wbextensions*** *table to specify the schema. The* ***wbextensions*** *table is discussed in a later section in this guide.*

Unless you currently use the previous version of APB to store your AppPages, you should always specify the ***apb2*** *Webdriver configuration in the Webdriver mapping used to invoke APB, to use the new APB as a tool for developing AppPages. Although Version 4.0 and later of the Web DataBlade module supports the previous APB and the use of the* ***MItab****,* ***MIcol****, and* ***MInam*** *Webdriver variables, future releases of the DataBlade module might not. Refer to the release notes for information about migrating Web DataBlade module applications from the previous schema to the new schema that uses the* ***wbextensions*** *table.*

Feature-Related Webdriver Variables

The feature-related Webdriver variables are those that enable specific Webdriver features, such as AppPage caching or large object support. You do not have to include these Webdriver variables in a Webdriver configuration; you only add and set them when you want to enable a particular feature.

For example, if you set the following feature-related Webdriver variables, you enable AppPage caching for Web DataBlade module applications that use the Webdriver configuration:

- **cache_page**
- **cache_directory**
- **cache_admin**
- **cache_admin_password**

The **apb2** Webdriver configuration, for example, does not by default include these Webdriver variables, because AppPage caching is turned off by default. However, by adding these Webdriver variables to the **apb2** Webdriver configuration, you enable AppPage caching for the APB application.

Specific feature-related Webdriver variables are described in detail in the sections of this guide that describe the feature and in sections of the *IBM Informix Web DataBlade Module Application Developer's Guide* that describe the feature.

User-Defined Webdriver Variables

A Webdriver configuration can also optionally include user-defined variables. User-defined variables are variables that an AppPage can access but are not used by Webdriver.

For example, you can use the Web DataBlade Module Administration Tool to set a user-defined variable called **company_name** for your Webdriver configuration. Then, in your AppPages, you can use the MIVAR tag to access the value of the **company_name** user-defined variable instead of explicitly specifying the name of the company.

Overwriting Webdriver Variables in a URL

When you use the Web DataBlade Module Administration Tool to add a Webdriver variable to a Webdriver configuration or to change its value, you can specify whether certain Webdriver variables can be overwritten in the URL that calls the AppPage.

The **MIval** Webdriver variable should *always* be overwritable, since **MIval** is the variable that identifies the unique name of the AppPage you want to invoke and you invoke many different AppPages in a Web application. Refer to "How Webdriver Locates AppPages" on page 3-8 for detailed information on the **MIval** Webdriver variable.

You can set the following Webdriver variables as overwritable:

- **MIval** (*always* set as overwritable)
- **MIqry2pass**
- **MI_WEBACCESSLEVEL**
- **MI_WEBGROUPLEVEL**

You cannot set the Webdriver variables not included in the preceding list as overwritable. For example, almost all the feature-related Webdriver variables are not overwritable. This is because feature-related Webdriver variables are used to change the way Webdriver behaves and are never used in AppPages.

Important: *The Web DataBlade Module Administration Tool allows you to check the* ***Overwrite*** *check box for these feature-related variables. However, Webdriver ignores this setting and never overwrites the value of these Webdriver variables.*

How Webdriver Locates AppPages

Webdriver uses the **wbextensions** table in combination with the **MIval** Webdriver variable to locate an AppPage stored in a database table.

You invoke an AppPage by either typing a URL directly in a browser or by clicking a link that the browser resolves into a URL. This URL always includes the **MIval** variable that specifies the unique name of an AppPage, as shown in the following example:

```
http://ariel:8080/hr_map/?MIval=/hr_app/welcome.html
```

In the example, `ariel:8080` specifies the domain name and port number of the Web server process, `/hr_map` is the Webdriver mapping, and `/hr_app/welcome.html` is the unique name of an AppPage.

Webdriver first parses the value of the **MIval** variable into three distinct pieces: an extension, an ID, and a path. In the example, `html` is the extension, `welcome` is the ID, and `/hr_app` is the path.

Webdriver then uses the extension value to query the **wbextensions** table to determine in which database table the AppPage is stored. If you use APB as your development tool, AppPages with the extension `html` are stored in the **wbPages** table.

Webdriver then uses all three pieces of the **MIval** variable (extension, ID, and path) to query the AppPage table and return the AppPage.

For detailed information on invoking AppPages and the **wbextensions** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

The Web DataBlade Module Administration Tool

The Web DataBlade Module Administration Tool is a Web DataBlade module application that allows you to add, update, or delete Webdriver mappings and Webdriver configurations for the database to which you are connected.

You must install the Web DataBlade Module Administration Tool in each database in which the Web DataBlade module has been registered, usually with the **websetup** utility. You run the **websetup** utility when you initially configure the Web DataBlade module for your database. For detailed information on running the **websetup** utility, refer to Chapter 2, "Getting Started."

For detailed information about the Web DataBlade Module Administration Tool and how to use it to modify or add Webdriver mappings and Webdriver configurations, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

The Webdriver Configuration File (web.cnf)

The Webdriver configuration file contains information about the connection between the Web server and your Web DataBlade module application. Webdriver reads this file to find out information such as how to connect to the Informix database server, which database to connect to, and what global variables have been set.

The Webdriver configuration file also contains a mapping between the URL prefixes of the Web server and the Webdriver configurations stored in an Informix database. URL prefixes are the URLs that client applications send to the Web server to invoke HTML pages, execute CGI programs, call Web server plug-ins, and so on.

Important: *You can name the Webdriver configuration file anything you want. By convention, however, this file is called* ***web.cnf****. For clarity and consistency, this book uses the name* ***web.cnf*** *for the Webdriver configuration file.*

You usually never have to manually update the Map sections of the **web.cnf** file because the Web DataBlade Module Administration Tool does this for you when you update the corresponding Webdriver mapping. However, if you need to update or add Webdriver variables to the Global or Setvar sections of the **web.cnf** file, then you must do so manually since the Web DataBlade Module Administration Tool cannot update these sections of the **web.cnf** file.

The Web DataBlade Module Administration Tool is described later in this chapter, in the section "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

File Permissions of the web.cnf File

The **web.cnf** file must be owned by the user who owns the Web server process. This means that, for example, if the Web server process is owned by the user **nobody**, then the **web.cnf** file must also be owned by the user **nobody**.

In addition, the **web.cnf** file should have read and write permissions for the owner of the file and read permissions for everyone else.

Format of the web.cnf File

The **web.cnf** file uses SGML-like syntax to store the information needed to connect the Web server with your Web DataBlade module application. The information typically takes the following form:

```
<Section>
variable1   value1
variable2   value2
...
</Section>
```

Section refers to the section of the **web.cnf** file that is being described and determines the variables you can include in the description.

You can include the following three sections in the **web.cnf** file: Global, Setvar, and Map. The following sections describe each section of the file.

Global Section of the web.cnf File

The Global section of the **web.cnf** file describes the Webdriver variables that are global to the client connection to the database.

When Webdriver makes a connection to a database, it automatically includes the variables in the Global section of the **web.cnf** file in every Webdriver configuration in every database in the database server. These variables (except for **debug_level**) cannot be overwritten for a particular Webdriver configuration.

The following table lists all the variables you can set in the Global section of the **web.cnf** file.

Variable	Mandatory?	Description
dbconnmax	No	Specifies the maximum number of connections to the database The default value is `16`.
anchorvar	Yes	Specifies the name of the anchor variable used when an AppPage calls another AppPage This variable is mandatory. For the NSAPI and Apache Webdrivers, **anchorvar** should always be set to `WEB_HOME`, with a trailing forward slash (/). For the ISAPI Webdriver, the variable should be set to `WEB_HOME/drvisapi.dll`. For the CGI Webdriver, the variable should be set to `WEB_HOME/webdriver`. Since **anchorvar** is always set to **WEB_HOME**, you can always use **WEB_HOME** as an anchor variable in any AppPage.
driverdir	No	Specifies the directory that Webdriver uses to internally coordinate its interaction with the Web server The default value of this variable is `/tmp`. This variable is only used by the Apache and CGI implementations of Webdriver.
debug_file	No	Specifies the full pathname of the log file to which Webdriver messages are written For more information on Webdriver tracing, refer to Chapter 12, “Debugging and Troubleshooting.”

(1 of 2)

Variable	Mandatory?	Description
debug_level	No	Enables Webdriver tracing to the log file specified by the **debug_file** variable For more information on Webdriver tracing, refer to Chapter 12. You can override the value of the **debug_level** variable in the Global section of the **web.cnf** file by setting it in your Webdriver configuration using the Web DataBlade Module Administration Tool.
maxcharsize	No	When set to a value greater than `1`, *each* character sent to the **WebExplode()** function is URL-encoded. If this variable is not set, Webdriver URL-encodes only special characters (such as `&`) before sending it to the **WebExplode()** function. Informix recommends you set this variable to a value greater than `1` *only* if you are using a multibyte character set. This is because you might see a degradation in performance if Webdriver is forced to URL-encode every character before sending it to the **WebExplode()** function. You can override the value of this variable for your Webdriver mapping by adding it as a Webdriver variable to the appropriate Webdriver configuration.
config_user	No	The name of the user who is allowed to use the Web DataBlade Module Administration Tool Add this variable to the **web.cnf** file only with the **webconfig** utility. For more information about this variable, refer to "Securing the Web DataBlade Module Administration Tool" on page 3-28.
config_password	No	The password of the **config_user** user Add this variable to the **web.cnf** file only with the **webconfig** utility. For more information about this variable, refer to "Securing the Web DataBlade Module Administration Tool" on page 3-28.

(2 of 2)

The following example shows a Global section of a **web.cnf** file:

```
<Global>
dbconnmax          10
anchorvar          WEB_HOME/
debug_file         /disk1/webdriver.log
debug_level        0xffff
config_user        admin_user
config_password    94928760340384028730928 64
</Global>
```

The variables in this Global section show that the maximum number of Webdriver connections to all databases at any one time is 10. **WEB_HOME** is the anchor variable. Webdriver tracing has been enabled, and all Webdriver messages are written to the file **/disk1/webdriver.log**. The user allowed to use the Web DataBlade Module Administration Tool, specified by the **config_user** variable, is `admin_user`.

Setvar Section of the web.cnf File

The Setvar section of the **web.cnf** file describes environment variables, or variables that you can also set on UNIX or Windows. In particular, the Setvar section describes the Informix environment variables such as **INFORMIXDIR**, and **INFORMIXSERVER**.

When Webdriver makes a connection to a database, it automatically includes the variables in the Setvar section of the **web.cnf** file in every Webdriver configuration in every database in the database server.

The following example shows the Setvar section of a **web.cnf** file:

```
<Setvar>
INFORMIXDIR    /disk1/informix
INFORMIXSERVER myserver
</Setvar>
```

Map Section of the web.cnf File

The Map section of the **web.cnf** file describes the mapping between a URL prefix on the Web server and a Webdriver configuration stored in a database. The variables in the Map section describe how Webdriver connects to a particular database. The collection of variables that make up a single Map section in the **web.cnf** file is called a *Webdriver mapping*.

Although you can include only one Global and Setvar section in the **web.cnf** file, you can include many Map sections, one for each different Webdriver mapping.

The format of a Map section in the **web.cnf** file is slightly different from that of the Global and Setvar sections:

```
<Map path=/URL_prefix>
variable1   value1
variable2   value2
....
</Map>
```

The */URL_prefix* variable refers to the name of the Webdriver mapping. This is the *same* name as that of the Web server URL prefix that you specify when you invoke AppPages in a browser. The variables specified for the */URL_prefix* Webdriver mapping describe the database to connect to, the user to connect to the database as, and the name of the Webdriver configuration stored in the **WebConfigs** table to use.

Each database in which the Web DataBlade module has been registered should have a Webdriver mapping called */dbname*/admin, where *dbname* refers to the name of the database. This means you should also define a */dbname*/admin URL prefix for your Web server. Use this URL prefix to invoke the Web DataBlade Module Administration Tool for the *dbname* database. For example, the URL prefix to invoke the Web DataBlade Module Administration Tool for the **hr_db** database should be /hr_db/admin. For more information on the */dbname*/admin Webdriver mapping, refer to "Setting Up the Web DataBlade Module Administration Tool" on page 3-22.

Informix recommends that the format for URL prefixes used to invoke Web DataBlade module applications other than the Web DataBlade Module Administration Tool include a descriptive name. For example, a good name for a URL prefix to invoke a Human Resources application is /hr_app.

Important: *Use the Web DataBlade Module Administration Tool to add new Webdriver mappings and to enter URL prefixes to ensure that Webdriver is automatically updated with the new information.*

The following table lists all the variables that can be included in the Map section of the **web.cnf** file.

Map Variable	Mandatory?	Description
database	Yes	The name of the database to which Webdriver connects when a URL prefix specifies this Webdriver mapping
user	Yes	The name of the user who connects to the database specified by the **database** variable
password	Yes	The encrypted password of the user specified by the **user** variable
password_key	Yes	The key that Webdriver uses to decrypt the password specified by the **password** variable
server	No	The Informix database server to use when making the connection to the database If this variable is not set, the connection is made using the **INFORMIXSERVER** database server.
config_name	Yes	The name of the Webdriver configuration to use The Webdriver configuration is stored in the **WebConfigs** system table in the database specified by the **database** variable.
config_security	No	When set to `ON`, security is enabled for this Webdriver mapping, which means that only the user specified by the **config_user** variable in the Global section of the **web.cnf** file can use this Webdriver mapping. The **config_security** variable should appear only in Webdriver mappings used to invoke the Web DataBlade Module Administration Tool. Refer to "Securing the Web DataBlade Module Administration Tool" on page 3-28 for more information on this variable.

The following example shows a Map section in a **web.cnf** file for the URL prefix `/hr_app`:

```
<Map path=/hr_app>
database       hr_db
user           hr_user
password       849284903403840243432432 4
password_key   hr_key
server         hr_server
config_name    hr_config
</Map>
```

If a URL invokes an AppPage with the /hr_app URL prefix, Webdriver connects to the **hr_db** database as the **hr_user** user, using the **hr_server** Informix database server instead of the default **myserver** database server specified by the **INFORMIXSERVER** variable in the Setvar section. Once Webdriver connects to the **hr_db** database, it uses the **hr_config** Webdriver configuration stored in the **WebConfigs** system table to determine the Webdriver variables.

Example of the web.cnf File

The following example shows a complete **web.cnf** file that includes Global, Setvar, and Map sections:

```
<Global>
dbconnmax         10
anchorvar         WEB_HOME/
debug_file        /disk1/webdriver.log
debug_level       0xffff
config_user       admin_user
config_password   94928760340384028730928 64
</Global>

<Setvar>
INFORMIXDIR       /disk1/informix
INFORMIXSERVER    myserver
</Setvar>

<Map path=/hr_db/admin>
database          hr_db
user              hr_user
password          84928490340384024343243 24
password_key      hr_key
config_name       admin
config_security   ON
</Map>

<Map path=/hr_app>
database          hr_db
user              hr_user
password          84928490340384024343243 24
password_key      hr_key
server            hr_server
config_name       hr_config
</Map>
```

Variables in the Global Section

The variables in the Global section indicate that the maximum number of Webdriver connections to all databases at any one time is 10. **WEB_HOME** has been set as the anchor variable. Webdriver tracing has been enabled, and all Webdriver messages are written to the file **/disk1/webdriver.log**. The user allowed to use the Web DataBlade Module Administration Tool, specified by the **config_user** variable, is `admin_user`.

Variables in the Setvar Section

The variables in the Setvar section set the Informix environment variables **INFORMIXDIR**, and **INFORMIXSERVER**.

Variables in the Map Section

Two Webdriver mappings are defined: `/hr_db/admin` and `/hr_app`.

The `/hr_db/admin` URL prefix in a URL invokes the Web DataBlade Module Administration Tool for the **hr_db** database. It connects to the database as the **hr_user** user. Once Webdriver connects to the **hr_db** database, it uses the special Webdriver configuration called **admin** to bring up the Web DataBlade Module Administration Tool. The special security feature of the Web DataBlade Module Administration Tool is enabled, since the **config_security** variable is set to `ON`.

If a URL invokes an AppPage with the `/hr_app` URL prefix, Webdriver connects to the **hr_db** database as the **hr_user** user, using the **hr_server** Informix database server instead of the default **myserver** database server specified by the **INFORMIXSERVER** variable in the Setvar section. Once Webdriver connects to the **hr_db** database, it uses the **hr_config** Webdriver configuration stored in the **WebConfigs** system table to determine the Webdriver variables.

For more detailed information on using the Web DataBlade Module Administration Tool, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

Setting the MI_WEBCONFIG Environment Variable

The **MI_WEBCONFIG** environment variable lets the Web server find the location of the **web.cnf** file. The type of Webdriver you are using (NSAPI, ISAPI, Apache, or CGI) determines how and where you set this environment variable.

The **MI_WEBCONFIG** variable should be set to the *full* pathname of the **web.cnf** file. For example, if the **web.cnf** file is located in the directory **/local/informix_info**, the **MI_WEBCONFIG** environment variable should be set to the value `/local/informix_info/web.cnf`.

The following table describes how and where to set the **MI_WEBCONFIG** environment variable depending on the type of Webdriver you are using.

Type of Webdriver	How to Set MI_WEBCONFIG
NSAPI	Set the variable in the Netscape Web server startup file *and* in the environment of the user who starts the Netscape Web server processes. The Netscape server startup file is usually a shell script that you execute to start the Netscape Web server processes.
ISAPI	Set **MI_WEBCONFIG** as a system environment variable. To set system environment variables on Windows NT, choose **Start→Settings→Control Panel** and double-click the **System** icon. Click the **Environment** tab, and add the value to the first list box.
Apache	Set the variable in the Apache Web server startup file. This file is usually a shell script that you execute to bring up the Apache Web server processes. If you do not use a shell script to start up the Apache Web server because you execute the **httpd** process directly at the operating system prompt, be sure the **MI_WEBCONFIG** environment variable is set in the environment of the user who starts the Apache Web server.
CGI	CGI Webdriver ignores the environment variable **MI_WEBCONFIG** and *always* looks for the **web.cnf** file in the same directory as the CGI Webdriver program.

You must also set the **MI_WEBCONFIG** variable in your operating system environment if you use any of the Web DataBlade module utilities, such as **webconfig**.

Managing Webdriver Connections to the Database

The Webdriver configuration file (**web.cnf**) contains information that Webdriver uses to connect to an Informix database. You can modify the behavior of these connections for specific Webdriver configurations by setting the Webdriver variables described in the following table.

Webdriver Variable	Mandatory?	Description
connection_life	No	Specifies the life of a connection, or in other words, the maximum number of requests (an integer value) that Webdriver makes to the database before the connection is shut down and reestablished The default value is 100. You should set this Webdriver variable to another value only under the guidance of Technical Support.
connection_wait	No	Specifies the amount of time, in milliseconds, that Webdriver yields and waits to establish a connection if Webdriver was unable to make the initial connection due to the maximum number of database connections having already been reached The maximum number of Webdriver connections to the database server is specified by the **dbconnmax** Webdriver variable in the Global section of **web.cnf** file.

(1 of 3)

Webdriver Variable	Mandatory?	Description
connect_as_user	No	When set to ON, specifies that Webdriver establish the connection to the database as the user specified by the **REMOTE_USER** Web browser variable and not as the user specified in the Map section of the **web.cnf** file By default, if this Webdriver variable is not set, Webdriver always establishes connections to the database as the user specified by the **user** Webdriver variable in the appropriate Map section of the **web.cnf** file. This Webdriver variable applies only to the NSAPI, ISAPI, and Apache implementation of Webdriver. In addition, you can only use this Webdriver variable if you have enabled user authentication for the corresponding Web server.
connect_user_max	No	Specifies the maximum number of connections that Webdriver establishes as the user specified by the **REMOTE_USER** Web browser variable The default value of this Webdriver variable is 1. The **connect_user_max** Webdriver variable can only be set in conjunction with the **connect_as_user** Webdriver variable. This Webdriver variable applies only to the NSAPI, ISAPI, and Apache implementation of Webdriver. In addition, you can only use this Webdriver variable if you have enabled user authentication for the corresponding Web server. For details on enabling user authentication, refer to Chapter 4, "Using the NSAPI Webdriver," Chapter 5, "Using the Apache Webdriver," and Chapter 6, "Using the ISAPI Webdriver."
query_timeout	No	Specifies the maximum number of seconds that Webdriver allows a query to run before Webdriver interrupts the query.
keepalive	No	Specifies the interval in seconds at which Webdriver checks the Web browser connection If the browser is no longer connected because a STOP or CANCEL signal has been sent by the browser, the running query is interrupted, and the Web server is freed to execute the next query request. This variable applies only to the NSAPI, ISAPI, and Apache implementation of Webdriver.

(2 of 3)

Webdriver Variable	Mandatory?	Description
init_sql	No	Specifies that Webdriver send initial SQL statements to the database server when Webdriver makes a connection to the database Set this Webdriver variable to one or more SQL statements, separated by semicolons and terminated by a carriage return. Do not include quotes. For example, if you want to set the isolation level of the connection to the database to dirty read, set the **init_sql** Webdriver variable to the value `SET ISOLATION TO DIRTY READ;`
max_html_size	No	Specifies the largest AppPage, in bytes, that Webdriver sends to the browser AppPages larger than this size are not sent to the browser. The default value for this Webdriver variable is 128 KB.
maxcharsize	No	When set to a value greater than `1`, *each* character sent to the **WebExplode()** function is URL-encoded. If this variable is not set, Webdriver URL-encodes only special characters (such as `&`) before sending it to the **WebExplode()** function. It is recommended that you set this variable to a value greater than `1` *only* if you are using a multibyte character set. This is because you might see a degradation in performance if Webdriver is forced to URL-encode every character before sending it to the **WebExplode()** function. You can specify the **maxcharsize** variable in the Global section of the **web.cnf** file if you want to specify globally that characters should be URL-encoded. By adding the variable to a Webdriver configuration, however, you can control this behavior for a single Webdriver configuration and not for the whole database server.

(3 of 3)

Use the Web DataBlade Module Administration Tool to set these Webdriver variables for your Webdriver configuration. For detailed information on using the Web DataBlade Module Administration Tool, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

Using Server-Side Includes in AppPages

Server-side includes define a mechanism for including dynamic text in AppPages. Server-side includes are special command codes that are recognized and interpreted by the Web server. The Web server places the output of the commands in the AppPage before the Web server sends the AppPage to the browser. You can use server-side includes to embed, for example, a date or time stamp in the text of the AppPage.

If you want to use server-side includes in your AppPages, you must use either the NSAPI or Apache Webdriver; the ISAPI and CGI Webdrivers do not recognize server-side includes. By default, the NSAPI Webdriver is automatically configured to recognize server-side includes. For the Apache Webdriver, however, you must perform an extra configuration step for it to recognize server-side includes. For detailed information, refer to Chapter 5, "Using the Apache Webdriver."

Setting Up the Web DataBlade Module Administration Tool

The Web DataBlade Module Administration Tool is a Web DataBlade module application with client-side JavaScript that allows you to add, update, or delete Webdriver mappings and Webdriver configurations for the database to which you are connected.

You must install the Web DataBlade Module Administration Tool in every database in which the Web DataBlade module is registered, typically with the **websetup** utility.

Important: *You can use the Web DataBlade Module Administration Tool to add, update, or delete Webdriver configurations and Webdriver mappings only in the database to which you are connected.*

Webdriver Mappings

Webdriver mappings, defined in the Map section of the **web.cnf** file, are the set of the Webdriver variables that are used to connect to a particular database. The Map sections provide a mapping between a Web server URL prefix and a Webdriver configuration stored in the database. When a user specifies a URL prefix in the URL used to invoke an AppPage, Webdriver uses the Map entry in the **web.cnf** file to determine the database to which to connect, the user to connect to the database as, and the Webdriver configuration to use once connected to the database.

Webdriver Configurations

The term *Webdriver configuration* refers to the set of Webdriver and user-defined variables, and their values, that reside in the database. Webdriver configurations are associated with a particular Web DataBlade module application. Each configuration can include Webdriver variables that are used to configure Webdriver features, such as AppPage caching, large object caching, AppPage-level security, and so on.

For example, a **sales** Webdriver configuration might enable AppPage caching (**cache_page** and **cache_directory** Webdriver variables) and enable AppPage-level security (**MIpagelevel** Webdriver variable). A **human_resources** Webdriver configuration might include unrelated Webdriver variables.

You can include disabled Webdriver variables in a Webdriver configuration. This means that although the Web DataBlade Module Administration Tool lists the Webdriver variable as being part of the Webdriver configuration, the Webdriver variable does not affect the way Webdriver behaves. This feature is useful when you develop applications and need to disable and enable Webdriver functionality many times and you do not want to keep adding and deleting the variable from the Webdriver configuration.

Installing the Administration Tool in Your Database

This section describes how to install the Web DataBlade Module Administration Tool in a new database. The procedure shows how to create the tool's database schema, load the AppPages, add a special Webdriver mapping to the **web.cnf** file to invoke the Web DataBlade Module Administration Tool, and add a URL prefix to your Web server configuration file that maps to the URL prefix specified in the special Webdriver mapping.

You must have already used BladeManager to register the Web DataBlade module in the database before you execute the following procedure. The procedures in this section use an example database called **production**.

Tip: *The Web DataBlade Module Administration Tool might already be installed in your database if you used the* ***websetup*** *utility to initially configure the Web DataBlade module for your database. The* ***websetup*** *utility also adds a special Webdriver mapping to the* ***web.cnf*** *file to bring up the Web DataBlade Module Administration Tool. If you are using the NSAPI Webdriver, the* ***websetup*** *utility might also have updated the Netscape configuration file (****obj.conf****) with the necessary Web DataBlade module information.*

Check your ***web.cnf*** *file to see if it contains a Webdriver mapping of the form* `/dbname/admin`*, where* `dbname` *refers to the name of your database. Also check your database to see if it contains the tables* ***WebConfigs****,* ***WebCMPages****,* ***WebCMImages****, and* ***WebEnvVariables****, which are the Web DataBlade Module Administration Tool system tables. If any of these mappings or tables exist, the Web DataBlade Module Administration Tool is already configured for your database, and you can invoke the tool as described in* *"Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29**.*

To configure the Web DataBlade Module Administration Tool for a new database

1. Execute the **cm_schema_create** and **cm_schema_load** utilities to create the Web DataBlade Module Administration Tool database schema and load its AppPages and initial Webdriver configurations into the new database.

   For detailed information on this step, refer to "Creating and Loading the Tool's Schema" on page 3-26.

2. Use the **webconfig** utility to add a special Webdriver mapping called */dbname/*admin to the **web.cnf** file used to invoke the Web DataBlade Module Administration Tool for this database.

   For detailed information on this step, refer to "Executing the webconfig Utility" on page 3-27.

3. Make the */dbname/*admin URL prefix known to your Web server.

   You typically use your Web server's administration server to perform this step.

   Be sure that the name of the new Web server URL prefix is *exactly* the same as the name of the special Webdriver mapping you created in step 2.

   The following chapters provide information on this step, for NSAPI, Apache, and ISAPI Web servers: Chapter 4, "Using the NSAPI Webdriver," Chapter 5, "Using the Apache Webdriver," and Chapter 6, "Using the ISAPI Webdriver."

Once you have completed all the preceding steps you should be able to invoke the Web DataBlade Module Administration Tool for your new database by using the */dbname/*admin URL prefix in your URL, as described in "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

Creating and Loading the Tool's Schema

To create the Web DataBlade Module Administration Tool schema, use the **cm_schema_create** utility, passing it the name of the database and the name of an sbspace, as shown in the following example:

```
cm_schema_create production sbspace
```

To load the AppPages into the **WebCMPages** system table, use the **cm_schema_load** utility, as shown in the following example:

```
cm_schema_load production
```

The **cm_schema_load** utility also loads all the Webdriver variables into the **WebEnvVariables** system table and the following three initial Webdriver configurations into the **WebConfigs** system table:

- **admin**. This is the Webdriver configuration that the Web DataBlade Module Administration Tool uses.

  This Webdriver configuration does *not* show up in the list of existing Webdriver configurations in the Web DataBlade Module Administration Tool because you should never modify it.
- **apb2**. This is the Webdriver configuration to invoke AppPage Builder (APB).
- **ddw**. This is the Webdriver configuration for IBM Informix Data Director for Web.

  For information on using IBM Informix Data Director for Web, refer to the *IBM Informix Data Director for Web User's Guide*.

Important: *The **cm_schema_load** utility also loads a Webdriver configuration called **apb** that invokes a previous (pre-Version 4.0) version of APB. You should use this Webdriver configuration only if you are using the previous version of APB. New users should always use the **apb2** Webdriver configuration.*

The **cm_schema_create** and **cm_schema_load** utilities are located in the directory *INFORMIXDIR*/**extend/web.*version*/admtool**, where *INFORMIXDIR* is the main Informix directory and ***version*** is the version of the Web DataBlade module installed on your computer.

For detailed information on using the **cm_schema_create** and **cm_schema_load** utilities, refer to "The cm_schema_create Utility" on page 13-3 and "The cm_schema_load Utility" on page 13-5, respectively.

Executing the webconfig Utility

The **webconfig** utility adds new Webdriver mappings to the **web.cnf** file. When you configure the Web DataBlade Module Administration Tool for your database, you use the **webconfig** utility to add a special Webdriver mapping that you use to invoke the Web DataBlade Module Administration Tool.

Before you execute the **webconfig** utility, be sure you set the **MI_WEBCONFIG** environment variable to point to the full pathname of the **web.cnf** file. For more information on this environment variable, refer to "Setting the MI_WEBCONFIG Environment Variable" on page 3-18.

Although you can call this special Webdriver mapping anything you want, it is recommended that you call it `/`*`dbname`*`/admin`, where *dbname* refers to the name of the database for which you are configuring the Web DataBlade Module Administration Tool.

You *must* specify the **admin** Webdriver configuration with the **-n** option to the **webconfig** utility.

For example, to add a special Webdriver mapping for the Web DataBlade Module Administration Tool for the **production** database and the **admin_user** user, execute the following command:

```
webconfig -addmap -p /production/admin -n admin -d production -u admin_user
```

The **webconfig** utility asks for the password for user **admin_user** and a password key.

The resulting Map section in the **web.cnf** file looks something like the following example:

```
<Map path=/production/admin>
database          production
user              admin_user
password          84928490340384024343243 24
password_key      admin_key
config_name       admin
config_security   ON
</Map>
```

For detailed information on using the **webconfig** utility, refer to "The webconfig Utility" on page 13-8.

Securing the Web DataBlade Module Administration Tool

Although you can use the features described in Chapter 8, "Implementing Security," to secure the Web DataBlade Module Administration Tool for your database, the methods might not always be adequate to ensure that no other user has access to the tool. For this reason, the Web DataBlade module provides an added security feature to specifically secure the Web DataBlade Module Administration Tool.

Because this security feature also uses Web server authentication, you can only use it with the NSAPI, ISAPI, and Apache Webdrivers. This feature has no effect when used with CGI Webdriver.

To enable the Web DataBlade Module Administration Tool security feature

1. Be sure the */dbname*/admin URL prefix in the Web server uses Web server authentication.

   For more information on enabling Web server authentication for a Webdriver mapping, refer to the appropriate NSAPI, ISAPI, or Apache Webdriver chapter.

2. Set the **MI_WEBCONFIG** environment variable to point to the full pathname of the **web.cnf** file.

3. Execute the **webconfig** utility with the **-secure** option at the operating system prompt to enable the security feature, as shown in the following example:

   ```
   webconfig -secure
   ```

   The **webconfig** utility asks for the name and password of the user allowed to use the Web DataBlade Module Administration Tool. The utility updates the **config_user** and **config_password** variables in the Global section of the **web.cnf** file with this information. Only the encrypted password is actually written to the **web.cnf** file.

Once you have enabled the security feature for the Web DataBlade Module Administration Tool, you must provide the name and password of the user specified in step 3 every time you invoke the tool. The password is compared to the password stored in the Global section of the **web.cnf** file. If the password matches, you are allowed to use the tool; otherwise, you are not allowed to even view the tool.

Invoking and Using the Web DataBlade Module Administration Tool

To invoke the Web DataBlade Module Administration Tool, use the URL prefix you added to the Web server configuration file when you configured your database to use the Web DataBlade Module Administration Tool. This URL prefix is typically `/`*`dbname`*`/admin`, where *dbname* refers to the name of the database to which you want to connect.

For example, if your Web server is on a computer called **ariel** at port number **8080**, the following URL will invoke the Web DataBlade Module Administration Tool in your Web browser for the database **production**:

```
http://ariel:8080/production/admin/
```

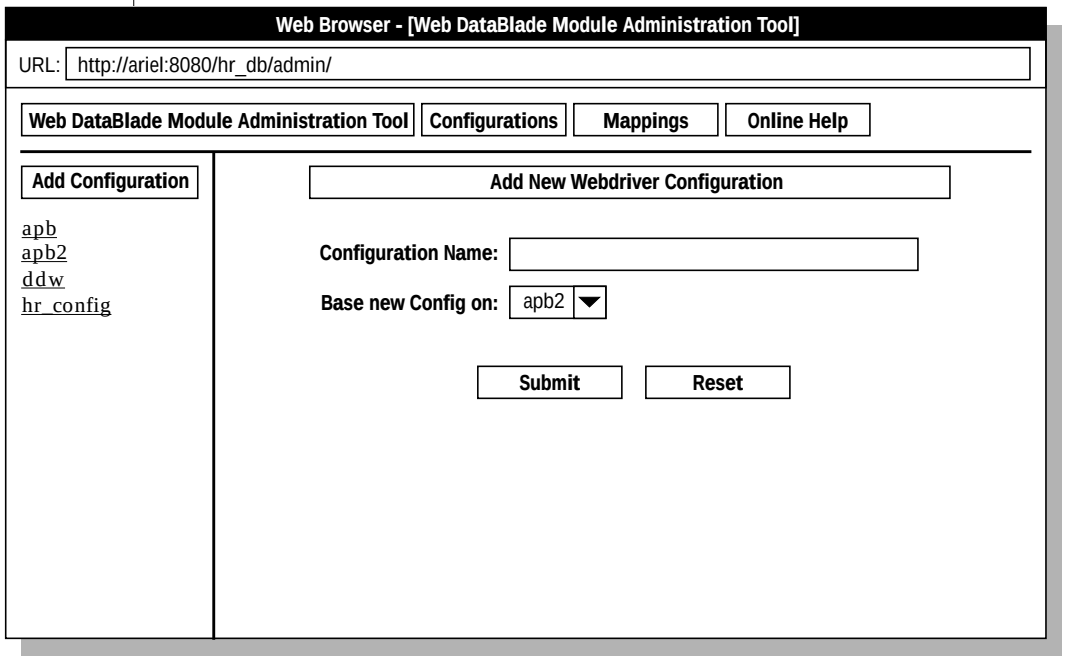
Tip: *Many Web servers require you to add the extra slash at the end of the URL.*

The `/`*`dbname`*`/admin` URL prefix is added to your Web server's configuration file and the **web.cnf** file when the Web DataBlade module is initially configured for your database. This can happen automatically with the **websetup** utility or manually if you have manually configured the Web DataBlade Module Administration Tool for your database.

For more information on configuring the Web DataBlade Module Administration Tool for new databases, refer to "Setting Up the Web DataBlade Module Administration Tool" on page 3-22.

The main Web DataBlade Module Administration Tool AppPage is divided into three frames, as shown in the following figure.

Figure 3-1
Main Web DataBlade Module Administration Tool AppPage



The top frame includes the name of the tool and buttons for viewing Webdriver configuration or Webdriver mapping information. It also includes a button to invoke help about the tool.

The left frame lists either the Webdriver configurations or the Webdriver mappings that currently exist in the database. The type of information that is displayed depends on which button you click in the top frame.

The left frame also contains a button to add a new Webdriver configuration or Webdriver mapping. The type of button that is displayed depends on which button you click in the top frame.

The right frame displays the details of the Webdriver configuration or Webdriver mapping that you select in the left frame.

Viewing Existing Webdriver Configurations

This section describes how to view the Webdriver configurations that currently exist in the database to which you are connected.

To view all the Webdriver configurations

1. Click **Configurations** in the top frame of the Web DataBlade Module Administration Tool.

   A list of all existing Webdriver configurations appears in the left frame below the **Add Configuration** button.

2. To view the details of a Webdriver configuration, click its name.

   A table appears in the right frame, listing all the Webdriver variables currently associated with the Webdriver configuration, as shown in Figure 3-2 on page 3-33. The table contains the following columns:

   - The **Variable** column lists the name of the Webdriver variable. Click the button labelled **?** to the left of each Webdriver variable to bring up help information about the Webdriver variable.
   - The **Value** column shows the current value of the Webdriver variable.
   - The **Overwrite** column specifies whether the variable can be overwritten in the URL used to bring up AppPages. For example, the Webdriver variable **MIval** should *always* be overwritable since this is the variable used to identify the AppPage you want to bring up in your browser. Only certain Webdriver variables can be overwritten; for the full list, refer to "Overwriting Webdriver Variables in a URL" on page 3-7.

     If **Overwrite** is checked, the corresponding Webdriver variable is overwritable; if the check box is unchecked, the Webdriver variable is not overwritable.
   - The **Disable** column specifies whether the Webdriver variable is disabled or enabled. When the Webdriver variable is disabled, it does not have any effect on the way Webdriver behaves. By default, all Webdriver variables are enabled.

     If **Disable** is checked, the corresponding Webdriver variable is disabled; if the check box is unchecked, the Webdriver variable is enabled.

3. To sort, in ascending alphabetical order, the list of Webdriver variables and the list of current Webdriver variable values, click the **Variable** or **Value** column header, respectively. Click the **Overwrite** or **Disable** column header to group all respective checked and unchecked items together.

Important: *The Web DataBlade Module Administration Tool discards any changes to the current AppPage that have not been saved to the database before it sorts the list of Webdriver variables.*

The other buttons on the page, such as the **Remove** button to the right of the **Disable** column, are used when you edit Webdriver configurations. They are described in the following section.

Editing an Existing Webdriver Configuration

This section describes how to edit an existing Webdriver configuration.

The following list describes the types of changes you can make to a Webdriver configuration:

- Change the current value, the overwrite specification, or the disable specification for a Webdriver or user-defined variable.

  For detailed instructions on how to make this type of change, refer to "Changing the Current Value of a Webdriver or User-Defined Variable" on page 3-34.

- Add a new Webdriver or user-defined variable to the Webdriver configuration.

  For detailed instructions on how to make this type of change, refer to "Adding a New Webdriver or User-Defined Variable" on page 3-35.

- Delete a Webdriver or user-defined variable from the Webdriver configuration.

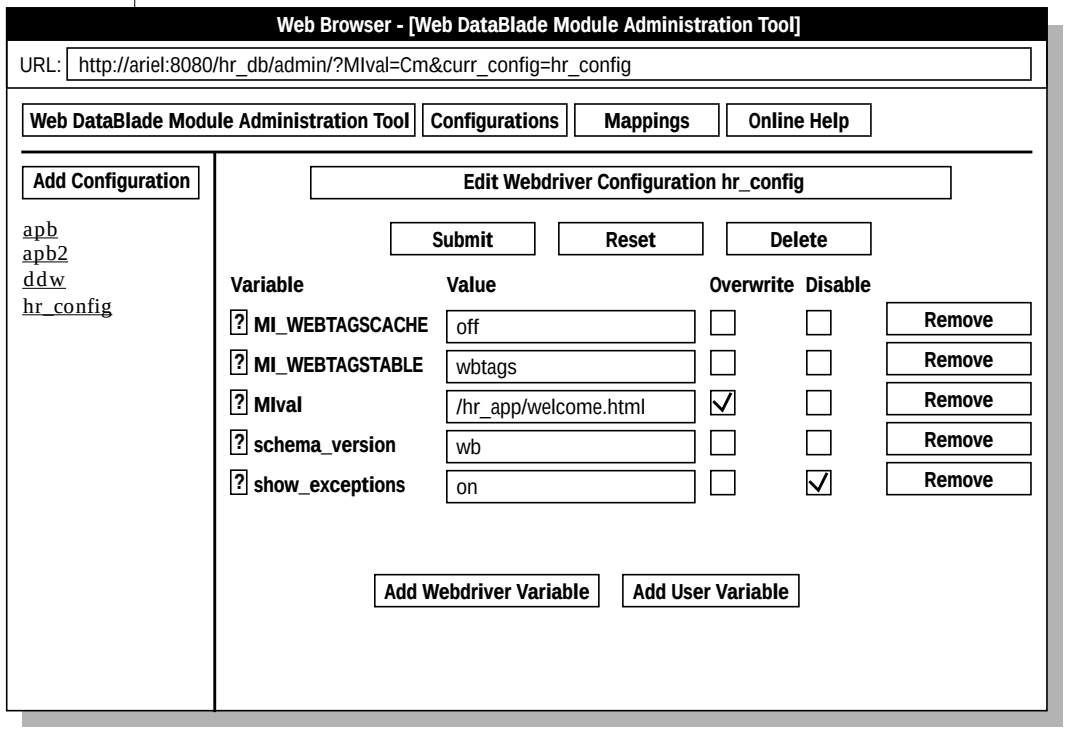
  For detailed instructions on how to make this type of change, refer to "Adding a New Webdriver or User-Defined Variable" on page 3-35.

Webdriver variables are all those variables defined and used by the Web DataBlade module, such as **redirect_url**, **cache_admin**, and **MIusername**. For a complete list of Webdriver variables, refer to Appendix B, "Web DataBlade Module Variables."

User-defined variables are variables defined by the AppPage developer and used in AppPages. The Web DataBlade module does not need to use these variables to function correctly.

The following figure shows the **Edit Webdriver Configuration** <*config_name*> AppPage of the Web DataBlade Module Administration Tool you use to edit Webdriver configurations.

Figure 3-2
AppPage to Edit a Webdriver Configuration



Changing the Current Value of a Webdriver or User-Defined Variable

This section describes how to use the Web DataBlade Module Administration Tool to edit a Webdriver configuration by changing the current value of a Webdriver or user-defined variable.

To change the current value of a Webdriver or user-defined variable

1. Click **Configurations** in the top frame of the Web DataBlade Module Administration Tool.

   A list of all existing Webdriver configurations appears in the left frame below the **Add Configuration** button.

2. Click the name of the Webdriver configuration.

   The **Edit Webdriver Configuration <***config_name***>** AppPage appears in the right frame.

3. Enter the new value of the Webdriver or user-defined variable in the **Value** column of the table that lists all the Webdriver variables for the Webdriver configuration.

   The new value of the Webdriver variable is validated against an internal list of possible values for the variable; if the new value does not match one of the possible values, an error is returned. For example, the **session** Webdriver variable can only be set to `cookie`, `url`, or `auto`; therefore, the Web DataBlade Module Administration Tool will not let you change the value of the **session** variable to anything other than these possible values.

4. To change whether a Webdriver or user-defined variable can be overwritten in the URL that calls the AppPage, check its **Overwrite** check box.

   If **Overwrite** is checked for a Webdriver or user-defined variable, the variable *can* be overwritten in the URL. Only certain Webdriver variables can be overwritten; for the full list, refer to "Overwriting Webdriver Variables in a URL" on page 3-7.

5. To change whether a Webdriver or user-defined variable is disabled, check its **Disable** check box. If **Disable** is checked for a Webdriver or user-defined variable, the variable is disabled.

6. Click **Submit** to update the database with the new information.

   All the modifications to the database, including inserts, updates, and deletes, for the Webdriver configuration are executed in a single database transaction.

Important: *No changes are made to the database until you click* ***Submit****.*

After the Web DataBlade Module Administration Tool makes the changes to the database, it returns one of two messages. The tool returns the following message if the Webdriver configuration is not assigned to a Webdriver mapping:

```
Changes have been saved to the database
```

The tool returns the following message if the Webdriver configuration is assigned to a Webdriver mapping:

```
Changes have been saved to the database. The configuration
config_name has been updated in cache.
```

Click **Reset** instead of **Submit** to set the Webdriver variables back to their value after the last change to the database.

Adding a New Webdriver or User-Defined Variable

This section describes how to use the Web DataBlade Module Administration Tool to edit a Webdriver configuration by adding a new Webdriver or user-defined variable.

To add a new Webdriver or user-defined variable

1. Click **Configurations** in the top frame of the Web DataBlade Module Administration Tool.

   A list of all existing Webdriver configurations appears in the left frame below the **Add Configuration** button.

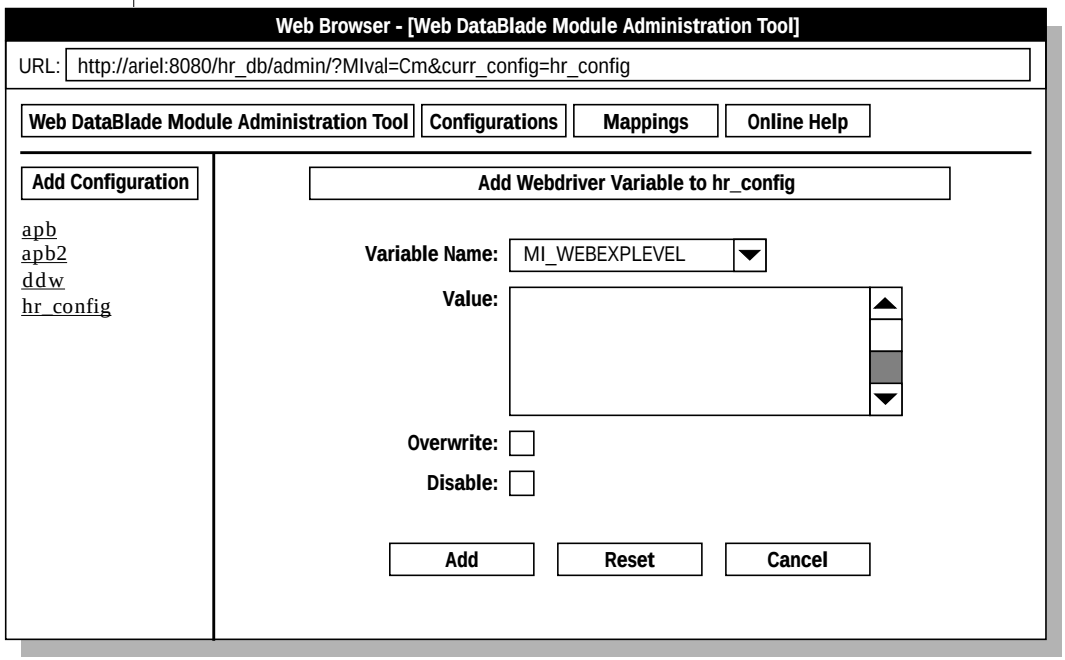
2. Click the name of the Webdriver configuration.

   The **Edit Webdriver Configuration <***config_name***>** AppPage appears in the right frame.

3. Click **Add Webdriver Variable** or **Add User Variable**, whichever is appropriate.

   When you click **Add Webdriver Variable**, the **Add Webdriver Variable to <***config_name***>** AppPage appears, as shown in the following figure.

Figure 3-3
AppPage to Add a Webdriver Variable to an Existing Webdriver Configuration



4. In the list box, select the name of the Webdriver variable you want to add.
5. Enter the value for the Webdriver variable in the **Value** text box.

   If the Webdriver variable has a specified list of possible values, the default value automatically appears in the **Value** text box.
6. Check the **Overwrite** check box if you want the Webdriver variable to be overwritten in the URL that brings up the AppPage.

   Only certain Webdriver variables can be overwritten; for the full list, refer to "Overwriting Webdriver Variables in a URL" on page 3-7.

7. Check the **Disable** check box if you want to disable the Webdriver variable so that it has no affect on the way Webdriver behaves.
8. Click **Add**.

   If you tried to set the value of a Webdriver variable that has a specified list of possible values to anything other than one of those possible values, an error is returned. Otherwise, the **Edit Webdriver Configuration** <*config_name*> AppPage reappears, with the new Webdriver variable listed in the table.
9. Click **Submit** to update the database with the new information.

   All the modifications to the database, including insertions, updates, and deletions, for the Webdriver configuration are executed in a single database transaction.

Important: *No changes are made to the database until you click* ***Submit****.*

After the Web DataBlade Module Administration Tool makes the changes to the database, it returns one of two messages. The tool returns the following message if the Webdriver configuration is not assigned to a Webdriver mapping:

```
Changes have been saved to the database
```

The tool returns the following message if the Webdriver configuration is assigned to a Webdriver mapping:

```
Changes have been saved to the database. The configuration
config_name has been updated in cache.
```

Click **Reset** instead of **Submit** to set all the Webdriver variables back to their value after the last change to the database.

Adding a user-defined variable is very similar to adding a Webdriver variable, except that you enter the name of the user-defined variable in a text box instead of selecting the name from a list box.

Deleting a Webdriver or User-Defined Variable

This section describes how use the Web DataBlade Module Administration Tool to edit a Webdriver configuration by deleting a Webdriver or user-defined variable.

To delete a Webdriver or user-defined variable

1. Click **Configurations** in the top frame of the Web DataBlade Module Administration Tool.

   A list of all existing Webdriver configurations appears in the left frame below the **Add Configuration** button.

2. Click the name of the Webdriver configuration.

   The **Edit Webdriver Configuration <***config_name***>** AppPage appears in the right frame.

3. Click **Remove** for the variable you want to delete.

   The variable is removed from the table that lists the variables for the Webdriver configuration.

4. Click **Submit** to update the database with the new information.

   All the modifications to the database, including insertions, updates, and deletions, for the Webdriver configuration are executed in a single database transaction.

Important: *No changes are made to the database until you click* ***Submit****.*

After the Web DataBlade Module Administration Tool makes the changes to the database, it returns one of two messages. The tool returns the following message if the Webdriver configuration is not assigned to a Webdriver mapping:

```
Changes have been saved to the database
```

The tool returns the following message is the Webdriver configuration is assigned to a Webdriver mapping:

```
Changes have been saved to the database. The configuration
config_name has been updated in cache.
```

Click **Reset** instead of **Submit** to set all the Webdriver variables back to their values after the last change to the database.

Adding a New Webdriver Configuration

This section describes how to add a new Webdriver configuration.

When you create a new Webdriver configuration, you always base it on an existing Webdriver configuration. The installation of the Web DataBlade Module Administration Tool creates the following two Webdriver configurations, which you can use as a basis for a new Webdriver configuration:

- **apb2:** Use this Webdriver configuration if you want to access APB. The **apb2** Webdriver configuration includes the following Webdriver variables:
  - **MI_WEBTAGSCACHE**
  - **MI_WEBTAGSTABLE**
  - **MIval**
  - **schema_version**
  - **show_exceptions**
- **ddw:** Use this Webdriver configuration if you want access IBM Informix Data Director for Web. The **ddw** Webdriver configuration includes the same Webdriver variables as the **apb2** configuration but with a different value for the **MIval** Webdriver variable. This means that the different Webdriver configurations invoke different default AppPages.

  For more information on IBM Informix Data Director for Web, refer to the *IBM Informix Data Director for Web User's Guide*.

The **apb2** and **ddw** Webdriver configurations both use the same database schema to store AppPages.

Important: *The Web DataBlade Module Administration Tool also adds an* ***admin*** *Webdriver configuration when you initially install the tool in a database. You use the* ***admin*** *Webdriver configuration to access the Web DataBlade Module Administration Tool. For this reason, you should never modify or delete the* ***admin*** *Webdriver configuration, and therefore, it does not appear in the list of Webdriver configurations in the left frame of the tool.*

In addition, the tool also adds a Webdriver configuration called ***apb****. The* ***apb*** *Webdriver configuration is used to invoke an older version of APB, used in Versions 3.3 and earlier of the Web DataBlade module. This older version of APB uses a different database schema to store AppPages than that of the new version of APB included in Version 4.0 and later of the Web DataBlade module. The old version of APB uses the Webdriver variables* ***MItab****,* ***MIcol****, and* ***MInam*** *to specify the AppPage table schema; the new version of APB uses the* ***wbextensions*** *table to specify the schema. The* ***wbextensions*** *table is discussed in a later section in this guide.*

Unless you currently use the previous version of APB to store your AppPages, you should always use the new APB as a tool for developing AppPages by specifying the ***apb2*** *Webdriver configuration in the Webdriver mapping used to invoke APB. Although Version 4.0 and later of the Web DataBlade module supports the previous APB and the use of the* ***MItab****,* ***MIcol****, and* ***MInam*** *Webdriver variables, future releases of the DataBlade module might not. Refer to the release notes for information about migrating Web DataBlade module applications from the previous schema to the new schema that uses the* ***wbextensions*** *table.*

To add a new Webdriver configuration

1. Click **Configurations** in the top frame of the Web DataBlade Module Administration Tool.
2. Click **Add Configuration**.

   The **Add New Webdriver Configuration** AppPage appears in the right frame.
3. Enter the name of the new Webdriver configuration.

   Do not enter the name of an existing Webdriver configuration, listed in the left frame.

4. Select an existing Webdriver configuration on which the new Webdriver configuration will be based.

   You can select one of the two default Webdriver configurations, **apb2** or **ddw**, or a Webdriver configuration you have previously created. All Webdriver variables currently defined for the Webdriver configuration you select are automatically included in the new Webdriver configuration.

Important: *When you base a new Webdriver configuration on an existing Webdriver configuration, the Webdriver variables of the existing configuration are physically copied to the new configuration. This means, for example, that if you subsequently edit the existing Webdriver configuration, these changes will not be reflected in your new Webdriver configuration.*

5. Click **Submit**.

   When you click **Submit**, the following events happen:

   - Your new Webdriver configuration is added to the Web DataBlade Module Administration Tool system catalogs.
   - The name of the new Webdriver configuration appears in the left frame of the Web DataBlade Module Administration Tool.
   - The **Edit Webdriver Configuration** <*config_name*> AppPage automatically appears in the right frame so you can immediately start editing your new Webdriver configuration. For detailed information on editing your new Webdriver configuration, refer to "Editing an Existing Webdriver Configuration" on page 3-32.

To reset all the values in the **Add New Webdriver Configuration** AppPage, click **Reset** instead of **Submit**.

Deleting an Existing Webdriver Configuration

This section describes how to delete an existing Webdriver configuration.

To delete an existing Webdriver configuration

1. Click **Configurations** in the top frame of the Web DataBlade Module Administration Tool.

   A list of all existing Webdriver configurations appears in the left frame below the **Add Configuration** button.

2. Click the name of the Webdriver configuration.

   The **Edit Webdriver Configuration <***config_name***>** AppPage appears in the right frame.

3. Click **Delete**.

   A confirmation message appears, asking you if you really want to delete the Webdriver configuration.

4. Click **Yes**.

   The name of the Webdriver configuration is removed from the list of Webdriver configurations in the left frame, and the **Edit Webdriver Configuration <***config_name***>** AppPage in the right frame disappears and is replaced with the **Add New Webdriver Configuration** AppPage.

Viewing Existing Webdriver Mappings

This section describes how to view all the Webdriver mappings currently in the **web.cnf** file for the database to which you are connected.

To view existing Webdriver mappings

1. Click **Mappings** in the top frame of the Web DataBlade Module Administration Tool.

   A list of all existing Webdriver mappings appears in the left frame below the **Add Mapping** button.

   The Webdriver mappings that map to databases other than the one to which you are connected do not appear in the list.

Important: *The Webdriver mapping used to bring up the Web DataBlade Module Administration Tool for a particular database does not appear in the left frame because this Webdriver mapping should never be modified.*

2. To view the details of a Webdriver mapping, click its name.

   The full details of the Webdriver mapping appear in the right frame. These details include the name of the Webdriver configuration with which this Webdriver mapping is associated, the name of the user to connect to the database as, and the user's encrypted password.

Editing an Existing Webdriver Mapping

When you edit an existing Webdriver mapping, its associated Map section in the **web.cnf** file is automatically updated with the information you provide.

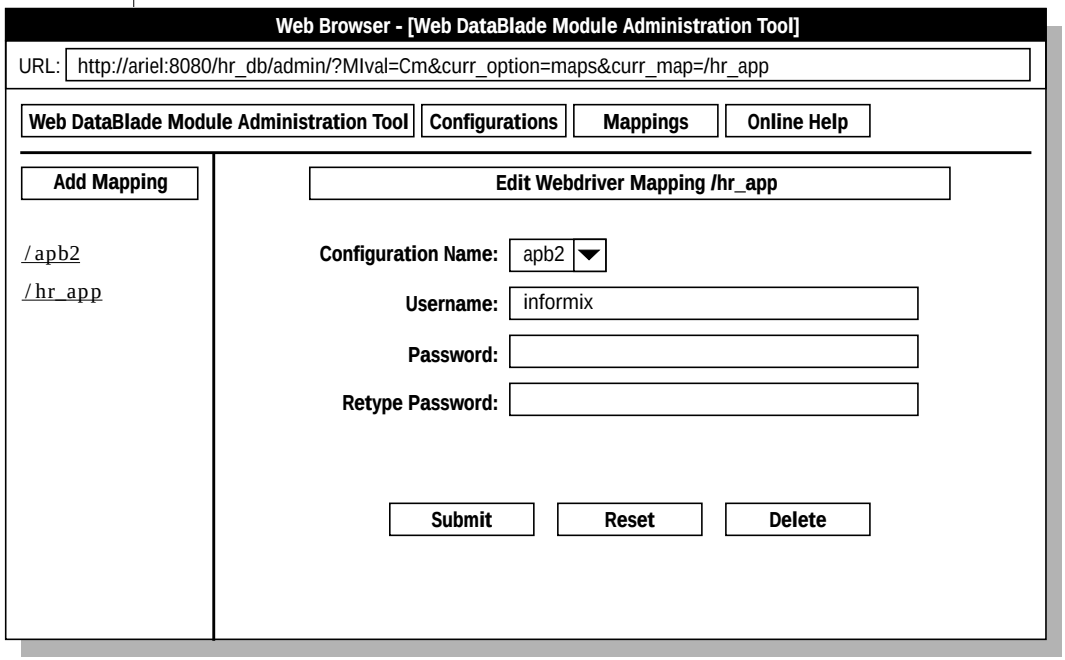
The **web.cnf** file must have correct file permissions for the Web DataBlade Module Administration Tool to be able to correctly update the file. For details on file permissions of the **web.cnf** file, refer to the section "File Permissions of the web.cnf File" on page 3-10.

To edit an existing Webdriver mapping

1. Click **Mappings** in the top frame of the Web DataBlade Module Administration Tool.
2. In the left frame, click the name of the Webdriver mapping you want to edit.

   The full details of the Webdriver mapping appear in the right frame, as shown in the following figure.

Figure 3-4
AppPage to Edit a Webdriver Mapping



3. Enter the new information for the Webdriver mapping.

   You can change the Webdriver configuration with which this Webdriver mapping is associated and the name and password of the user to connect to the database as.

   When you edit a Webdriver mapping, the **Password** text box always appears blank, even if a password exists for the Webdriver mapping. If you do not want to change the password, leave the **Password** text box blank.

4. Click **Submit**.

   The Web DataBlade Module Administration Tool automatically updates the Map section in the **web.cnf** file with the new information.

If you want to reset all the values in the AppPage back to their original settings, click **Reset** instead of **Submit**.

Adding a New Webdriver Mapping

When you use the Web DataBlade Module Administration Tool to add a new Webdriver mapping, a new Map section is automatically added to the **web.cnf** file with the information you provide.

The **web.cnf** file must have correct file permissions for the Web DataBlade Module Administration Tool to be able to correctly update the file. Refer to the section "File Permissions of the web.cnf File" on page 3-10 for details on file permissions of the **web.cnf** file.

To add a new Webdriver mapping

1. Create a new Webdriver configuration if you do not want to use an existing Webdriver configuration.

   For detailed information on this step, refer to "Adding a New Webdriver Configuration" on page 3-39.

2. Create the Webdriver mapping with the Web DataBlade Module Administration Tool.

   For detailed information on this step, refer to the next section.

3. Add a new URL prefix to your Web server.

   For detailed information on this step, refer to "Adding a URL Prefix to Your Web Server" on page 3-47.

Creating the Webdriver Mapping

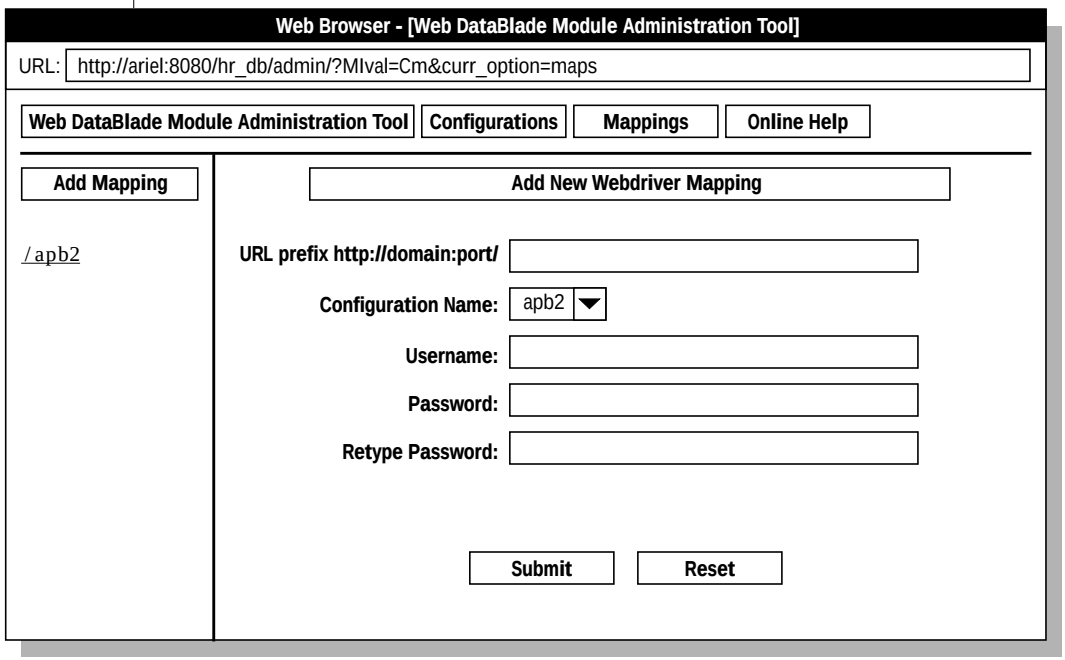
This section describes the second step in adding a new Webdriver mapping: creating the Webdriver mapping with the Web DataBlade Module Administration Tool.

To create a new Webdriver mapping

1. Click **Mappings** in the top frame of the Web DataBlade Module Administration Tool.
2. Click **Add Mapping** in the left frame.

   The **Add New Webdriver Mapping** AppPage appears in the right frame, as shown in the following figure.

Figure 3-5
AppPage to Add a Webdriver Mapping



3. Enter the name of the new Webdriver mapping in the text box labelled **URL prefix http://domain:port/**.

   The Web DataBlade Module Administration Tool automatically prefixes the name of the mapping with the necessary slashes. This means that if you want to create a Webdriver mapping called `/mapname`, enter `mapname` in the text box, without the first slash.

   It is recommended that you pick a descriptive name for the new Webdriver mapping. For example, if you are creating a new Webdriver mapping that will be used for a Human Resources application, a good name for the new Webdriver mapping is `/hr_app`.

4. Choose the Webdriver configuration you want to associate with this Webdriver mapping.

5. Enter the name and password of the user you want to connect to the database as.

   When the Web DataBlade Module Administration Tool adds the new Webdriver mapping to the **web.cnf** file, it automatically creates an encrypted password.

6. Click **Submit** to create the new Webdriver mapping.

   The Web DataBlade Module Administration Tool writes all the information as a Map section in the **web.cnf** file.

If you want to reset all the values in the AppPage back to their original settings, click **Reset** instead of **Submit**.

Adding a URL Prefix to Your Web Server

After you have created a new Webdriver mapping, you must also create a URL prefix on the Web server that matches its name exactly. For example, if you created a Webdriver mapping called `/hr_map`, the URL prefix you create for your Web server must be `/hr_map`.

Typically, you use the administration server for your particular Web server to add new URL prefixes. Refer to your Web server documentation for information on how to add URL prefixes.

The following chapters, describing the NSAPI, Apache, ISAPI, and CGI Webdrivers, also provide information on adding URL prefixes to the corresponding Web server:

- Chapter 4, "Using the NSAPI Webdriver"
- Chapter 5, "Using the Apache Webdriver"
- Chapter 6, "Using the ISAPI Webdriver"
- Chapter 7, "Using the CGI Webdriver"

Deleting an Existing Webdriver Mapping

When you use the Web DataBlade Module Administration Tool to delete a Webdriver mapping, the tool deletes its associated Map section in the **web.cnf** file.

The **web.cnf** file must have correct file permissions for the Web DataBlade Module Administration Tool to be able to correctly update the file. Refer to the section "File Permissions of the web.cnf File" on page 3-10 for details on file permissions of the **web.cnf** file.

Important: *You cannot use the Web DataBlade Module Administration Tool to delete the Webdriver mapping you are currently using to access the tool.*

To delete an existing Webdriver mapping

1. Display the details of the mapping by first clicking **Mappings** in the top frame of the Web DataBlade Module Administration Tool
2. In the left frame, click the name of the Webdriver mapping you want to delete.

   The full details of the Webdriver mapping appear in the right frame.
3. Click **Delete** to delete the Webdriver mapping.

Chapter

4

Using the NSAPI Webdriver

In This Chapter

This chapter describes how to configure and use the NSAPI Webdriver. It includes the following topics:

Overview of the NSAPI Webdriver

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

- Eliminate CGI process overhead.
- Call NSAPI functions directly in a AppPage.

  This feature is described in "Invoking NSAPI Functions in an AppPage" on page 4-17.
- Use server-side includes in AppPages.

  With this feature, AppPage developers can add sections to their AppPages that the Netscape Web server looks for and executes after the AppPage is exploded.

  This feature is described in "Using Server-Side Includes in AppPages with the NSAPI Webdriver" on page 4-18.
- Use security features built into the Netscape Web server to control access to AppPages.

  This feature is described in "Implementing User Authentication with the NSAPI Webdriver" on page 4-20.

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 sections. If you have enabled user authentication in the Netscape Web server, a different NSAPI Webdriver shared object is loaded so that the database connection is also used for user validation and security.

The NSAPI Webdriver is compatible with specific Netscape Web server versions. Check the Web DataBlade module release notes for a list of Netscape Web server versions that can implement the NSAPI Webdriver.

Configuring the NSAPI Webdriver

The NSAPI Webdriver obtains configuration information about the Web application environment from the **web.cnf** file, from the Webdriver configuration information stored in the database, and from the **obj.conf** Netscape Web server configuration file.

If you used the **websetup** utility to initially configure the Web DataBlade module for your database, the utility might have automatically performed some of the steps in the following procedure, depending on the answers you provided the utility. In particular, the **websetup** utility:

- Updates the **obj.conf** Netscape Web server configuration file with the necessary information as long as the **obj.conf** file does not already contain any NSAPI Webdriver information
- Copies and updates the **web.cnf** file with the required information
- Runs the **webconfig** utility to add the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool

The following procedure, therefore, is provided in case you need to configure the NSAPI Webdriver manually.

For more information on the **websetup** utility, refer to Chapter 2, "Getting Started," and Chapter 13, "Web DataBlade Module Utilities."

To configure the NSAPI Webdriver manually

1. If you have *never* run the **websetup** utility to configure Web server and database components, go to step 2.

   If you have previously run the **websetup** utility to configure Web server and database components, but the utility did not update the Netscape Web server configuration file, go to step 4.

   If you have run the **websetup** utility to configure Web server and database components and the utility updated the Netscape Web server configuration file **obj.conf**, you do not need follow this procedure since the NSAPI Webdriver has already been configured.

2. Copy the sample Webdriver configuration file, called **web.cnf.example**, to a directory on the Web server computer and rename it **web.cnf**.

   The **web.cnf.example** file is located in the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

   Update the **web.cnf** file by setting the Informix environment variables **INFORMIXDIR** and **INFORMIXSERVER** to point to the main Informix directory and default Informix database server, respectively.

   If you already have a working **web.cnf** file, you do not need to perform this step.

3. Run the **webconfig** utility at the operating system command prompt to add the special Webdriver mapping, called `/dbname/admin`, to the **web.cnf** file that invokes the Web DataBlade Module Administration Tool.

   For detailed information on this step, refer to "Executing the webconfig Utility" on page 4-8.

4. Stop the Netscape Web server.

5. Back up the **obj.conf** file.

6. Register Webdriver with the Netscape Web server by adding an Init directive to the **obj.conf** file.

   For detailed instructions on how to perform this step, see "Adding Init Directives to the obj.conf File" on page 4-9.

7. Add a */dbname/*admin URL prefix for your database to the **obj.conf** file. This URL prefix matches the name of the special Webdriver mapping you use to invoke the Web DataBlade Module Administration Tool.

   You only need to *manually* update the **obj.conf** file with URL prefix information *once* for a particular database. You add subsequent URL prefixes for your database with the Netscape Administration server.

   For detailed instructions on how to perform this step, see "Adding URL Prefix Information to the obj.conf File" on page 4-10.

8. Add Object directives to the **obj.conf** file for the two Webdriver shared objects: the basic Webdriver shared object and the secure Webdriver shared object.

   For detailed instructions on how to perform this step, see "Adding Object Directives to the obj.conf File" on page 4-12.

9. In the Netscape Web server startup file, typically called **startup**, set the following two variables:

   - **MI_WEBCONFIG**. Set this environment variable to point to the full pathname of the **web.cnf** file.
   - **LD_LIBRARY_PATH**. Update this environment variable to point to the following Informix libraries: ***INFORMIXDIR*/lib** and ***INFORMIXDIR*/lib/esql**. *INFORMIXDIR* refers to the main Informix directory.

   For more information on the **MI_WEBCONFIG** environment variable and the **web.cnf** file, refer to Chapter 3, "Configuring Webdriver."

10. Restart the Netscape Web server.

11. Invoke the Web DataBlade Module Administration Tool in your browser by specifying a URL of the following form in your browser:

    ```
    http://domain:port/dbname/admin/
    ```

    In this URL, *domain* refers to the name of your Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of your database.

    Note that */dbname/*admin is the URL prefix you added to your Web server, as described in Step 7.

    Many Web servers require you to add the slash at the end of the URL.

For general information on invoking AppPages with the NSAPI Webdriver, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

After you have invoked the Web DataBlade Module Administration Tool in your browser, use it to add new Webdriver mappings and Webdriver configurations to invoke your own Web DataBlade module applications or existing applications, such as AppPage Builder (APB).

For more information on adding Webdriver mappings and Webdriver configurations with the Web DataBlade Module Administration Tool, refer to Chapter 3. For detailed information on invoking and using APB, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Important: *When you manually edit the* ***obj.conf*** *file for Netscape Enterprise and FastTrack servers, the Netscape Administration server detects the changes and generates the following JavaScript Alert:*

```
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 ***OK*** *and then click* ***Apply*** *to load the changes. The Netscape Administration server should now function normally.*

Executing the webconfig Utility

Use the **webconfig** utility to add a special Webdriver mapping to the **web.cnf** file to invoke the Web DataBlade Module Administration Tool for this database.

Although you can call the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool anything you want, it is recommended that you call it */dbname/*admin, where *dbname* refers to the name of the database for which you are configuring the Web DataBlade Module Administration Tool.

When you add the special Webdriver mapping to the **web.cnf** file used to invoke the Web DataBlade Module Administration Tool, you *must* specify the **admin** Webdriver configuration with the **-n** option to the **webconfig** utility.

For example, to add a special Webdriver mapping for the Web DataBlade Module Administration Tool for the **hr_db** database and the **fred** user, execute the following command:

```
webconfig -addmap -p /hr_db/admin -n admin -d hr_db -u fred
```

The **webconfig** utility asks for the password for user **fred** and a password key.

The resulting Map section in the **web.cnf** file looks like the following example:

```
<Map path=/hr_db/admin>
database          hr_db
user              fred
password          84928490340384024343243 24
password_key      akey
config_name       admin
config_security   ON
</Map>
```

For detailed information on using the **webconfig** utility, refer to Chapter 13.

Adding Init Directives to the obj.conf File

The Init directives of the **obj.conf** file register Web server application functions with the Netscape Web server and point to the location of the shared object that contains the functions.

You can register only *one* NSAPI Webdriver with a particular Netscape Web server. This is because the names of the modules registered in the Netscape Web server are always the same for any version of the Web DataBlade module (**informix_auth**, **informix_require_auth**, **informix_explode**, and **informix_init**), and you cannot register more than one module with the same name with the Netscape Web server.

To register Webdriver as an NSAPI function, you must add the following directives to the beginning of the **obj.conf** file (if you are running iPlanet Version 6, you must make this change in the file **magnus.conf** instead of **obj.conf**):

```
Init fn="load-modules" \
shlib="path_to_INFORMIXDIR/extend/web.version/netscape/drivernsapi35.so" \
 funcs="informix_auth,informix_require_auth,informix_explode,informix_init"
Init fn="informix_init"
```

Init directives in the **obj.conf** file must fit on a single line; for clarity, however, the example shows the first Init directive on three separate lines.

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.

When you enter Init directives in the **obj.conf** file, replace the italicized *path_to_INFORMIXDIR* with the contents of the **INFORMIXDIR** Informix environment variable and *version* with the version of the Web DataBlade module you are currently using.

For example, assume that the **INFORMIXDIR** environment variable is `/disk1/informix` and the version of the Web DataBlade module you are currently using is `4.13.UC1`. The Init directive entry would look like this:

```
Init fn="load-modules" \
shlib="/disk1/informix/extend/web.4.13.UC1/netscape/drivernsapi35.so" \
 funcs="informix_auth,informix_require_auth,informix_explode,informix_init"
Init fn="informix_init"
```

Init directives in the **obj.conf** file must fit on a single line; for clarity, however, the example shows the first Init directive on three separate lines.

Adding URL Prefix Information to the obj.conf File

You add a URL prefix to the **obj.conf** file by adding a NameTrans directive. The NameTrans directives in the **obj.conf** file specify the Netscape object that is called when a user specifies the URL prefix in a browser.

You only need to manually add the first NameTrans directive for a particular database to the **obj.conf** file, typically one whose URL prefix points to the special Webdriver mapping that invokes the Web DataBlade Module Administration Tool. Use the Netscape Administration server to add any subsequent NameTrans directives.

You can include many NameTrans directives in the **obj.conf** file, but each NameTrans directive points to only one Object directive.

Important: *When you subsequently use the Web DataBlade Module Administration Tool to add a new Webdriver mapping, you must also add a new URL prefix to the Netscape Web Server. Be sure the URL prefix is exactly the same as the name of the new Webdriver mapping. For detailed information on Webdriver mappings, refer to* *Chapter 3**.*

Adding the Initial admin URL Prefix

To add the initial URL prefix to invoke the Web DataBlade Module Administration Tool, add the following NameTrans directive for the */dbname/*admin URL prefix in the **default** Object directive:

```
NameTrans fn="pfx2dir" from="/dbname/admin" dir="/ifx" name="ifx-webdriver"
```

The *dbname* variable refers to the name of the database with which you are currently working.

Important: *The Netscape Web server evaluates the directives in sequential order; therefore, to ensure that the correct URL mapping is chosen, the NSAPI Webdriver NameTrans directives should precede other NameTrans directives.*

The preceding NameTrans directive indicates to the Web server that any URL that includes the */dbname/*admin URL prefix specified in the **from** parameter follows the directives for the object specified by the **name** parameter, or **ifx-webdriver**. The **name** parameter maps to the Object setting for the named object; the section "Adding Object Directives to the obj.conf File" on page 4-12 describes how to add Webdriver Object directives to the **obj.conf** file.

The following example shows a NameTrans directive for the /hr_db/admin URL prefix within the default Object directive:

```
<Object name=default>
NameTrans fn="pfx2dir" from="/hr_db/admin" dir="/ifx" name="ifx-webdriver"
</Object>
```

Adding Subsequent URL Prefixes

After you have added the NameTrans directive that specifies the */dbname/*admin URL prefix to invoke the Web DataBlade Module Administration Tool, you might later want to add a NameTrans directive that specifies, for example, the /apb URL prefix to invoke APB for your database.

Use the Netscape Administration server to add all subsequent NameTrans directives to the **obj.conf** file.

Important: *The following procedure is written with the assumption that you have added the two Object directives for the two types of Webdriver objects to the* ***obj.conf*** *file. For more information on adding Object directives, refer to "Adding Object Directives to the obj.conf File" on page 4-12.*

To use the Netscape Administration server to add subsequent NameTrans directives

1. Click your Netscape Web server name from the main Netscape Administration Web page.
2. Click **Content Management** and then click the **Additional Document Directories** link.
3. Enter the URL prefix in the **URL prefix** text box.

   For example, if you are adding a URL prefix to invoke APB for the **hr_db** database, enter the following text in the text box:

   ```
   /hr_db/apb
   ```
4. Enter `/ifx` in the **Map to Directory** text box.
5. In the **Apply Style** list box, choose **ifx-webdriver** if you want to use the basic Webdriver or **ifx-webdriver-auth** if you want to use the secure Webdriver.
6. Click OK.

   You can ignore the warning that states that the **/ifx** directory does not exist.
7. Click **Save and Apply**.

   The Netscape Web server is automatically restarted.

Adding Object Directives to the obj.conf File

The Object directives in the **obj.conf** file describe the object named in the **name** parameter of the NameTrans directives.

Add two Object directives to the **obj.conf** file for the two types of Webdriver objects: the basic Webdriver and the secure Webdriver.

The following example shows the Object directive for the basic **ifx-webdriver** object:

```
<Object name="ifx-webdriver">
Service method=(GET|POST) fn="informix_explode"
</Object>
```

The Service directive indicates the call to the **informix_explode** NSAPI function, which in turn makes the call to the Informix **WebExplode()** function.

The following example shows the Object directive for the secure **ifx-webdriver-auth** object; the directive contains more information about how user authentication is performed:

```
<Object name="ifx-webdriver-auth">
AuthTrans fn="basic-auth" auth-type="basic" userdb="ifx"
  userfn="informix_auth"
PathCheck fn="informix_require_auth" auth-type="basic" realm="Secure"
Service method=(GET|POST) fn="informix_explode"
</Object>
```

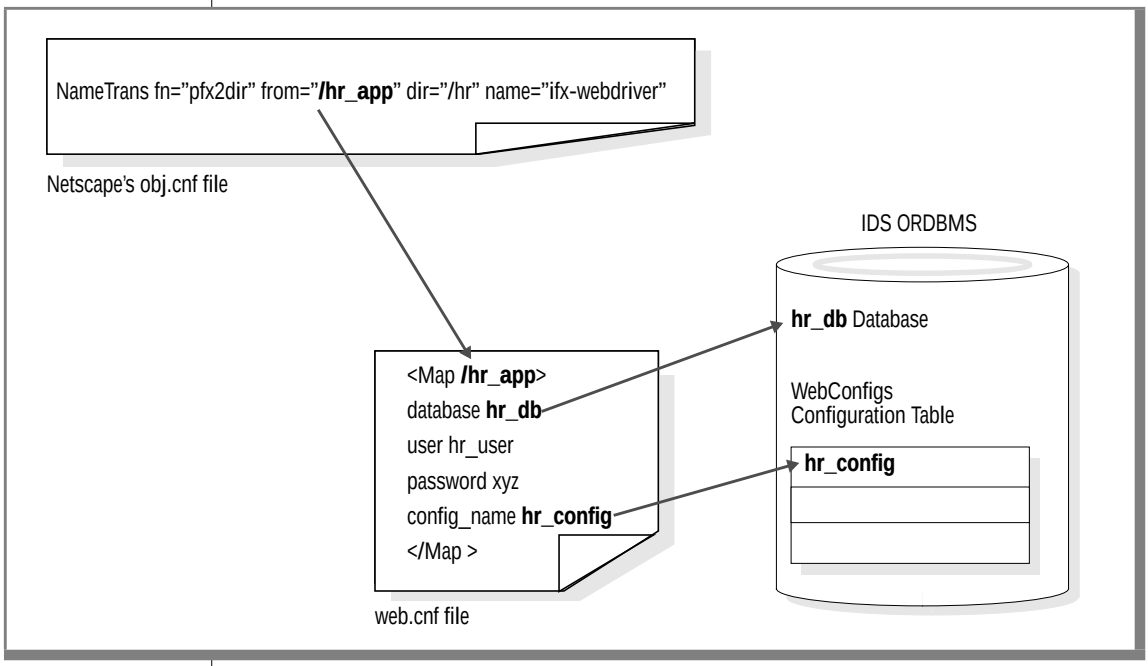
The PathCheck directive indicates that user authentication is required. The AuthTrans directive indicates that the **informix_auth** NSAPI function performs the user authentication.

For detailed information on using Netscape Web server user authentication with the NSAPI Webdriver, see "Implementing User Authentication with the NSAPI Webdriver" on page 4-20.

How It All Fits Together

The URL prefix specified in the **from** parameter of the NameTrans directive matches the name of the Webdriver mapping specified in the **web.cnf** file. This Webdriver mapping then maps to a Webdriver configuration in the database specified in the **web.cnf** file, as shown in Figure 4-1.

Figure 4-1
Mapping Between Netscape obj.conf File, web.cnf, and WebConfigs Configuration Table



Executing NSAPI Functions in AppPages

The NSAPI Webdriver is a Netscape server application function. NSAPI uses the dynamic linking functions of operating systems to enable you to load NSAPI-compliant code modules, or NSAPI functions, into the Web server at runtime. When you include these NSAPI functions as Web server extensions, you can use the NSAPI Webdriver and the MIFUNC tag to call the NSAPI functions within an AppPage.

When the **WebExplode()** function encounters an MIFUNC tag in an AppPage, it passes the function name to the NSAPI Webdriver. The NSAPI Webdriver uses the Netscape function lookup routine to find the code module. The argument list supplied to the MIFUNC call is then passed through to the function, using the **Request vars pblock** structure. After the function is executed, any value that has been modified is passed back to the **WebExplode()** function. When the code between the MIFUNC tags is processed, the new values are effectively passed by reference.

To call an NSAPI function within an AppPage

1. Create the NSAPI function.

   For detailed instructions for this step, see "Creating NSAPI Functions," following.

2. Configure the **obj.conf** Netscape configuration file to load the function.

   Refer to your Netscape Web server documentation for detailed instructions on this step.

3. Invoke the function in the MIFUNC tag within an AppPage.

   For detailed instructions for this step, see "Invoking NSAPI Functions in an AppPage" on page 4-17.

Creating NSAPI Functions

NSAPI functions are described by the following prototype:

```
int function(pblock *pb, Session *sn, Request *rq)
```

For additional information on writing NSAPI functions, see the Netscape server API documentation or visit the Netscape developer's Web site at http://developer.netscape.com.

The following example shows how to make a call to a function that executes a program (in this case, the UNIX **ls -l** command) and returns the results within the AppPage.

When you write the NSAPI function, you define variables that are passed by reference, or imported, to the function. (Use the **Request vars pblock** structure to import variables from the AppPage and to return, or export, values to the AppPage.) The following **example_dir()** function imports the **OPTIONS** variable and exports results in the **RESULTS** variable:

```
#include <stdio.h>
#include <nsapi.h>
#define MAXRESULTS 16384

example_dir(pblock *param, Session *sn, Request *rq)
{
char *result;
char cmd[128];
char *options;
char fname[60];
int bcount;
int fd;

    /* extract the options */

    options = pblock_findval("OPTIONS",rq->vars);

    sprintf(fname, "/tmp/dir.%d-%d", getpid(), systhread_current());
    sprintf(cmd, "/bin/ls %s >%s", options, fname);
    system(cmd);

    /* allocate some memory */

    result = malloc(MAXRESULTS);

    /* read results */

    fd = open(fname,0);
    bcount = read(fd, result, MAXRESULTS);
    close(fd);

    sprintf(cmd, "rm %s", fname);
    system(cmd);
```

```
        /* null terminate */
        *(result + bcount - 1) = '\0';

        pblock_remove("RESULTS",rq->vars);
        pblock_nvinsert("RESULTS",result,rq->vars);
        free(result);
    }
```

Use the standard NSAPI return codes in your functions. For example, return the REQ_PROCEED code for successful completion of the function and return the REQ_ABORTED code for failure.

Invoking NSAPI Functions in an AppPage

Within the MIFUNC tag, you must include the variables to be imported and exported (passed by reference) as name/value pairs. You must also specify the FUNCTION attribute to locate the NSAPI function in the Web server. Assign the value of the FUNCTION attribute to the name of the NSAPI function.

When the MIFUNC tag is executed, all AppPage processing stops until the function has completed execution. After the NSAPI function is executed, AppPage processing continues, and everything between the MIFUNC tags is executed using the variables that have been modified by reference in the NSAPI function, in addition to the variables originally supplied to the AppPage.

The following AppPage segment invokes the **example_dir()** function.

```
<?MIBLOCK COND=$(NXST,$opt)><?MIVAR>$(SETVAR,$opt,"-l
  /tmp")<?/MIVAR><?/MIBLOCK>
<?MIFUNC FUNCTION=example_dir OPTIONS=$opt RESULTS="">
<?MIVAR>$(HTTPHEADER,Content-type,text/plain)$RESULTS<?/MIVAR>
<?/MIFUNC>
```

You can nest MIFUNC tags so that one function can execute depending on the results of another. You can also include any number of MIFUNC tags in an AppPage; however, you should pay attention to the ordering of the tags to achieve the desired result.

For more information on using the MIFUNC Web DataBlade module tag in AppPages, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide.*

Using Server-Side Includes in AppPages with the NSAPI Webdriver

Server-side includes (also called *parsed html*) are special commands embedded in an HTML page that are recognized and interpreted by the Web server; the output of the commands is placed in the HTML page before the AppPage is sent to the browser. Server-side includes can be used, for example, to include a date or time stamp in the text of the HTML page.

The following procedure describes the steps you must take for the NSAPI Webdriver to correctly handle AppPages that contains server-side includes.

To use server-side includes in an AppPage

1. In your AppPage, use the **PARSE-HTML** variable-processing function together with the MIVAR tag to indicate that your AppPage contains server-side includes.

   The **PARSE-HTML** variable-processing function has two options: SHARED and DYNAMIC.

   The SHARED option specifies that Webdriver send the cached AppPage to the Web server. This option presumes that you have enabled AppPage caching for the AppPage. The following example shows how to specify the SHARED option:

   ```
   <?MIVAR>$(PARSE-HTML,SHARED)<?/MIVAR>
   Webdriver sends the cached AppPage to the Web server.
   ```

   The DYNAMIC option specifies that Webdriver always send the AppPage to the **WebExplode()** function for dynamic processing and then temporarily store the AppPage in a directory that will subsequently be read by the Web server. The following example shows how to specify the DYNAMIC option:

   ```
   <?MIVAR>$(PARSE-HTML,DYNAMIC)<?/MIVAR>
   Webdriver sends the AppPage to the WebExplode function
   and then temporarily stores it in a directory for the
   Web server to read.
   ```

   For more information on the **PARSE-HTML** variable processing function, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

2. If you specified the `SHARED` option in step 1, enable AppPage caching for the AppPage.

   For detailed information on enabling AppPage caching, refer to Chapter 9, "Improving Performance."

3. If you specified the `DYNAMIC` option in step 1, use the Web DataBlade Module Administration Tool to set the **parse_html_directory** Webdriver variable in your Webdriver configuration.

   The following table describes this Webdriver variable.

Webdriver Variable	Mandatory?	Description
parse_html_directory	Yes	Specifies the full pathname of the directory on the Web server computer where Webdriver temporarily stores the AppPage to be subsequently read by the Web server Webdriver does not create this directory, so be sure the directory exists before you use server-side includes in an AppPage.

For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

Implementing User Authentication with the NSAPI Webdriver

You can use the *user authentication* feature of the Netscape Web server to control user access to AppPages. User authentication refers to the process of users verifying their identity by entering a user name and password when they access secure AppPages.

To implement user authentication with NSAPI Webdriver

1. Set Webdriver variables in your Webdriver configuration to enable user authentication for the NSAPI Webdriver.

   For detailed information on this step, refer to "Setting Webdriver Variables to Enable User Authentication" on page 4-20.

2. Update the Netscape Web server configuration file, **obj.conf**, to enable user authentication for the Netscape Web server.

   For detailed information on this step, refer to "Updating the obj.conf File to Enable User Authentication" on page 4-23.

3. Add users to authenticate against to the appropriate database table.

   For detailed information on this step, refer to "Adding Users to the MIusertable Table" on page 4-24.

4. Set the access level for each AppPage for which you want to control access.

   For detailed information on this step, refer to "Specifying AppPage Access Levels" on page 4-25.

Setting Webdriver Variables to Enable User Authentication

This section describes the first step of the four-part process described in "Implementing User Authentication with the NSAPI Webdriver" on page 4-20: setting the Webdriver variables necessary to enable user authentication with the NSAPI Webdriver.

Use the Web DataBlade Module Administration Tool to set the Webdriver variables described in the following table.

Variable Name	Mandatory?	Content
MIusertable	Yes	Name of the table that contains user access information
MIusername	Yes	Name of the VARCHAR column in the user access table (**MIusertable**) that contains the name of the database user
MIuserpasswd	Yes	Name of the VARCHAR column of the user access table (**MIusertable**) that contains the password of the database user
MIuserlevel	Yes	Name of the INTEGER column of the user access table (**MIusertable**) that contains the access level of the database user
MIpagelevel	Yes	Name of the INTEGER column of the table that stores your AppPage that contains the access level of the AppPage
MIusergroup	No	Name of the INTEGER column of the user access table (**MIusertable**) that contains the group access level of the user
redirect_url	No	URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
auth_crypt_udr	No	Enables password encryption when set to `ON` If password encyrption is enabled, Webdriver encrypts the password entered by the user and compares it to the encrypted password in the **MIusertable** table. If they match, then the user is authenticated. If set to `OFF` (default value), then Webdriver does not encrypt the password. Refer to "Using Encrypted Passwords in the MIusertable Table" on page 4-26 for detailed information on using encrypted passwords.

For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

Important: *If* ***MIpagelevel*** *has not been set for your Webdriver configuration, no user authentication check is performed.*

How the NSAPI Webdriver Uses the Webdriver Variables

If the access level of the retrieved AppPage is less than or equal to the access level of the authenticated user, Webdriver allows the user to access the AppPage. Webdriver stores the access level of the user (obtained from the value of the **MIuserlevel** column of the **MIusertable** table) 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.

When user authentication is not available or access is denied, you can redirect the browser to another URL by setting the **redirect_url** variable to that URL. If **redirect_url** is not set and a user attempts to access an AppPage with an access level higher than 0, an access error is raised. You can access the error by creating an error page from which you can query the **MI_DRIVER_ERROR** variable. For detailed information on using the **MI_DRIVER_ERROR** variable, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Example of Setting User Authentication Webdriver Variables

The following table shows sample values for the Webdriver variables that enable NSAPI security. User access settings are based on the **wbUsers** table, described in the schema for APB in the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Webdriver Variable	Sample Value
MIpagelevel	`read_level`
MIusertable	`wbUsers`
MIusername	`name`

(1 of 2)

Webdriver Variable	Sample Value
MIuserpasswd	`password`
MIuserlevel	`security_level`
redirect_url	`http://domain/hp_app/errors`

(2 of 2)

Updating the obj.conf File to Enable User Authentication

This section describes the second step of the four-part process described in "Implementing User Authentication with the NSAPI Webdriver" on page 4-20: updating the **obj.conf** file.

You add user authentication information to the **obj.conf** file by adding a new Object directive that specifies user authentication information. You specify user authentication information by including AuthTrans and PathCheck directives in this new Objective directive.

The following example shows an Objective directive in the **obj.conf** file that specifies user authentication information:

```
<Object name="ifx-webdriver-auth">
AuthTrans userfn="informix_auth" userdb="ifx" fn="basic-auth"
  auth-type="basic"
PathCheck realm="Secure" fn="informix_require_auth" auth-type="basic"
Service method="(GET|POST)" fn="informix_explode"
</Object>
```

The name of this Object directive is **ifx-webdriver-auth**, and the directive includes AuthTrans and PathCheck directives that invoke the secure Webdriver object.

You add the following NameTrans directive to the **obj.conf** file to use the secure Webdriver object:

```
NameTrans from="/secure" fn="pfx2dir" dir="/secure"
  name="ifx_webdriver-auth"
```

In the example, `/secure` is the URL prefix.

The following example shows standard NameTrans and Object directive entries in the **obj.conf** file for nonsecure users:

```
<Object name="ifx-webdriver">
Service method="(GET|POST)" fn="informix_explode"
</Object>

NameTrans from="/guest" fn="pfx2dir" dir="/guest" name="ifx-webdriver"
```

In the example, `/guest` is the URL prefix.

For a description of the preceding **obj.conf** file entries, see "Adding URL Prefix Information to the obj.conf File" on page 4-10.

An AppPage can only be accessed by authorized users with an 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 AppPage table.

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

```
http://mydomain:port/guest?MIval=/pagename.html
```

An example of a secure URL is:

```
http://mydomain:port/secure?MIval=/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.

Adding Users to the MIusertable Table

This section describes the third step of the four-part process described in "Implementing User Authentication with the NSAPI Webdriver" on page 4-20: adding users to the **MIusertable** table.

Webdriver authenticates users who request a secure AppPage against the list of users stored in the table specified by the **MIusertable** Webdriver variable. This table contains user access information, such as the name of the user, their password, their access level, and so on. When a user requests a secure AppPage, Webdriver checks their inputted password against the password in the **MIusertable** table, looks up the user's access level, checks it against the access level needed to view the AppPage, and then decides whether the user is allowed to view the AppPage.

You add users to the **MIusertable** table with an INSERT statement with the DB-Access or SQL Editor tools.

In the APB schema, the table to store user access information is called **wbUsers**. The following example shows how to insert a new user into this table:

```
INSERT INTO wbUsers VALUES
('fred' , 'fred_password', 99, 'APB 2.0', 'AppPage', 80, 20, 't', '0');
```

In the example, user **fred** has an access level of 99. This means that Webdriver allows user **fred** to view AppPages whose access level is 99 or less.

When you install APB into your database, two users are automatically inserted into the **wbUsers** table: **default** and **admin**.

For detailed information on encrypting the user password, refer to "Using Encrypted Passwords in the MIusertable Table" on page 4-26.

For a detailed description of the columns of the **wbUsers** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Specifying AppPage Access Levels

This section describes the fourth step of the four-part process described in "Implementing User Authentication with the NSAPI Webdriver" on page 4-20: specifying AppPage access levels.

You specify the access level of an AppPage by updating the access level column in the table that stores the AppPage. The name of the column that contains access level information for each AppPage is specified by the **MIpagelevel** Webdriver variable.

For example, in the APB schema, the table that stores AppPages is called **wbPages**. The **wbPages** table contains a column called **read_level** that specifies the minimum access level a user must have to be able to view the corresponding AppPage. Therefore, for the APB schema, the **MIpagelevel** Webdriver variable is set to `read_level`.

If you want to specify a high access level for a particular AppPage, then update the **read_level** column in the **wbPages** table for that AppPage to the appropriate integer.

For a detailed description of the columns of the **wbPages** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide.*

Using Encrypted Passwords in the MIusertable Table

For security reasons, you might want to encrypt all the passwords stored in the **MIusertable** table. This section describes:

- How to encrypt the passwords in the **MIusertable** table
- How to set the **auth_crypt_udr** Webdriver variable to ensure that Webdriver recognizes encrypted passwords
- How to update APB so it uses encrypted passwords
- Tips on creating your own AppPage to insert and edit users in the **MIusertable** table

Encrypting Passwords

If you set the **auth_crypt_udr** Webdriver variable to ON, Webdriver encrypts the password of the user being authenticated and compares it to the password in the **MIusertable** table. If they match, the user is authenticated.

This means that if you set **auth_crypt_udr** to ON, you must be sure all the passwords stored in the **MIusertable** table are encrypted as well. Use the **webpwcrypt()** routine to encrypt passwords in the **MIusertable** table.

For example, if you are using the APB schema and you already have users in the **wbUsers** table whose passwords are not encrypted, then the following UPDATE statement encrypts the existing passwords:

```
UPDATE wbUsers
SET password = webpwcrypt(password, '');
```

To encrypt passwords as you insert new users into the **wbUsers** table, use an INSERT statement similar to the following example:

```
INSERT INTO wbUsers VALUES
('fred2' , webpwcrypt('fred2_password',''), 99, 'APB 2.0', 'AppPage', 80,
  20, 't', '0');
```

The preceding INSERT statement is very similar to the INSERT statement in "Adding Users to the MIusertable Table" on page 4-24 except that it executes the **webpwcrypt()** routine on the password column.

For a detailed description of the columns of the **wbUsers** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Setting the auth_crypt_udr Webdriver Variable

Once you have encrypted all the passwords in the **MIusertable** table, you must ensure that Webdriver always encrypts the password entered by a user *before* Webdriver checks to see if the passwords match. By default, Webdriver does not encrypt incoming passwords.

To specify that Webdriver encrypt incoming passwords before checking them against the passwords stored in the **MIusertable** table, set the **auth_crypt_udr** Webdriver variable for your Webdriver configuration to `ON`.

Tips for Creating Your Own AppPage to Edit User Password Information

If you create your own AppPage to insert or update a user in the **MIusertable**, and you use encrypted passwords, the INSERT or UPDATE statement contains the clear text password as part of the **webpwcrypt()** routine. If you have enabled tracing with the **debug_level** Webdriver variable, then this INSERT or UPDATE statement might be written to the log file, with the clear text password clearly visible.

To ensure that the clear text password is *not* written to the log file, set the **MIdriver** Webdriver variable to `debug_off` to turn off logging of the UPDATE or INSERT statements to the log file. You set the **MIdriver** Webdriver variable as a hidden variable in an INPUT tag after you create a FORM with the attribute REQUEST=POST.

The following section of the **Edit User** AppPage of APB shows an example of how to use the **MIdriver** Webdriver variable to turn off logging of a single INSERT or UPDATE statement:

```
<FORM METHOD=POST ACTION="<?MIVAR>$WEB_HOME<?/MIVAR>">
<INPUT TYPE=hidden NAME=MIval
VALUE="/APB20/apb_edit_User.html">
<INPUT TYPE=hidden NAME=MIdriver VALUE="debug_off">
```

Using the REMOTE_USER Web Browser Variable for User Authentication

An additional user authentication method is available for the NSAPI Webdriver.

If the **connect_as_user** Webdriver variable is set to ON in your Webdriver configuration, all database requests connect as the user specified by the **REMOTE_USER** Web browser variable instead of the user defined in the **web.cnf** file for your Webdriver mapping. The user specified by the **REMOTE_USER** Web browser variable must be added to the user access table identified by the **MIusertable** variable (typically the **wbUsers** table) to enable **connect_as_user** user authentication. The password in the user access table is ignored since user authentication is performed when connecting to the database.

Important: *If you set* ***connect_as_user*** *to* ON*, the user specified by the* ***REMOTE_USER*** *Web browser variable must be a valid operating system user with database connection privileges. Therefore, this authentication method should be restricted to Intranet, rather than Internet, applications.*

Additional NSAPI Webdriver Information

This section describes information specific to the NSAPI Webdriver that you should be aware of when you use the Web DataBlade module.

WebExplode() Buffer Size with NSAPI Webdriver

The **max_html_size** Webdriver variable specifies the maximum size of the buffers returned from the **WebExplode()** function. Setting this variable allows dynamic memory allocation without causing Web server memory growth. The default buffer size is 128 KB.

For the NSAPI Webdriver, each thread initially allocates 8 KB of memory and then grows up to the value of the **max_html_size** Webdriver variable.

Passing Image Map Coordinates with the NSAPI Webdriver

For details on passing image map coordinates with Webdriver in standard image maps and forms, see the *IBM Informix Web DataBlade Module Application Developer's Guide*.

An example image map URL is:

```
http://mydomain:port/hr_app/MImap=on&MIval=image_example?100,13
```

Important: *You must specify* ***MImap*** *as the first variable in* ***PATH_INFO****. The NSAPI Webdriver ignores anything preceding* ***MImap****.*

Administering the NSAPI Webdriver

This section discusses NSAPI Webdriver performance and error logging.

NSAPI Webdriver Performance

The NSAPI Webdriver shared object, **drivernsapi35.so**, is loaded by the Netscape Web server. The NSAPI Webdriver uses the same threading system used by Netscape to allow multiple simultaneous connections to a database.

As requests are made to the Web server, connections are opened to the database. These connections remain open and are reused as new threads request database access. Specify the maximum number of connections to the database with the **dbconnmax** Webdriver variable in Global section of the **web.cnf** file. The default number of connections is 16.

To monitor the database connection pool

1. Turn on the debug flag for dumping connection pool information by setting the **debug_level** Webdriver variable for your Webdriver configuration using the Web DataBlade Module Administration Tool.

   Set the variable to either `0x0400` or `1024`.

2. Set a trace file by setting the **debug_log** Webdriver variable in the Global section of the **web.cnf** file, as described in Chapter 12, "Debugging and Troubleshooting."

The following example shows sample output to the **debug_log** file:

```
CP[014714e8]    SID  S  Rqs  Max Database  Server  User     QT  QD  KA  WT
CP[014714e8]000 703  QR  14  100 webtest   bldsvr  webuser   0   0   0   0
CP[00000000]001 704  I   10  100 webtest   bldsvr  webuser   0   0   0   0
CP[00000000]002 706  I    9  100 webtest   bldsvr  webuser   0   0   0   0
CP[00000000]003 705  I   10  100 webtest   bldsvr  webuser   0   0   0   0
CP[00000000]004 707  I    8  100 webtest   bldsvr  webuser   0   0   0   0
```

In the preceding output, the number in the first column enclosed in brackets ([ ]) is the thread identifier.

The following table describes the remaining columns of the output to the **debug_log** file.

Column Name	Description
SID	Session ID in the database
S	Refers to the status of the thread The status can be one of the following values: ■ QR (query request) ■ LR (large object request) ■ I (idle) ■ - (closed) ■ +R (connection opening)
Rqs	Number of requests on this connection so far
Idle	Number of seconds the Webdriver connection has been idle (Only displayed when **dbconntimeout** is set in the **web.cnf** file; **dbconntimeout** is described in the *IBM Informix Web DataBlade Module Application Developer's Guide*).
Max	Maximum number of requests for the connection (**connection_life** Webdriver variable)
Database	Name of the database being accessed
Server	Name of the database server being used to access the database
User	Name of the user accessing database
QT	Query time-out interval (**query_timeout** Webdriver variable)

(1 of 2)

Column Name	Description
QD	Query duration (how long the query has been running, in KA amounts)
KA	Timer to check if the connection is still alive (**keepalive** Webdriver variable)
WT	Number of times that the last thread that succeeded in getting a connection had to go into a wait cycle

(2 of 2)

The **query_timeout** and **keepalive** Webdriver variables are described in "Managing Webdriver Connections to the Database" on page 3-19. For detailed information on updating Webdriver variables with the Web DataBlade Module Administration Tool, refer to Chapter 3.

Connections are shut down and reestablished after **connection_life** number of requests. The default value for **connection_life** is 100 requests; this value should be modified only under the guidance of Technical Support.

If you reach the maximum number of connections (**dbconnmax** variable in the Global section of the **web.cnf** file) and all threads are processing queries, you will notice a WAITING message by the headers (to the right of the **WT** column).

Error Logging with 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 directive in the **magnus.conf** file.

Chapter

5

Using the Apache Webdriver

In This Chapter

This chapter describes how to configure and use the Apache Webdriver. It includes the following topics:

Overview of the Apache Webdriver

The Apache Webdriver uses the Apache API rather than the CGI interface to connect to the database and execute AppPages.

Usually, when you configure the Apache Webdriver, you rebuild the Apache Web server **httpd** binary, to link in the Apache Webdriver object file. This process is described in detail in "Configuring the Apache Webdriver" on page 5-4. If you do not have the Apache source code or if you want to run the default **httpd** binary that was installed at the time you installed the Apache Web server, refer to the section "Dynamically Loading the Apache Webdriver" on page 5-26.

The Apache Webdriver offers the ability to:

- Eliminate CGI process overhead.
- Use security features built into the Apache Web server to control access to AppPages.

  This feature is described in "Implementing User Authentication with Apache Webdriver" on page 5-15.
- Use server-side includes in AppPages. With this feature, AppPage developers can add sections to their AppPages that the Apache Web server looks for and executes after the AppPage is exploded.

  This feature is described in "Using Server-Side Includes in AppPages with the Apache Webdriver" on page 5-24.

This chapter is written with the assumption that you have already installed the Web DataBlade module on your computer, created a database with logging enabled, registered the Web DataBlade module in the database, and installed the Web DataBlade Module Administration Tool in the database.

The Apache Webdriver is compatible with specific Apache Web server versions. Check the Web DataBlade module release notes for a list of Apache Web server versions that can implement the Apache Webdriver.

Configuring the Apache Webdriver

The Apache Webdriver obtains configuration information about the Web application environment from the **web.cnf** file, from the Webdriver configuration information stored in the database, and from the Apache Web server configuration files.

The following procedure describes the basic steps you must perform to configure the Apache Webdriver. Some of the steps are described in more detail later in this section.

If you used the **websetup** utility to initially configure the Web DataBlade module for your database, the **websetup** utility might have automatically performed some of the steps in the following procedure. In particular, the **websetup** utility:

- Copies and updates the **web.cnf** file with the required information
- Runs the **webconfig** utility to add the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool

For more information on the **websetup** utility, refer to Chapter 2, "Getting Started," and Chapter 13, "Web DataBlade Module Utilities."

Important: *The following procedure assumes that you have previously installed and configured the Apache Web server and that it is up and running. The procedure also assumes that the configuration options you specified when you ran the Apache* ***configure*** *utility are stored in the file* ***$APACHEDIR/src/apaci****. The Web DataBlade module uses the options stored in this file when you rebuild the* ***httpd*** *binary.*

To configure the Apache Webdriver

1. If you have *never* run the **websetup** utility to configure Web server and database components, go to step 2.

   If you have previously run the **websetup** utility to configure Web server and database components, go to step 4.

2. Copy the sample Webdriver configuration file, called **web.cnf.example**, to a directory on the Web server computer and rename it **web.cnf**.

   The **web.cnf.example** file is located in the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

   Update the **web.cnf** file by setting the Informix environment variables **INFORMIXDIR** and **INFORMIXSERVER** to point to the main Informix directory and default Informix database server, respectively.

   If you already have a working **web.cnf** file, you do not need to perform this step.

3. Run the **webconfig** utility at the operating system command prompt to add the special Webdriver mapping to the **web.cnf** file used to invoke the Web DataBlade Module Administration Tool.

   For detailed information on this step, refer to "Executing the webconfig Utility" on page 5-8.

4. Stop the Apache Web server.

5. Register the Apache Webdriver with the Apache Web server by editing the Apache Web server configuration file called **Configuration**, located in the **src** directory under the main Apache Web server directory.

   For detailed information on this step, refer to "Editing the Apache Web Server Configuration File" on page 5-9.

6. Configure the Apache Web server for your operating system by running the **Configure** program located in the **src** directory under the main Apache Web server directory.

   For detailed information on this step, refer to the Apache Web site at http://www.apache.org.

7. If you want AppPage developers to use server-side includes in their AppPages, edit one of the Apache Web server source code files in the **src** directory under the main Apache Web server directory.

   For detailed information on editing Apache Web server source code, refer to "Editing Apache Web Server Source Code" on page 5-11.

8. Set the **INFORMIXDIR** environment variable in your environment to point to the main Informix directory.

9. Type `make` in the main Apache directory to create the new Apache Web server binary, **httpd**.

   For detailed information on this step, refer to the Apache Web site at http://www.apache.org.

10. Type `make install` in the main Apache directory to update your Apache installation.

    For detailed information on this step, refer to the Apache Web site at http://www.apache.org.

11. Add mapping information to the Apache Web server configuration file **httpd.conf** so that the Web server calls the Apache Webdriver when you specify a URL prefix that maps to a Webdriver mapping. At the very least, you must add a mapping that invokes the Web DataBlade Module Administration Tool.

    For detailed information on this step, refer to "Adding URL Prefix Information to the Apache Web Server" on page 5-13.

12. Set the **MI_WEBCONFIG** and **LD_LIBRARY_PATH** environment variables to point to the full pathname of the **web.cnf** file and the Informix libraries, respectively.

    Set these environment variables either in the Apache Web server startup file or, if you start up the Web server directly at the operating system prompt, in the environment of the user who starts up the Web server.

    For more information on the **MI_WEBCONFIG** environment variable and the **web.cnf** file, refer to Chapter 3, "Configuring Webdriver."

13. Restart the Apache Web server.

14. Invoke the Web DataBlade Module Administration Tool in your browser by specifying a URL of the following form in your browser:

    ```
    http://domain:port/dbname/admin/
    ```

    In this URL, *domain* refers to the name of your Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of your database.

    As described in Step 11, */dbname/*`admin` is the URL prefix you added to your Web server.

    Many Web servers require you add the slash at the end of the URL.

For general information on invoking AppPages with the Apache Webdriver, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

After you have invoked the Web DataBlade Module Administration Tool in your browser, use it to add new Webdriver mappings and Webdriver configurations to invoke your own Web DataBlade module applications or to invoke existing applications, such as AppPage Builder (APB).

For more information on adding Webdriver mappings and Webdriver configurations with the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver." For detailed information on invoking and using APB, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Executing the webconfig Utility

Use the **webconfig** utility to add a special Webdriver mapping to the **web.cnf** file to invoke the Web DataBlade Module Administration Tool for the database for which you are configuring the Web DataBlade module.

Although you can name the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool anything you want, Informix recommends you name it */dbname*/admin, where *dbname* is the name of the database for which you are configuring the Web DataBlade Module Administration Tool.

When you add the special Webdriver mapping to the **web.cnf** file used to invoke the Web DataBlade Module Administration Tool, you *must* specify the **admin** Webdriver configuration with the **-n** option to the **webconfig** utility.

For example, to add a special Webdriver mapping for the Web DataBlade Module Administration Tool for the **hr_db** database and the **fred** user, execute the following command:

```
webconfig -addmap -p /hr_db/admin -n admin -d hr_db -u fred
```

The **webconfig** utility asks for the password for user **fred** and a password key.

The resulting Map section in the **web.cnf** file looks like the following example:

```
<Map path=/hr_db/admin>
database        hr_db
user            fred
password        84928490340384024343 24324
password_key    akey
config_name     admin
</Map>
```

For detailed information on using the **webconfig** utility, refer to Chapter 13, "Web DataBlade Module Utilities."

Editing the Apache Web Server Configuration File

The **Configuration** file, located in the **src** directory under the main Apache Web server directory, specifies the modules that are compiled into the Apache Web server binary and the flags that are used when `make` is executed to create the binary. Edit this file so that the Apache Webdriver module is compiled into the binary.

To edit the Configuration file to add the Apache Webdriver module

1. Add the following Informix directories to the EXTRA_LDFLAGS option of the **Configuration** file:

   - -L*INFORMIXDIR* `/lib`
   - -L*INFORMIXDIR* `/lib/esql`
   - -L*INFORMIXDIR* `/extend/web.`*version*`/apache`

   Separate each entry with a blank space.

   In the preceding list, *INFORMIXDIR* refers to the full pathname of the main Informix directory, and *version* refers to the current version of the Web DataBlade module installed on your computer.

   For example, assume that *INFORMIXDIR* is **/disk/ifmx** and the current version of the Web DataBlade module is 4.13.UC1. The following example shows the entry for the EXTRA_LDFLAGS option in the **Configuration** file:

   ```
   EXTRA_LDFLAGS=-L/disk/ifmx/lib -L/disk/ifmx/lib/esql
   -L/disk/ifmx/extend/web.4.13.UC1/apache
   ```

   The preceding entry should not contain any carriage returns: it should fit on exactly one line.

2. Add the following directory to the EXTRA_CFLAGS option:

   ```
   `APACHEDIR/src/apaci`
   ```

   APACHEDIR refers to the main Apache Web server directory. Be sure you enclose the option in back quotes (`).

   For example, assume that *APACHEDIR* is **/local/apache**. The following example shows the entry for the EXTRA_CFLAGS option in the **Configuration** file:

   ```
   EXTRA_CFLAGS=`/local/apache/src/apaci`
   ```

3. Add the following libraries to the EXTRA_LIBS option:

```
-lifxw -lifsql -lifasf -lifgen -lifos -lifgls
INFORMIXDIR/lib/esql/checkapi.o -lifglx -lifsql -lifasf
-lifgen -lifos -lifgls -lm
```

   Separate each entry with a blank space.

   In the preceding list, *INFORMIXDIR* refers to the full pathname of the main Informix directory.

   For example, assume that *INFORMIXDIR* is **/disk/ifmx**. The following example shows the entry for the EXTRA_LIBS option in the **Configuration** file:

```
EXTRA_LIBS=-lifxw -lifsql -lifasf -lifgen -lifos -lifgls
/disk/ifmx/lib/esql/checkapi.o -lifglx -lifsql -lifasf -lifgen
-lifos -lifgls -lm
```

   The preceding entry must not contain any carriage returns: it must fit on exactly one line.

4. Add a Module directive to the end of the **Configuration** file to specify the location of the Apache Webdriver shared object.

   There are two types of Apache Webdriver shared objects: the basic shared object and the server-side-includes-enabled shared object.

   To specify the basic shared object, add the following lines to the **Configuration** file:

   ```
   ## Add the Informix Webdriver module
   Module informix_module \
      INFORMIXDIR/extend/web.version/apache/apache_version/explode.o
   ```

   Module directives must fit on a single line; for clarity, however, the preceding entry shows the Module directive on two separate lines.

   In the preceding entry, *INFORMIXDIR* refers to the full pathname of the main Informix directory, *version* refers to the current version of the Web DataBlade module installed on your computer, and *apache_version* refers to the name of the directory that corresponds to the version of the Apache Web server you are using.

   For example, assume *INFORMIXDIR* is **/dsk/ifx**, the current version of the Web DataBlade module installed on your computer is 4.13.UC1, and the version of the Apache Web server you are using is 1.3.12. The following example shows the Module entry for the basic shared object in the **Configuration** file:

   ```
   ## Add the Informix Webdriver module
   Module informix_module \
      /dsk/ifx/extend/web.4.13.UC1/apache/apache_1.3.12/explode.o
   ```

   To specify the server-side-includes-enabled shared object, specify the **explode_ssi.o** shared object instead, as shown in the following example:

   ```
   ## Add the Informix Webdriver module
   Module informix_module \
      /dsk/ifx/extend/web.4.13.UC1/apache/apache_1.3.12/explode_ssi.o
   ```

Editing Apache Web Server Source Code

If you want AppPage developers to use server-side includes in their AppPages, you must modify two Apache Web server source code files before you create the Apache Web server binary, as described in the following two sections.

Tip: *Make a backup copy of the two files before you modify them.*

Editing the mod_include.c File

The first file you must edit is called ***APACHEDIR*/src/modules/standard/mod_include.c**, where *APACHEDIR* refers to the main Apache Web server directory.

Change the **mod_include.c** file by removing the static declaration to the **send_shtml_file** function so the Apache Web server can directly call the SSI-enabled Apache Webdriver module.

In particular, change the following line in the **mod_include.c** file:

```
static int send_shtml_file(request_rec *r)
```

Change this line to:

```
int send_shtml_file(request_rec *r)
```

Editing the http_request.c File

The second file you must edit is called ***APACHEDIR*/src/main/http_request.c**, where *APACHEDIR* refers to the main Apache Web server directory.

Change the **http_request.c** file by removing the static declaration to the **get_path_info** function.

In particular, change the following line in the **http_request.c** file:

```
static int get_path_info(request_rec *r)
```

Change this line to:

```
int get_path_info(request_rec *r)
```

Adding URL Prefix Information to the Apache Web Server

The Apache Web server configuration file **httpd.conf**, located in the **conf** directory under the main Apache Web server directory, contains mapping information that tells the Web server to call the Apache Webdriver module when a particular URL prefix is used in a URL.

Each Webdriver mapping in the **web.cnf** file has a corresponding Location directive in the **httpd.conf** file. The Location directive must take the following form:

```
<Location /URL_prefix>
SetHandler informix_explode
</Location>
```

In this directive, */URL_prefix* must be exactly the same as the name of the Webdriver mapping in the Map section of the **web.cnf** file.

Each time you create a new database, register the Web DataBlade module, and install the Web DataBlade Module Administration Tool in the database, you must add the following Location directive in the **httpd.conf** file:

```
<Location /database/admin>
SetHandler informix_explode
</Location>
```

In this directive, *database* is the name of the database in which you registered the Web DataBlade module. Use this URL prefix to invoke the Web DataBlade Module Administration Tool in your browser.

Remember to stop and restart the Apache Web server each time you add a URL prefix to the **httpd.conf** file.

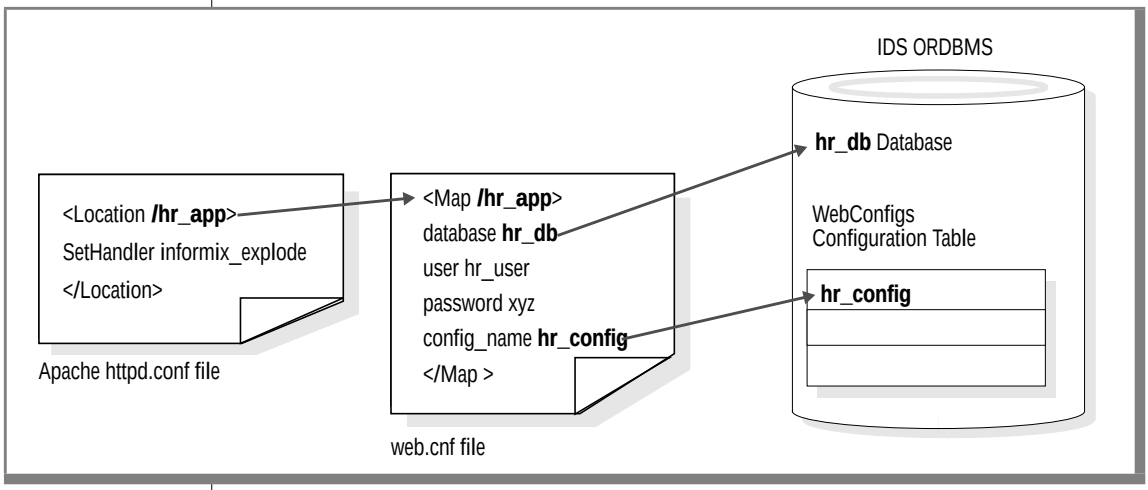
Important: *If you subsequently use the Web DataBlade Module Administration Tool to add a new Webdriver mapping, you must also add a new URL prefix to the Apache Web Server. Be sure the URL prefix is exactly the same as the name of the new Webdriver mapping.*

For detailed information on Webdriver mappings, refer to Chapter 3, "Configuring Webdriver."

How It All Fits Together

Figure 5-1 shows the relationship between the Apache **httpd.conf** file, the **web.cnf** file, and the Webdriver configuration information in the database.

Figure 5-1
Mapping Between Apache httpd.conf File, web.cnf, and WebConfigs Configuration Table



The figure shows a Webdriver mapping in the **web.cnf** file called **/hr_app**. The name of the Location directive in the **httpd.conf** file is also called **/hr_app**. The **web.cnf** file indicates that the **/hr_app** Webdriver mapping uses the **hr_db** database and the **hr_config** Webdriver configuration. For more information on the Web DataBlade Module Administration Tool and Webdriver mappings, refer to Chapter 3, "Configuring Webdriver."

Implementing User Authentication with Apache Webdriver

You can use the *user authentication* feature of the Apache Web server to control user access to AppPages. User authentication refers to the process of users verifying their identity by entering a user name and password when they access secure AppPages.

To implement user authentication with Apache Webdriver

1. Set Webdriver variables in your Webdriver configuration to enable the Apache Webdriver to recognize user authentication.

   For detailed information on this step, refer to "Setting Webdriver Variables" on page 5-16.

2. Enable Apache Web server user authentication by adding the AuthType, AuthName, and require directives to the appropriate entry in the **httpd.conf** file.

   For detailed information about this step, refer to "Updating the httpd.conf File to Enable User Authentication" on page 5-19.

3. Add users to authenticate against to the appropriate database table.

   For detailed information on this step, refer to "Adding Users to the MIusertable Table" on page 5-20.

4. Set the access level for each AppPage for which you want to control access.

   For detailed information on this step, refer to "Specifying AppPage Access Levels" on page 5-21.

Setting Webdriver Variables

The first step to use the security features of the Apache Web server is to set the Webdriver variables listed in the following table. Use the Web DataBlade Module Administration Tool to set these Webdriver variables.

Variable Name	Mandatory?	Content
MIusertable	Yes	Name of the table that contains user access information
MIusername	Yes	Name of the VARCHAR column in the user access table (**MIusertable**) that contains the name of the database user
MIuserpasswd	Yes	Name of the VARCHAR column of the user access table (**MIusertable**) that contains the password of the database user
MIuserlevel	Yes	Name of the INTEGER column of the user access table (**MIusertable**) that contains the access level of the database user
MIpagelevel	Yes	Name of the INTEGER column of the table that stores AppPage that contains the access level of the AppPage

(1 of 2)

Variable Name	Mandatory?	Content
MIusergroup	No	Name of the INTEGER column of the user access table (**MIusertable**) that contains the group access level of the user
redirect_url	No	URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
auth_crypt_udr	No	Enables password encryption when set to `ON` If password encyrption is enabled, Webdriver encrypts the password entered by the user and compares it to the encrypted password in the **MIusertable** table. If they match, then the user is authenticated. If set to `OFF` (default value), then Webdriver does not encrypt the password. Refer to "Using Encrypted Passwords in the MIusertable Table" on page 5-21 for detailed information on using encrypted passwords.

(2 of 2)

For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

Important: *If **MIpagelevel** has not been set for your Webdriver configuration, no user authentication check is performed.*

How the Apache Webdriver Uses the Webdriver Variables

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.

When authorization is not available or access is denied, you can redirect the browser to another URL by setting the **redirect_url** variable to that URL. If **redirect_url** is not set and a user attempts to access an AppPage with an access level higher than 0, an access error is raised. You can access the error by querying the **MI_DRIVER_ERROR** variable in your AppPage. For detailed information on **MI_DRIVER_ERROR**, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Example of Setting User Authentication Webdriver Variables

The following table shows sample values for the Webdriver variables that enable security in an Apache Web server.

Webdriver Variable	Sample Value
MIpagelevel	`read_level`
MIusertable	`wbUsers`
MIusername	`name`
MIuserpasswd	`password`
MIuserlevel	`security_level`
redirect_url	`http://domain/hr_app/errors`

User access settings are based on the **wbUsers** table, described in the schema for APB in the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Updating the httpd.conf File to Enable User Authentication

You enable user authentication for a URL prefix specified in the Location directive of the **httpd.conf** file by adding AuthType, AuthName, and require directives.

Set the AuthType directive to `Basic`, set the AuthName directive to the same URL prefix specified in the Location directive, and set the require directive to `valid-user`.

For example, assume that you previously added the following Location directive to the **httpd.conf** file:

```
<Location /secure>
SetHandler informix_explode
</Location>
```

If you want to enable user authentication for this URL prefix, add the AuthType, AuthName, and require directives, as shown in the following example:

```
<Location /secure>
SetHandler informix_explode
AuthType Basic
AuthName /secure
require valid-user
</Location>
```

The following example shows a Location directive for a guest user who is not authorized to view secure AppPages:

```
<Location /guest>
SetHandler informix_explode
</Location>
```

In the example, the `/secure` URL prefix contains user authentication information; the `/guest` URL prefix does not.

Any AppPage can be accessed only by authorized users with an access level (**MIuserlevel**) equal to or higher than the access level for the AppPage (**MIpagelevel**). Both authorized and unauthorized users can access AppPages from the same AppPage table.

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

```
http://mydomain:port/guest?MIval=/pagename.html
```

An example of a secure URL is:

```
http://mydomain:port/secure?MIval=/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.

Adding Users to the MIusertable Table

Webdriver authenticates users that request a secure AppPage against the list of users stored in the table specified by the **MIusertable** Webdriver variable. This table contains user access information, such as the name of the user, their password, their access level, and so on. When a user requests a secure AppPage, Webdriver checks their inputted password against the password in the **MIusertable** table, looks up the user's access level, checks it against the access level needed to view the AppPage, then decides whether the user is allowed to view the AppPage.

You add users to the **MIusertable** table with an INSERT statement with the DB-Access or SQL Editor tools.

In the APB schema, the table to store user access information is called **wbUsers**. The following example shows how to insert a new user into this table:

```
INSERT INTO wbUsers VALUES
('fred' , 'fred_password', 99, 'APB 2.0', 'AppPage', 80, 20, 't', '0');
```

In the example, user **fred** has an access level of 99. This means that Webdriver allows user **fred** to view AppPages whose access level is 99 or less.

When you install APB into your database, two users are automatically inserted into the **wbUsers** table: **default** and **admin**.

Refer to "Using Encrypted Passwords in the MIusertable Table" on page 5-21 for detailed information on encrypting the user password.

For a detailed description of the columns of the **wbUsers** table, refer to Appendix B of the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Specifying AppPage Access Levels

You specify the access level of an AppPage by updating the access level column in the table that stores the AppPage. The name of the column that contains access level information for each AppPage is specified by the **MIpagelevel** Webdriver variable.

For example, in the APB schema, the table that stores AppPages is called **wbPages**. The **wbPages** table contains a column called **read_level** that specifies the minimum access level a user must have to be able to view the corresponding AppPage. Therefore, for the APB schema, the **MIpagelevel** Webdriver variable is set to `read_level`.

If you want to specify a high access level for a particular AppPage, then update the **read_level** column in the **wbPages** table for that AppPage to the appropriate integer.

For a detailed description of the columns of the **wbPages** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Using Encrypted Passwords in the MIusertable Table

For security reasons, you might want to encrypt all the passwords stored in the **MIusertable** table. This section describes:

- How to encrypt the passwords in the MIusertable table
- How to set the **auth_crypt_udr** Webdriver variable to ensure that Webdriver recognizes encrypted passwords
- How to update APB so it uses encrypted passwords
- Tips on creating your own AppPage to insert and edit users in the **MIusertable** table

Encrypting Passwords

If you set the **auth_crypt_udr** Webdriver variable to ON, Webdriver encrypts the password of the user being authenticated and compares it to the password in the **MIusertable** table. If they match, the user is authenticated.

This means that if you set **auth_crypt_udr** to ON, you must be sure all the passwords stored in the **MIusertable** table are encrypted as well. Use the **webpwcrypt()** routine to encrypt passwords in the **MIusertable** table.

For example, if you are using the APB schema and you already have users in the **wbUsers** table whose passwords are not encrypted, then the following UPDATE statement encrypts the existing passwords:

```
UPDATE wbUsers
SET password = webpwcrypt(password, '');
```

To encrypt passwords as you insert new users into the **wbUsers** table, use an INSERT statement similar to the following example:

```
INSERT INTO wbUsers VALUES
('fred2' , webpwcrypt('fred2_password',''), 99, 'APB 2.0', 'AppPage', 80,
  20, 't', '0');
```

The preceding INSERT statement is very similar to the INSERT statement in "Adding Users to the MIusertable Table" on page 5-20 except that it also executes the **webpwcrypt()** routine on the password column.

For a detailed description of the columns of the **wbUsers** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Setting the auth_crypt_udr Webdriver Variable

Once you have encrypted all the passwords in the **MIusertable** table, you must ensure that Webdriver always encrypts the password entered by a user *before* Webdriver checks to see if the passwords match. By default, Webdriver does not encrypt incoming passwords.

To specify that Webdriver encrypt incoming passwords before checking them against the passwords stored in the **MIusertable** table, set the **auth_crypt_udr** Webdriver variable for your Webdriver configuration to ON.

Tips for Creating Your Own AppPage to Edit User Password Information

If you create your own AppPage to insert or update a user in the **MIusertable**, and you use encrypted passwords, the INSERT or UPDATE statement contains the clear text password as part of the **webpwcrypt()** routine. If you have enabled tracing with the **debug_level** Webdriver variable, then this INSERT or UPDATE statement might be written to the log file, with the clear text password visible.

To ensure that the clear text password is *not* written to the log file, set the **MIdriver** Webdriver variable to `debug_off` to turn off logging of the UPDATE or INSERT statements to the log file. You set the **MIdriver** Webdriver variable as a hidden variable in an INPUT tag after creating a FORM with the attribute REQUEST=POST.

The following section of the **Edit User** AppPage of APB shows an example of how to use the **MIdriver** Webdriver variable to turn off logging of a single INSERT or UPDATE statement:

```
<FORM METHOD=POST ACTION="<?MIVAR>$WEB_HOME<?/MIVAR>">
<INPUT TYPE=hidden NAME=MIval VALUE="/APB20/apb_edit_User.html">
<INPUT TYPE=hidden NAME=MIdriver VALUE="debug_off">
```

Using the REMOTE_USER Web Browser Variable for User Authentication

An additional user authentication method is available for the Apache Webdriver.

If the **connect_as_user** Webdriver variable is set to `ON` in your Webdriver configuration, all database requests connect as the **REMOTE_USER** user instead of the user defined in the **web.cnf** file for your Webdriver mapping. The **REMOTE_USER** user must be added to the user access table identified by the **MIusertable** variable (typically the **wbUsers** table) to enable **connect_as_user** user authentication. The password in the user access table is ignored since user authentication is performed when connecting to the database.

Important: *If you set* ***connect_as_user*** *to* ON*, the* ***REMOTE_USER*** *user must be a valid operating system user with database connection privileges. Therefore, this authentication method should be restricted to Intranet, rather than Internet, applications.*

Using Server-Side Includes in AppPages with the Apache Webdriver

Server-side includes (also called *parsed html*) are special commands embedded in an HTML page that are recognized and interpreted by the Web server; the output of the commands is placed in the HTML page before the AppPage is sent to the browser. Server-side includes can be used, for example, to include a date or time stamp in the text of the HTML page.

The following procedure describes the steps you must take for the Apache Webdriver to correctly handle AppPages that contains server-side includes.

To use server-side includes in an AppPage

1. In your AppPage, use the **PARSE-HTML** variable-processing function together with the MIVAR tag to indicate that your AppPage contains server-side includes.

   The **PARSE-HTML** variable-processing function has two options: SHARED and DYNAMIC.

   The SHARED option specifies that Webdriver send the cached AppPage to the Web server. This option presumes that you have enabled AppPage caching for the AppPage. The following example shows how to specify the SHARED option:

   ```
   <?MIVAR>$(PARSE-HTML,SHARED)<?/MIVAR>
   Webdriver sends the cached AppPage to the Web server.
   ```

   The DYNAMIC option specifies that Webdriver always send the AppPage to the **WebExplode()** function for dynamic processing and then temporarily store the AppPage in a directory that will subsequently be read by the Web server. The following example shows how to specify the DYNAMIC option:

   ```
   <?MIVAR>$(PARSE-HTML,DYNAMIC)<?/MIVAR>
   Webdriver sends the AppPage to the WebExplode function
   and then temporarily stores it in a directory for the
   Web server to read.
   ```

2. If you specified the SHARED option in step 1, enable AppPage caching for the AppPage.

   For detailed information on enabling AppPage caching, refer to Chapter 9, “Improving Performance.”

3. If you specified the DYNAMIC option in step 1, use the Web DataBlade Module Administration Tool to set the **parse_html_directory** Webdriver variable in your Webdriver configuration. For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

   The following table describes this Webdriver variable.

Webdriver Variable	Mandatory?	Description
parse_html_directory	Yes	Specifies the full pathname of the directory on the Web server computer where Webdriver temporarily stores the AppPage to be subsequently read by the Web server Webdriver does not create this directory, so be sure the directory exists before you use server-side includes in an AppPage.

Dynamically Loading the Apache Webdriver

This section describes how to dynamically load the Apache Webdriver shared object into the Apache **httpd** process at runtime. You need to do this if you do not have the Apache source code or if you want to run the default **httpd** binary that was installed at the time you installed the Apache Web server. Otherwise, to configure your Apache Webdriver, refer to "Configuring the Apache Webdriver" on page 5-4.

Before You Begin

Before you can dynamically load the Apache Webdriver shared object, you must ensure that your Apache **httpd** binary contains a necessary module called **mod_so.c**.

As the owner of the Apache Web server, execute the following command at the operating system prompt to list all the modules that have been linked into your Apache **httpd** binary:

```
httpd -l
```

If the **mod_so.c** module is not in the list, you cannot dynamically load the Apache Webdriver at runtime. Instead, you must rebuild your **httpd** binary, as described in "Configuring the Apache Webdriver" on page 5-4.

Updating The Apache Web Server Configuration File

To dynamically load the Apache Webdriver shared object, you must update the Apache configuration file **httpd.cnf**. Assuming you are using Version 4.13 of the Web DataBlade module, add the following LoadModule directive to the **httpd.conf** file.

```
LoadModule informix_module \
$INFORMIXDIR/extend/web.4.13.UC1/apache/apache_ver/explode.so
```

LoadModule directives must fit on a single line; for clarity, however, the preceding entry shows the LoadModule directive on two separate lines.

In the preceding entry, *ver* refers to the version of the Apache Web server.

For example, assume $INFORMIXDIR points to **/disk/ifmx**, and the version of your Apache Web server is 1.3.12. The corresponding LoadModule directive is:

```
LoadModule informix_module \
/disk/ifmx/extend/web.4.13.UC1/apache/apache_1.3.12/explode.so
```

After you have updated the **httpd.conf** file, restart the Apache Web server.

Be sure you set the LD_LIBRARY_PATH and MI_WEBCONFIG environment variables either in the Web server startup script or in the environment of the user who starts the Web server. The LD_LIBRARY_PATH environment variable stores the location of the Informix libraries, and the MI_WEBCONFIG environment variable stores the full pathname of the Webdriver configuration file **web.cnf**.

Chapter

6

Using the ISAPI Webdriver

In This Chapter

This chapter describes how to configure and use the ISAPI Webdriver. It includes the following topics:

Overview of the ISAPI Webdriver

The ISAPI Webdriver uses the Microsoft Windows NT Internet Information Server API rather than a CGI interface to connect to the database and execute AppPages. The ISAPI Webdriver offers the ability to:

- Eliminate CGI process overhead.
- Use security features built into Microsoft Windows NT. For more information, see "Implementing Security with the ISAPI Webdriver" on page 6-11.

The ISAPI Webdriver is a dynamic link library (DLL). Microsoft's Internet Information Server loads the DLL the first time a URL pointing to the ISAPI Webdriver is encountered.

The ISAPI Webdriver is compatible with specific Microsoft Internet Information Server versions. Check the Web DataBlade module release notes for a list of Microsoft Internet Information Server versions that can implement the ISAPI Webdriver.

Configuring the ISAPI Webdriver

The ISAPI Webdriver obtains configuration information about the Web application environment from the **web.cnf** file, the Webdriver configuration information stored in the database, the Microsoft Internet Information Server configuration, and the filtering information from the Internet Service Manager.

The following procedure describes the basic steps you must perform to configure the ISAPI Webdriver. Some of the steps are described in more detail later in this section.

Tip: *This chapter uses the term "websetup" to refer to the* ***setup.exe*** *utility in the directory* ***INFORMIXDIR\extend\web.version\websetup****, where INFORMIXDIR is the main Informix directory and version is the current version of the Web DataBlade module installed on your computer.*

If you used the **websetup** utility to initially configure the Web DataBlade module for your database, the **websetup** utility might have automatically performed some of the steps in the following procedure. In particular, the **websetup** tool:

- Copies and updates the **web.cnf** file with the required information
- Runs the **webconfig** utility to add the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool

For more information on the **websetup** utility, refer to Chapter 2, "Getting Started," and Chapter 13, "Web DataBlade Module Utilities."

To configure the ISAPI Webdriver

1. If you have *never* run the **websetup** utility to configure Web server and database components, go to step 2.

   If you have previously run the **websetup** utility to configure Web server and database components, go to step 4.

2. Copy the sample Webdriver configuration file, called **web.cnf.example**, to a directory on the Web server computer and rename it **web.cnf**. Update the **web.cnf** file with the minimum required information.

   For detailed information on the minimum information needed to update the **web.cnf** file, refer to "Updating the web.cnf File" on page 6-9.

   The **web.cnf.example** file is located in the directory ***INFORMIXDIR*\extend\web.*version*\install**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

   If you already have a working **web.cnf** file, you do not need to perform step 2.

3. Run the **webconfig.exe** utility at the Windows command prompt to add the special Webdriver mapping to the **web.cnf** file to invoke the Web DataBlade Module Administration Tool.

   For detailed information on this step, refer to "Executing the webconfig.exe Utility" on page 6-7.

4. Copy the **drvisapi.dll** file from the directory where the Web DataBlade module is installed to a directory on the Web server computer.

   The **drvisapi.dll** file is located in the directory ***INFORMIXDIR*\extend\web.*version*\microsoft**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

5. Add mapping information to the Microsoft Internet Information Server so that the Web server calls the ISAPI Webdriver when you specify a URL prefix that maps to a Webdriver mapping. At the very least, you must add a URL prefix to invoke the Web DataBlade Module Administration Tool.

   For detailed information on this step, refer to "Adding URL Prefix Information to the Web Server" on page 6-8.

6. Set the Windows NT system environment variable **MI_WEBCONFIG** to the full pathname of the location of the **web.cnf** file.

   Choose **Start→Settings→Control Panel→System** and click the **Environment** tab.

   For example, if you copied the **web.cnf** file to the directory **d:\web\isapi**, set the **MI_WEBCONFIG** variable as follows:

   ```
   MI_WEBCONFIG=d:\web\isapi\web.cnf
   ```

7. Update the Windows NT system environment variable **PATH** to include the directory that contains the Informix executables.

8. Restart the Windows NT computer, so that the new **MI_WEBCONFIG** system environment variable becomes effective in the Internet Information Server Service.

9. Invoke the Web DataBlade Module Administration Tool in your browser by specifying a URL of the following form in your browser:

   ```
   http://domain:port/dbname_admin/drvisapi.dll
   ```

   In this URL, *domain* refers to the name of your Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of your database.

   In this URL, *dbname*`_admin` is the URL prefix you add to the Microsoft Internet Information server, as described in "Adding URL Prefix Information to the Web Server" on page 6-8.

For general information on invoking AppPages with the ISAPI Webdriver, refer to "Invoking AppPages with ISAPI Webdriver" on page 6-10.

After you have invoked the Web DataBlade Module Administration Tool in your browser, use it to add new Webdriver mappings and Webdriver configurations to invoke your own Web DataBlade module applications or existing applications such as AppPage Builder (APB).

For more information on adding Webdriver mappings and Webdriver configurations with the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver." For detailed information on invoking and using APB, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

You can also optionally implement security with the ISAPI Webdriver. For detailed information on this step, refer to "Implementing Security with the ISAPI Webdriver" on page 6-11.

Executing the webconfig.exe Utility

Use the **webconfig.exe** utility to add a special Webdriver mapping to the **web.cnf** file to invoke the Web DataBlade Module Administration Tool for the database for which you are configuring the Web DataBlade module.

Be sure the **MI_WEBCONFIG** system environment variable correctly points to the full pathname of the **web.cnf** file before you run this utility.

Although you can name the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool anything you want, Informix recommends you name it */dbname*_admin, where *dbname* is the name of the database for which you are configuring the Web DataBlade Module Administration Tool.

You *must* specify the **admin** Webdriver configuration with the **-n** option to the **webconfig** utility.

For example, to add a special Webdriver mapping for the Web DataBlade Module Administration Tool for the **production** database and the user **fred**, execute the following command:

```
webconfig -addmap -p /production_admin -n admin -d production -u fred
```

The **webconfig** utility asks for the password for user **fred** and a password key.

The resulting Map section of the **web.cnf** file looks something like the following example:

```
<Map path=/production_admin>
database        production
user            fred
password        84928490340384024343243 24
password_key    akey
config_name     admin
</Map>
```

For detailed information on using the **webconfig** utility, refer to Chapter 13, "Web DataBlade Module Utilities."

Adding URL Prefix Information to the Web Server

This section provides details on how to map the directory in which you put the **drvisapi.dll** to the Microsoft Internet Information Server. This mapping tells the Microsoft Internet Information Server to call the ISAPI Webdriver when a particular URL prefix is used in a URL.

Each time you create a new database, register the Web DataBlade module, and install the Web DataBlade Module Administration Tool in the database, you must add a special URL prefix to the Microsoft Internet Information Server, which invokes the Web DataBlade Module Administration Tool. Typically, this URL prefix takes the form *dbname*_admin, where *dbname* is the name of the database.

Important: *When you subsequently use the Web DataBlade Module Administration Tool to add a new Webdriver mapping, you must also add a new URL prefix to the Microsoft Internet Information Server. Be sure the URL prefix is exactly the same as the name of the new Webdriver mapping.*

For detailed information on Webdriver mappings, refer to Chapter 3, "Configuring Webdriver."

To add mapping information to the Microsoft Internet Information Server

1. Start the Microsoft Internet Service Manager.

   This launches the Microsoft Management Console.
2. Click the **Internet Information Server** expander button.
3. Click the expander button for the current computer connection.

4. Right-click the **Default Web Site** icon and select **New→Virtual Directory**.

   This launches the **New Virtual Directory** wizard.

5. Enter the new URL prefix in the text box of the first page of the wizard.

   The URL prefix for the Web DataBlade Module Administration Tool is typically *dbname*_admin, where *dbname* is the name of the database for which you are configuring the ISAPI Webdriver.

   Be sure the URL prefix is the same as the name of the corresponding Webdriver mapping.

6. Click **Next**.

7. Enter the full path for the directory in which the **drvisapi.dll** file resides in the text box of the second page of the wizard.

8. Click **Next**.

9. Be sure **Allow Execute Access** is checked in the third page of the wizard.

10. Click **Finish** to save the new virtual directory and close the **New Virtual Directory** wizard.

Updating the web.cnf File

This section describes how to update the **web.cnf** file with the required minimum information after you have copied it to a new location on the Web server computer.

Edit the **web.cnf** file by adding or updating the following entries:

- Update the **anchorvar** variable in the Global section of the **web.cnf** file from WEB_HOME to WEB_HOME/drvisapi.dll, as shown in the following example:

```
<Global>
.
anchorvar   WEB_HOME/drvisapi.dll
.
</Global
```

  For detailed information on the **anchorvar** variable, refer to Chapter 3, “Configuring Webdriver.”

- Add the correct values for **INFORMIXDIR** and **INFORMIXSERVER** in the Setvar section of the **web.cnf** file. These two Informix variables describe the main Informix directory and the name of the default Informix database server.

  For detailed information on these variables, refer to the *Administrator's Guide* for your database server.

Invoking AppPages with ISAPI Webdriver

The ISAPI Webdriver is invoked by specifying a URL prefix that has previously been added as mapping information to the Microsoft Internet Information Server along with the **drvisapi.dll** in the URL. Parameters, such as the Webdriver variable **MIval** to specify the AppPage you want to invoke, are passed using standard query string syntax.

For example, assume that you have previously added the URL prefix `mydb_mymap` to the Microsoft Internet Information Server and want to use it to invoke the AppPage called **myAppPage**. The following URL invokes the AppPage:

```
http;//domain:port/mydb_mymap/drvisapi.dll?MIval=myAppPage
```

In this URL, *domain* refers to the Web server computer, and *port* refers to the port of the Web server service.

In your AppPages, specify a URL that uses the ISAPI Webdriver, as shown in the following example:

```
<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=/APB20/apb.html
```

In the preceding example, **WEB_HOME** is an anchor variable, specified by the **anchorvar** variable in the Global section of the **web.cnf** file.

For more information on variables in the Global section of the **web.cnf** file, refer to Chapter 3, "Configuring Webdriver."

For detailed information on adding URL prefixes to the Microsoft Internet Information Server, refer to "Adding URL Prefix Information to the Web Server" on page 6-8. For detailed information on the **MIval** Webdriver variable, refer to Chapter 3.

Using Session Variables with the ISAPI Webdriver

When you use the ISAPI Webdriver in conjunction with session variables, specifically if you set the **session** Webdriver variable to URL, you must also attach the ISAPI filter library to the Microsoft Internet Information Server service. The ISAPI filter library is the same file as the ISAPI Webdriver: **drvisapi.dll**.

For detailed information on how to attach the ISAPI filter library, refer to "Attaching the ISAPI Filter Library" on page 6-14.

For detailed information on session variables, how to use them in an AppPage, and what Webdriver variables have to be set to enable session variables, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Implementing Security with the ISAPI Webdriver

This section describes how to implement Microsoft Internet Information Server user authentication with the ISAPI Webdriver. Most of the steps are described in later sections of this chapter.

To implement user authentication with the ISAPI Webdriver

1. Create a valid Windows NT user with ordinary user privileges.

   User access to AppPages is authenticated at the system level against this user.

2. Set the security-specific Webdriver variables for your Webdriver configuration with the Web DataBlade Module Administration Tool.

   Set the **iis_nt_user** and **iis_nt_password** Webdriver variables to the user you created in step 1 and set the user's password.

   For detailed information on this step, refer to "Setting Webdriver Security Variables" on page 6-12.

3. Attach the ISAPI filter library to the Microsoft Internet Information Server service.

   For detailed information on this step, refer to "Attaching the ISAPI Filter Library" on page 6-14.

4. Turn on the security function of the ISAPI Webdriver.

   For detailed information on this step, refer to "Turning On the Security Feature of the ISAPI Webdriver" on page 6-14.

5. Add users to authenticate against to the appropriate database table.

   For detailed information on this step, refer to "Adding Users to the MIusertable Table" on page 6-15.

6. Set the access level for each AppPage for which you want to control access.

   For detailed information on this step, refer to "Specifying AppPage Access Levels" on page 6-16.

Setting Webdriver Security Variables

Use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table. For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

Variable	Mandatory?	Content
MIusertable	Yes	Name of the table that contains user access information
MIusername	Yes	Name of the VARCHAR column in the user access table (**MIusertable**) that contains the name of the database user
MIuserpasswd	Yes	Name of the VARCHAR column of the user access table (**MIusertable**) that contains the password of the database user
MIuserlevel	Yes	Name of the INTEGER column of the user access table (**MIusertable**) that contains the access level of the database user
MIpagelevel	Yes	Name of the INTEGER column of the table that stores your AppPage that contains the access level of the AppPage

(1 of 2)

Variable	Mandatory?	Content
iis_nt_user	Yes	Name of a valid Windows NT user
iis_nt_password	Yes	Password of a valid Windows NT user
redirect_url	No	URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
auth_crypt_udr	No	Enables password encryption when set to `ON` If password encyrption is enabled, Webdriver encrypts the password entered by the user and compares it to the encrypted password in the **MIusertable** table. If they match, then the user is authenticated. If set to `OFF` (default value), then Webdriver does not encrypt the password. For detailed information on using encrypted passwords, refer to "Using Encrypted Passwords in the MIusertable Table" on page 6-16.

(2 of 2)

After the ISAPI Webdriver security is turned on, user access to AppPages is authenticated against the database table specified by the **MIusertable** Webdriver variable. If you are using the APB schema, this table is called **wbusers**. Add to the **MIusertable** table the user access information for users who are allowed to view AppPages.

The **wbusers** table provides database-level authentication. You enter the user and password information from this table when access to an AppPage is interrupted by a window asking for user validation to view the AppPage. The **iis_nt_user** and **iis_nt_password** Webdriver variables, however, refer to the name and password of a valid Windows NT user you have previously created specifically for system authentication. The authentication against the Windows NT user is performed internally by the ISAPI Webdriver.

Attaching the ISAPI Filter Library

The ISAPI filter library is the same file as the ISAPI Webdriver file: **drvisapi.dll**.

To attach the ISAPI Filter Library to the Microsoft Internet Information Server service

1. Start the Microsoft Internet Service Manager.

   This launches the Microsoft Management Console.
2. Click the **Internet Information Server** expander button.
3. Click the expander button for the current computer connection.
4. Right-click the **Default Web Site** icon and select **Properties**.
5. Select the **ISAPI Filters** page.
6. Click **Add**.
7. Add the full pathname of **drvisapi.dll** into the **Executable** text box from the **Filter Properties** dialog box. Give the filter a meaningful name.

Turning On the Security Feature of the ISAPI Webdriver

The ISAPI Webdriver module allows you to turn the security feature on and off with the Microsoft Internet Services Manager. By default, the security feature is turned off; this section describes how to turn the feature on.

To turn on the security feature of the ISAPI Webdriver

1. Start the Microsoft Internet Service Manager.

   This launches the Microsoft Management Console.
2. Double-click the icon for the URL prefix for which you want to turn on the security feature.
3. Right-click **Properties**.
4. Click the **Directory Security** tab.
5. Click **Edit**. The Authentication Methods window appears.
6. Uncheck **Allow Anonymous Access** and **Windows NT Challenge/Response**.

7. Check **Basic Authentication**.
8. Click **Yes** on the warning window.
9. Click **OK**.
10. Click **Apply**.
11. Click **OK**.

Adding Users to the MIusertable Table

Webdriver authenticates users that request a secure AppPage against the list of users stored in the table specified by the **MIusertable** Webdriver variable. This table contains user access information, such as the name of the user, their password, their access level, and so on. When a user requests a secure AppPage, Webdriver checks their inputted password against the password in the **MIusertable** table, looks up the user's access level, checks it against the access level needed to view the AppPage, then decides whether the user is allowed to view the AppPage.

You add users to the **MIusertable** table with an INSERT statement with the DB-Access or SQL Editor tools.

In the APB schema, the table to store user access information is called **wbUsers**. The following example shows how to insert a new user into this table:

```
INSERT INTO wbUsers VALUES
('fred' , 'fred_password', 99, 'APB 2.0', 'AppPage', 80, 20, 't', '0');
```

In the example, user **fred** has an access level of 99. This means that Webdriver allows user **fred** to view AppPages whose access level is 99 or less.

When you install APB into your database, two users are automatically inserted into the **wbUsers** table: **default** and **admin**.

For detailed information on encrypting the user password, refer to "Using Encrypted Passwords in the MIusertable Table" on page 6-16.

For a detailed description of the columns of the **wbUsers** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Specifying AppPage Access Levels

You specify the access level of an AppPage by updating the access level column in the table that stores the AppPage. The name of the column that contains access level information for each AppPage is specified by the **MIpagelevel** Webdriver variable.

For example, in the APB schema, the table that stores AppPages is called **wbPages**. The **wbPages** table contains a column called **read_level** that specifies the minimum access level a user must have to be able to view the corresponding AppPage. Therefore, for the APB schema, the **MIpagelevel** Webdriver variable is set to `read_level`.

If you want to specify a high access level for a particular AppPage, then update the **read_level** column in the **wbPages** table for that AppPage to the appropriate integer.

For a detailed description of the columns of the **wbPages** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Using Encrypted Passwords in the MIusertable Table

For security reasons, you might want to encrypt all the passwords stored in the **MIusertable** table. This section describes:

- How to encrypt the passwords in the **MIusertable** table
- How to set the **auth_crypt_udr** Webdriver variable to ensure that Webdriver recognizes encrypted passwords
- How to update APB so it uses encrypted passwords
- Tips for creating your own AppPage to insert and edit users in the **MIusertable** table

Encrypting Passwords

If you set the **auth_crypt_udr** Webdriver variable to ON, Webdriver encrypts the password of the user being authenticated and compares it to the password in the **MIusertable** table. If they match, the user is authenticated.

This means that if you set **auth_crypt_udr** to ON, you must be sure all passwords stored in the **MIusertable** table are encrypted as well. Use the **webpwcrypt()** routine to encrypt passwords in the **MIusertable** table.

For example, if you are using the APB schema and you already have users in the **wbUsers** table whose passwords are not encrypted, then the following UPDATE statement encrypts the existing passwords:

```
UPDATE wbUsers
SET password = webpwcrypt(password, '');
```

To encrypt passwords as you insert new users into the **wbUsers** table, use an INSERT statement similar to the following example:

```
INSERT INTO wbUsers VALUES
('fred2' , webpwcrypt('fred2_password',''), 99, 'APB 2.0', 'AppPage', 80,
  20, 't', '0');
```

The preceding INSERT statement is very similar to the INSERT statement in "Adding Users to the MIusertable Table" on page 6-15 except that it also executes the **webpwcrypt()** routine on the password column.

For a detailed description of the columns of the **wbUsers** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide.*

Setting the auth_crypt_udr Webdriver Variable

Once you have encrypted all the passwords in the **MIusertable** table, you must ensure that Webdriver always encrypts the password entered by a user *before* Webdriver checks to see if the passwords match. By default Webdriver does not encrypt incoming passwords.

To specify that Webdriver encrypt incoming passwords before checking them against the passwords stored in the **MIusertable** table, set the **auth_crypt_udr** Webdriver variable for your Webdriver configuration to ON.

Tips for Creating Your Own AppPage to Edit User Password Information

If you create your own AppPage to insert or update a user in the **MIusertable**, and you use encrypted passwords, the INSERT or UPDATE statement contains the clear text password as part of the **webpwcrypt()** routine. If you have enabled tracing with the **debug_level** Webdriver variable, then this INSERT or UPDATE statement might be written to the log file, with the clear text password clearly visible.

To ensure that the clear text password is *not* written to the log file, set the **MIdriver** Webdriver variable to `debug_off` to turn off logging of the UPDATE or INSERT statements to the log file. You set the **MIdriver** Webdriver variable as a hidden variable in an INPUT tag after creating a FORM with the attribute REQUEST=POST.

The following section of the **Edit User** AppPage of APB shows an example of how to use the **MIdriver** Webdriver variable to turn off logging of a single INSERT or UPDATE statement:

```
<FORM METHOD=POST ACTION="<?MIVAR>$WEB_HOME<?/MIVAR>">
<INPUT TYPE=hidden NAME=MIval
VALUE="/APB20/apb_edit_User.html">
<INPUT TYPE=hidden NAME=MIdriver VALUE="debug_off">
```

Using the REMOTE_USER Web Server Variable for User Authentication

An additional user authentication method is available for the ISAPI Webdriver.

If the **connect_as_user** Webdriver variable is set to `ON` in your Webdriver configuration, all database requests connect as the **REMOTE_USER** user instead of the user defined in the **web.cnf** file for your Webdriver mapping. The **REMOTE_USER** user must be added to the user access table identified by the **MIusertable** variable (typically the **wbUsers** table) to enable **connect_as_user** user authentication. The password in the user access table is ignored since user authentication is performed when connecting to the database.

Important: *If you set* ***connect_as_user*** *to* ON*, the* ***REMOTE_USER*** *user must be a valid operating system user with database connection privileges. Therefore, this authentication method should be restricted to Intranet, rather than Internet, applications.*

Executing ISAPI Functions in an AppPage

The ISAPI Webdriver allows you to load ISAPI-compliant code modules, or *ISAPI functions*, into the Web server at runtime. When you include these ISAPI functions as Web server extensions, you can use the ISAPI Webdriver and the MIFUNC tag to call the ISAPI functions within an AppPage.

When the **WebExplode()** function encounters an MIFUNC tag in an AppPage, it passes the name of the DLL and the function name within the DLL to the ISAPI Webdriver. The ISAPI Webdriver then loads the DLL and calls the specified function, with two callbacks as parameters. After the function is executed, any value that has been modified is passed back to the **WebExplode()** function.

To call an ISAPI function in an AppPage

1. Create and build the DLL.

   For detailed information on this step, refer to "Creating and Building the DLL," following.

2. Copy the DLL to a directory that is included in the **PATH** system environment variable.

   Alternatively, you can specify the full pathname of the DLL in the DLL attribute of the MIFUNC tag. The MIFUNC tag is described in the next step.

3. Invoke the function in the MIFUNC tag within an AppPage.

   For detailed instructions for this step, see "Invoking ISAPI Functions in an AppPage" on page 6-21.

Creating and Building the DLL

The ISAPI functions invoked with the MIFUNC tag must have the following format:

```
void WINAPI function( char *(WINAPI *GetVar) ( char *name ),
   BOOL  (WINAPI *SetVar) ( char *name, char *value ) )
```

The **GetVar** function takes a string that corresponds to the name in a name/value pair and returns a pointer to the value string.

For example, the following code looks up the SYMBOL value in the specified parameters and returns a pointer to the value. The function returns NULL if the name was not specified in the MIFUNC tag that invoked the function.

```
char *stockSymbol  = GetVar( "SYMBOL" );
```

The **SetVar** function takes name and value strings and overwrites the current value for that name. The function returns false if the name was not specified in the MIFUNC tag that invoked the function. The following sample code shows how to use the **SetVar** function:

```
BOOL success = SetVar( "SYMBOL",  "IFMX" );
```

The following sample function from a DLL, called **GetTime**, shows an example of C code that returns the time and the number of times the **GetTime** function has been called:

```
#include <windows.h>
#include <stdio.h>
#include <time.h>

void WINAPI function( char *(WINAPI *GetVar) ( char *name ),
   BOOL  (WINAPI *SetVar) ( char *name, char *value ) );

int i = 0;  /* Keeps track of how many times GetTime() is called */

void WINAPIGetTime( char *(WINAPI *GetVar) ( char *name ),
   BOOL  (WINAPI *SetVar) ( char *name, char *value ) )

{
   struct tm *newtime;
   time_t aclock;
   char   buf[10];
   time( &aclock );                 /* Get time in seconds */
   newtime = localtime( &aclock );  /* Convert time to struct */
                                    /* tm form */
   SetVar( "TIME", asctime( newtime ) );
   itoa(++i, buf, 10);
   SetVar( "COUNT", buf );
}
```

Use the following compile and link command at the Windows command prompt to build the DLL once you have finished coding it:

```
cl /LD stat.c /link /EXPORT:GetTime
```

The example generates the DLL called **stat.dll** in the current directory.

Invoking ISAPI Functions in an AppPage

Within the MIFUNC tag, you must include the variables to be imported and exported (passed by reference) as name / value pairs. You specify the name of the DLL with the DLL attribute. The FUNCTION attribute specifies the ISAPI function in the DLL to call.

When the MIFUNC tag is executed, all AppPage processing stops until the function has completed execution. When AppPage processing continues, everything between the MIFUNC tags is executed using the variables that have been modified by reference in the ISAPI function, in addition to the variables originally supplied to the AppPage.

The following AppPage invokes the **GetTime** function in **stat.dll**. The AppPage displays the time and the number of times that the **GetTime** function has been called:

```
<?MIFUNC DLL=stat FUNCTION=GetTime TIME="" COUNT="">
<?MIVAR>$TIME Count = $COUNT<?/MIVAR>
<?/MIFUNC>
```

The sample **GetTime** function and **stat.dll** are described in the section "Creating and Building the DLL" on page 6-20.

You can nest MIFUNC tags so that one function can execute depending on the results of another. You can also include any number of MIFUNC tags in an AppPage; however, you should pay attention to the order of the tags to achieve the desired result.

For more information on using the MIFUNC Web DataBlade module tag in AppPages, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Chapter

7

Using the CGI Webdriver

In This Chapter

This chapter describes how to configure and use the CGI Webdriver. It includes the following topics:

Overview of the CGI Webdriver

The CGI Webdriver is a CGI (Common Gateway Interface) program that connects to a database and executes AppPages. You can use the CGI Webdriver with any Web server, since all Web servers are able to execute CGI programs.

It is recommended that you use the CGI Webdriver only when a Webdriver implementation for a specific Web server is unavailable. For example, if you use the Apache Web server, you should use the Apache Webdriver. Similarly, if you use the Netscape Web server, you should use the NSAPI Webdriver.

Configuring the CGI Webdriver

The following procedure describes the basic steps you must perform to configure the CGI Webdriver for your database. Some of the steps are described in later sections.

If you used the **websetup** utility to initially configure the Web DataBlade module for your database, the **websetup** utility might have automatically performed some of the steps in the following procedure. In particular, the **websetup** utility:

- Copies and updates the **web.cnf** file with the required information
- Runs the **webconfig** utility to add the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool

For more information on the **websetup** utility, refer to Chapter 2, "Getting Started," and Chapter 13, "Web DataBlade Module Utilities."

To configure the CGI Webdriver for your database

1. As the owner of the Web server installation, create a directory on your Web server to contain CGI programs.

   For detailed information on this step, refer to "Creating a CGI Directory for Your Web Server" on page 7-6.

2. Copy the CGI Webdriver program to the CGI directory you created in step 1.

   The CGI Webdriver program is called **webdriver** and is located in the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the directory in which the Informix database is installed and *version* refers to the latest version of the Web DataBlade module installed on your computer. On Windows NT, the CGI Webdriver program is called **webdriver.exe**.

3. Copy the file **web.cnf.example** to the CGI directory you created in step 1 and rename it **web.cnf**.

   The **web.cnf.example** file is located in the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the directory in which the Informix database is installed and *version* refers to the latest version of the Web DataBlade module installed on your computer.

   The CGI Webdriver ignores the **MI_WEBCONFIG** variable that other implementations of Webdriver use to locate the **web.cnf** file. Instead, CGI Webdriver *always* looks for the **web.cnf** file in the same directory as the CGI Webdriver program.

4. Edit the **web.cnf** file you just copied to the CGI directory, adding or updating the required minimum entries.

   For detailed information on this step, refer to "Updating the web.cnf File" on page 7-6.

5. Run the **webconfig** utility at the operating system command prompt to add the special Webdriver mapping to the **web.cnf** file used to invoke the Web DataBlade Module Administration Tool.

   For detailed information on this step, refer to "Executing the webconfig Utility" on page 7-8.

6. Invoke the Web DataBlade Module Administration Tool in your browser by specifying a URL of the following form in your browser:

   ```
   http://domain:port/dbname/admin/webdriver
   ```

   In this URL, *domain* refers to the name of your Web server computer, *port* refers to the port number of the Web server process, and *dbname* refers to the name of your database.

   */dbname/*admin is the name of the CGI directory you added to your Web server as described in step 1.

For general information on invoking AppPages with the CGI Webdriver, refer to "Invoking AppPages with the CGI Webdriver" on page 7-9.

After you have invoked the Web DataBlade Module Administration Tool in your browser, use it to add new Webdriver mappings and Webdriver configurations to invoke your own Web DataBlade module applications or existing applications such as AppPage Builder (APB).

For more information on adding Webdriver mappings and Webdriver configurations with the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver." For detailed information on invoking and using APB, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Creating a CGI Directory for Your Web Server

When you create a new CGI directory for your Web server, be sure you update the Web server configuration files so that the Web server executes the programs in this directory instead of displaying them to the browser.

Each time you use the Web DataBlade Module Administration Tool to add a new Webdriver mapping, you must also update your Web server configuration to add a new virtual CGI directory. This virtual CGI directory can map to a new, actual CGI directory on your computer or to an existing, actual CGI directory.

This section describes how to create the CGI directory that corresponds to the special Webdriver mapping used to invoke the Web DataBlade Module Administration Tool. It is recommended that you call the CGI directory for this special Webdriver mapping `/`*`dbname`*`/admin`, where *dbname* is the name of your database.

For example, you might create a CGI directory called `/hr_db/admin` for the **hr_db** database whose actual full pathname on the Web server computer is **/local/webserver/hr_db/admin**. This CGI directory name will be used to invoke the Web DataBlade Module Administration Tool for the **hr_db** database.

When you subsequently use the Web DataBlade Module Administration Tool to add a new Webdriver mapping, you must also add a new CGI directory to your Web server. Be sure the name of the CGI directory is exactly the same as the name of the new Webdriver mapping.

For example, if you create a Webdriver mapping called **/hr_app**, you must create a CGI directory called **/hr_app**. The new CGI directory should have the same full pathname as the CGI directory that corresponds to the special Webdriver mapping that invokes the Web DataBlade Module Administration Tool. In the example above, this pathname is **/local/webserver/hr_db/admin**.

For detailed information on Webdriver mappings, refer to Chapter 3.

Refer to your Web server documentation for detailed information on how to create a CGI directory.

Updating the web.cnf File

This section describes how to update the **web.cnf** file with the required information after you have copied it to the CGI directory.

Edit the **web.cnf** file by adding or updating the following entries:

- Update the **anchorvar** variable in the Global section from `WEB_HOME` to `WEB_HOME/webdriver`, as shown in the following example:

```
<Global>
.
anchorvar   WEB_HOME/webdriver
.
</Global>
```

  On Windows NT, specify `webdriver.exe`, as shown in the following example:

```
<Global>
.
anchorvar   WEB_HOME/webdriver.exe
.
</Global>
```

  For detailed information on the **anchorvar** variable, refer to Chapter 3, "Configuring Webdriver."

- Add the correct values for the Informix environment variables **INFORMIXDIR** and **INFORMIXSERVER** in the Setvar section. These two Informix variables describe the main Informix directory and the name of the default Informix database server.

  For detailed information on these variables, refer to the *Administrator's Guide* for your database server.

Executing the webconfig Utility

Use the **webconfig** utility to add a special Webdriver mapping to the **web.cnf** file to invoke the Web DataBlade Module Administration Tool for the database for which you are configuring the Web DataBlade module.

To execute the **webconfig** utility successfully, you must set the **MI_WEBCONFIG** variable in your environment to point to the full pathname of the **web.cnf** file.

Important: *You only set the **MI_WEBCONFIG** variable in your environment when you execute the **webconfig** utility to configure CGI Webdriver. When you use the CGI Webdriver, CGI Webdriver ignores the **MI_WEBCONFIG** variable in your environment. Instead, CGI Webdriver always looks for the **web.cnf** file in the same directory as the CGI Webdriver program.*

Although you can name the special Webdriver mapping to invoke the Web DataBlade Module Administration Tool anything you want, it is recommended that you name it */dbname/*admin, where *dbname* refers to the name of the database for which you are configuring the Web DataBlade Module Administration Tool.

You *must* specify the **admin** Webdriver configuration with the **-n** option to the **webconfig** utility.

For example, to add a special Webdriver mapping for the Web DataBlade Module Administration Tool for the **hr_db** database and the user **fred**, execute the following command:

```
webconfig -addmap -p /hr_db/admin -n admin -d hr_db -u fred
```

The **webconfig** utility asks for the password for user **fred** and a password key.

The resulting Map section in the **web.cnf** file looks something like this:

```
<Map path=/hr_db/admin>
database       hr_db
user           fred
password       84928490340384024343243 24
password_key   akey
config_name    admin
</Map>
```

For detailed information on using the **webconfig** utility, refer to "The websetup Utility" on page 13-14.

Invoking AppPages with the CGI Webdriver

You invoke the CGI Webdriver by specifying the CGI directory on your Web server that you created to contain the CGI Webdriver, along with the **webdriver** CGI program in the URL. Parameters, such as the Webdriver variable **MIval** to specify the AppPage you want to invoke, are passed using standard query string syntax.

For the CGI Webdriver, each time you add a new Webdriver mapping with the Web DataBlade Module Administration Tool, you must add a new CGI directory to your Web server. The new CGI directory should map to the actual directory on the operating system that contains the **webdriver** CGI program.

For example, assume you have previously added a CGI directory called `/mymap` to your Web server and want to use it to invoke the AppPage called **/myAppPage.html**. The following URL invokes the AppPage:

```
http://domain:port/mymap/webdriver?MIval=/myAppPage.html
```

In this URL, *domain* refers to the Web server computer, and *port* refers to the port of the Web server service.

On Windows NT, specify the **webdriver.exe** CGI program, as shown in the following example:

```
http://domain:port/mymap/webdriver.exe?MIval=/myAppPage.html
```

In your AppPages, specify a URL that uses the CGI Webdriver as shown in the following example:

```
<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=/APB20/apb.html
```

In the preceding example, **WEB_HOME** is an anchor variable, specified by the **anchorvar** variable in the Global section of the **web.cnf** file.

For more information on variables in the Global section of the **web.cnf** file, Webdriver variables, Webdriver mappings, and Webdriver configurations, refer to Chapter 3, "Configuring Webdriver."

For more information about using **WEB_HOME** to create dynamic links between AppPages, see the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Chapter

8

Implementing Security

In This Chapter

You can implement security in a Web DataBlade module application by restricting access to the database, to AppPages, and to large objects.

This chapter describes how to enable security with the IBM Informix Web DataBlade module. It includes the following topics:

For Web-server specific security for the NSAPI, Apache, and ISAPI Webdrivers, refer to the following chapters:

Database Access Security

You can control access to the database by specifying the user with which all connections to the database are made. When you create a Webdriver mapping with the Web DataBlade Module Administration Tool, you specify the user who makes the client Webdriver connection to the database server. The tool automatically encrypts the user's password before it writes the information in the Map section of the **web.cnf** file. For added security, you can use your own encryption key to encrypt the password and update the **web.cnf** file manually.

The database server requires that a client database connection satisfy one of the following conditions:

- The process is running as the user who makes the connection request.
- The password for the user is supplied if the process is running as another user.

Webdriver mappings in the configuration file, typically called **web.cnf**, define the database to connect to and the user to connect to the database as. The Webdriver mappings also define the user's password, if needed.

Important: *Typically, you use the Web DataBlade Module Administration Tool to update Webdriver mappings and do not need to update the* ***web.cnf*** *file manually. The following sections, therefore, are for your information only.*

For detailed information about using the Web DataBlade Module Administration Tool and a description of Webdriver mappings, refer to *Chapter 3, "Configuring Webdriver."*

Assume that the name of the database you want to connect to with the `/hr_map` Webdriver mapping is **hr_db** and that the owner of the database is the **webuser** user. The Webdriver mapping in your **web.cnf** file would contain the following definitions:

```
<Map path=/hr_map>
database    hr_db
user        webuser
config_name hr_config
</Map>
```

If the Web server has been configured to run as the **webuser** user, you do not need to specify a password in the **web.cnf** file. However, if the Web server runs as **nobody** (as is often the case), or as any user other than **webuser**, you must specify a password for the **webuser** user. The **web.cnf** file would then look like the following example:

```
<Map path=/hr_map>
database       hr_db
user           webuser
password       849284903403840243432432 4
password_key   webuser_key
config_name    hr_config
</Map>
```

The Web DataBlade Module Administration Tool automatically creates an encrypted password so that the actual password is not stored in the **web.cnf** file.

Important: *Webdriver does not pass either the* ***password*** *or the* ***password_key*** *variables in the Map section of the* ***web.cnf*** *file to the* ***WebExplode()*** *function. The values of the variables cannot be viewed in RAW mode.*

Encrypting Passwords Manually

You typically do not have to update the **web.cnf** file with password or password key information manually because this is automatically done for you when you update Webdriver mappings with the Web DataBlade Module Administration Tool.

The Web DataBlade Module Administration Tool, however, uses its own key when it encrypts the password. If you want to use your own key to encrypt the password, you must use the **webpwcrypt** utility and then update the **web.cnf** file manually.

The **webpwcrypt** utility takes three arguments and requires you to enter the password for the database user:

```
webpwcrypt <database> <user> <key>
```

The *database* argument is the name of the database being accessed, *user* is the name of the user accessing the database, and *key* is the user-supplied string used in the encryption process. The utility prompts you for the user's password; the password is not echoed.

The following command encrypts the **webuser** password with the user-supplied string webkey:

```
webpwcrypt webdb webuser webkey
Enter password for user "webuser":    <enter webuserpassword>
Enter password again:                 <re-enter webuserpassword>
```

The **webpwcrypt** utility returns:

```
password       c47c6e1c91d32affd138212b24277f85
password_key   webkey
```

Copy the preceding variables and values into the **web.cnf** file:

```
<Map path=/hr_map>
database       hr_db
user           webuser
password       c47c6e1c91d32affd138212b24277f85
password_key   webkey
config_name    hr_config
</Map>
```

Afterward, Webdriver can connect to the database without running as the **webuser** user or specifying the webuserpassword password in the **web.cnf** file.

Resetting User Name/Password Combinations

User name / password combinations are cached in the Web server. This can cause problems for applications in which users change their passwords.

The **auth_cache** Webdriver variable allows you to reset user name / password combinations so users can change their passwords within an application.

The **auth_cache** Webdriver variable allows the settings described in the following table.

Variable	Mandatory?	Description
auth_cache	Yes	Allows you to reset user name and password combinations so users can change their passwords within in application You can set the auth_cache Webdriver variable to three values: on, `off`, and check. The default value is on. If you set the variable to on, Webdriver always uses the password value in the Web server cache. If you set the variable to `off`, Webdriver always uses the password value in the database. If you set the variable to check, if the value in the Web server cache is different from the Web browser value, Webdriver updates the Web server cache with the password value in the database.

To set the **auth_cache** Webdriver variable for your Webdriver configuration, use the Web DataBlade Module Administration Tool. For more information on this tool, refer to Chapter 3, “Configuring Webdriver.”

AppPage-Level Security

You can restrict access to AppPages by:

- Enabling the simple AppPage security feature, described in the next section
- Configuring your Web server to use the user authentication feature and configuring the appropriate Webdriver to recognize Web server user authentication

  Restricting access to AppPages with user authentication is only available for the NSAPI, Apache, and ISAPI Webdrivers. For detailed instructions for enabling user authentication with these Webdrivers, refer to the corresponding Webdriver chapters. In particular refer to:

  - "Implementing User Authentication with the NSAPI Webdriver" on page 4-20
  - "Implementing User Authentication with Apache Webdriver" on page 5-15
  - "Implementing Security with the ISAPI Webdriver" on page 6-11

Configuring Simple Webdriver AppPage-Level Security

By default, all AppPages are visible to all users who are able to connect to the database that contains the AppPages. Sometimes, however, it is desirable to limit access to some AppPages. By configuring certain Webdriver variables, you can perform this AppPage-level authorization.

To configure AppPage-level authorization, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table. For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3.

Variable	Mandatory?	Description
MIpagelevel	Yes	Specifies the name of the INTEGER column of the table that stores AppPages that contains the access level of the AppPage
MI_WEBACCESSLEVEL	Yes	Specifies the access level of all users for a particular Webdriver configuration
redirect_url	No	Specifies the URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
error_page	No	Set to the value of the AppPage that contains error handling routines

Important: *If the **MIpagelevel** variable is not set for your Webdriver configuration, no security check is performed.*

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 **redirect_url** variable to that URL. If **redirect_url** 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 the **error_page** Webdriver variable is set, Webdriver calls the corresponding AppPage and all error handling is processed on that page. If the **error_page** Webdriver variable is not set, the **MI_DRIVER_ERROR** variable is set and the requested page is processed. For detailed information on using the **MI_DRIVER_ERROR** variable, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

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.

Example of Setting Simple AppPage-Level Security

The following table shows sample Webdriver variable settings used to enable AppPage-level security.

Webdriver Variable	Sample Value
Mipagelevel	`read_level`
MI_WEBACCESSLEVEL	`1`
redirect_url	`http://cgi-bin/errors`

The name of the column in the table that stores the AppPages is **read_level**. Since **MI_WEBACCESSLEVEL** is set to `1`, the only AppPages that can be accessed by users are those whose **read_level** value is `0` or `1`.

AppPages whose **read_level** value is greater than `1` cannot be read by users. If users try to access these pages, they are redirected to the following URL: `http://cgi-bin/errors`.

Large Object Security

As described in the *IBM Informix Web DataBlade Module Application Developer's Guide*, the simplest way to retrieve a large object stored in the **wbBinaries** table into your AppPage is to use the **MIval** Webdriver variable to specify the path, ID, and extension of the large object, as shown in the following example:

```
<IMG SRC=<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=/images/flower.gif>
```

By default, all large objects stored in the **wbBinaries** table are visible to any user who is able to connect to the database. Sometimes, however, it is desirable to limit the access to specific large objects in this table to specific users. This section shows how you can secure large objects by customizing the query that Webdriver uses to retrieve the large objects from a user-defined table rather than the **wbBinaries** table.

Setting Webdriver Variables

To customize the query that Webdriver uses to retrieve large objects, add the Webdriver variables described in the following table to your Webdriver configuration using the Web DataBlade Module Administration Tool.

Variable	Mandatory?	Content
lo_query_string	Yes	Contains the SQL statement that is used to query the database for a large object Use standard C language variable syntax '%s' to specify a parameter string.
lo_query_params	Yes	Specifies the variables that are substituted for the parameters in the SQL statement specified by the **lo_query_string** variable You *must* use the variable name **MIvalObj** to specify the name of the large object you want to retrieve.
lo_error_zerorows	No	Specifies the integer error number that Webdriver should return if the SQL statement that Webdriver uses to retrieve large objects, specified by the **lo_query_string** variable, returned zero rows
lo_error_sql	No	Specifies the integer error number that Webdriver should return if an SQL error occurs when Webdriver retrieves a large object using the SQL statement specified by the **lo_query_string** variable

For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

The following sections show an example of how to use these Webdriver variables to secure large objects.

Background for the Example

In this example, assume that the only types of large objects that your Web DataBlade module application uses are images and that you want to store all these images in a table called **pictures**. The **pictures** table has the following schema:

```
CREATE TABLE pictures
(
    id           VARCHAR(20) PRIMARY KEY,
    description  VARCHAR(200),
    picture      BLOB
);
```

Another table, **authorization**, specifies the users that can access each picture in the **pictures** table. For every picture in the **picture** table, the **authorization** table contains a row for every user who is allowed to view the picture. The **authorization** table has the following schema:

```
CREATE TABLE authorization
(
    id           VARCHAR(20) REFERENCES pictures (id),
    authorize    VARCHAR(40)
);
```

Further assume that you use one of the non-CGI implementations of Webdriver (NSAPI, ISAPI, or Apache) and take advantage of the Web server authentication, as detailed in the chapters that describe each type of non-CGI Webdriver.

Finally, you want to limit access of some of the images in the **pictures** table to certain users, according to the value of the **REMOTE_USER** Web server environment variable when the user views AppPages in a browser. This means that when an AppPage retrieves a large object from the **pictures** table, the value of the **REMOTE_USER** variable will be verified against the users who are authorized to view the large object, specified in the **authorization** table.

Implementation of the Example

To implement the example, you must retrieve the large objects into your AppPage by specifying a custom SELECT statement that Webdriver runs when it retrieves large objects rather than letting Webdriver construct its own default query of the **wbBinaries** table.

Use the Web DataBlade Module Administration Tool to set the **lo_query_string** and **lo_query_params** Webdriver variables to the values shown in the following table.

Webdriver variable	Value
lo_query_string	`SELECT picture::lvarchar FROM pictures, authorization WHERE pictures.id = authorization.id AND pictures.id = '%s' AND authorization.user = '%s';`
lo_query_params	`MIvalObj, REMOTE_USER`

For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3. For information about using Web server variables in your AppPages, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Then, in your AppPage, use the **MIvalObj** variable to specify the large object you want to retrieve. For example, the following URL that uses the `/sales` URL prefix retrieves a large object called **my_logo**:

```
http://domain:port/sales?MIvalObj=my_logo
```

If the value of the **REMOTE_USER** browser variable is `mary`, Webdriver uses the values of Webdriver variables passed in the URL, along with the Webdriver variables stored in the Webdriver configuration, to internally execute the following SELECT statement to retrieve the large object:

```
SELECT picture FROM pictures, authorization WHERE
    pictures.id = authorization.id AND
    pictures.id = 'my_logo' AND
    authorization.id = 'mary';
```

If the preceding query returns a row, the user **mary** is authorized to view the **my_logo** image, and Webdriver renders the image in the browser. If, however, the query does *not* return a row, user **mary** is not authorized to view the image, and the image is not rendered in the browser.

Use the **lo_error_zerorows** Webdriver variable to specify the integer that Webdriver should return to the AppPage if the customized SELECT statement returns no rows. Use the **lo_error_sql** Webdriver variable to specify the integer that Webdriver should return to the AppPage if the query returns an error.

For more information on retrieving large objects, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Chapter

9

Improving Performance

In This Chapter

This chapter describes how to improve the performance of applications that use the IBM Informix Web DataBlade module. It includes the following topics:

Overview of Performance

The main method of improving the performance of Web DataBlade module applications is to enable caching of both AppPages and large objects (such as images and video clips). Retrieving data from the disk is always much faster than retrieving data from the database, which is why caching improves the performance of Web applications.

When you enable AppPage caching, static AppPages are stored on the database server computer's disk the first time they are passed through the **WebExplode()** function. In subsequent calls to the AppPage, Webdriver calls the cached AppPage instead of requesting the AppPage from the database and passing it through the **WebExplode()** function.

You can use partial AppPage caching for AppPages that include some dynamic content. The static portion of the AppPage is cached, and only the dynamic content is processed by the **WebExplode()** function.

Large object caching is similar to AppPage caching except that it is large objects, instead of AppPages, that are cached to disk.

AppPage Caching

If your application contains many static AppPages, you can improve performance by eliminating some database requests and retrieving AppPages directly from Webdriver's disk cache. While all AppPages are dynamically generated by the **WebExplode()** function, there are different levels of volatility for different types of AppPages:

- **Single instance**. The AppPage does not change. Variables passed into the AppPage have no effect on the content of the AppPage.
- **Multiple instances, low volatility**. Different instances of the AppPage are generated for different values of the variables passed into the AppPage. The underlying data changes at known intervals. For example, you might have a products table that is updated once an hour.
- **Multiple instances, high volatility**. Different instances of the AppPage are generated for different values of the variables passed into the AppPage. The underlying data can change at any time. For example, a real-time data feed is updated constantly.

In all of the preceding cases, AppPage caching significantly improves performance if the AppPage is retrieved many times before the underlying data changes.

Important: *Webdriver does not detect when the underlying data has changed. For this reason, use the features described in* *"Setting AppPage Caching for a Webdriver Configuration" on page 9-9* *to force AppPages to be refreshed after a specified length of time or to purge AppPages when necessary.*

AppPages That Are Not Cached

Webdriver's implementation of AppPage caching checks all of the variables on the AppPage and creates an instance of the AppPage for each possible set of variable assignments for the AppPage.

AppPages with the following characteristics, however, are never cached by Webdriver:

- The AppPage contains the HTTPHEADER variable-processing function.

- The AppPage contains a session variable (a variable that is prefixed with the **session** keyword) in the static or nondeferred content of the AppPage.

  For more information on creating deferred content in an AppPage, refer to "Partial AppPage Caching" on page 9-25. For more information on session variables, refer to the "Using Session Variables to Improve Performance" on page 9-31.

- The AppPage does not have caching enabled.

Important: *If you have enabled both session management and AppPage caching for your Webdriver configuration, refer to "Session Management and AppPage Caching" on page 9-31 for detailed information on how to ensure that AppPages that contain only static content are cached correctly.*

Global Cache For Dynamic Tags and User-Defined Routine Tags

The current version of the DataBlade module provides a global cache for dynamic tags and user-defined routine (UDR) tags. This global cache is shared by all sessions using the same database. In previous versions of the DataBlade module, each session had its own cache.

The advantages of a global cache are:

- **Increased performance**. Tags are loaded the first time they are used, then subsequent sessions can use the cached tags.
- **Reduced memory usage**. Only one copy of the tag is stored in the cache rather than many copies for each session.
- **Automatic tag updates**. The tag cache is updated whenever the tags table is updated.

Compatibility with Previous Versions

The global tag cache is largely compatible with the previous tag caching scheme except that:

- The global tag cache is always kept up to date, even when the system tables that store tag information are updated.
- If you have implemented your own tag table, you must install triggers on the tag table.
- The global tag cache is *not* automatically updated when the system tables that store tag information are dropped. In this case, you must explicitly clear the cache by restarting the database server or calling a user-defined routine, described in the sections following.
- Since the global cache is global to the database, a tag has the same meaning to all users of the database. This means that all users use the same SELECT statement to fetch tags from the database.

Important: *The global tag cache does not support multiple tags tables; you should only enable it for databases that have a single tags table.*

Enabling the Global Tag Cache

You enable the global tag cache using either the **websetup** utility or the **installGlobalTagCache** utility.

If you use **websetup**, you are prompted for the following information:

1. The name of the table that contains the dynamic tags
2. The name of the tag table column that contains the tag names
3. The name of the tag table column that contains the tag parameter list
4. The name of the tag table column that contains the tag bodies
5. The WHERE clause used to select the row containing a particular tag

   Use the variable $MI_WEBTAGSID for the target tag name.

For example, to be compatible with the APB2/DDW schema, enter the following values for the five preceding pieces of information:

```
wbtags
id
parameters
object
upper(id)=upper("$MI_WEBTAGSID")
```

The **installGlobalTagCache** utility is located in the **$INFORMIXDIR/extend/web.version/install** directory. The following example shows how to execute the utility:

```
installGlobalTagCache db tagtbl name param body tagselect
```

db is the name of the database.

tagtbl is the name of the table that contains the tags.

name is the name of the tag table column that contains the tag names.

param is the name of the tag table column that contains the tag parameters.

body is the name of the tag table column that contains the tag body.

tagselect is the WHERE clause used to select the tag parameters and body.

Clearing the Global Tag Cache

To clear the global tag cache, execute the **WebClearTagCache()** procedure, as shown in the following example:

```
execute procedure WebClearTagCache();
```

The procedure does not take any arguments. Execute this procedure if there is a failure in updating the global tag cache or if you drop either of the system tables that stores information on dynamic tags or UDR tags.

Debugging the Global Tag Cache

To dump the contents of the global tag cache to a file, execute the routine **WebDumpTagCache()**. The routine takes two arguments:

- The name of the file on the client computer
- An integer, either 0 or 1

  If you want a full dump of the information, specify 0. If you want an abbreviated dump of the information, specify 1.

Using AppPage Caching

If you decide you want to use AppPage caching for one or more AppPages in your application, you must:

1. Set AppPage caching for your Webdriver configuration by setting certain Webdriver variables.

   For detailed information on this step, refer to "Setting AppPage Caching for a Webdriver Configuration," next.

2. Enable AppPage caching for a particular AppPage.

   For detailed information on this step, refer to "Enabling AppPage Caching for a Particular AppPage" on page 9-13.

Setting AppPage Caching for a Webdriver Configuration

To set AppPage caching for your Webdriver configuration, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table. For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

Webdriver Variable	Mandatory?	Description
cache_page	Yes	Specifies whether AppPage caching is enabled or disabled Set to `ON` to enable AppPage caching and `OFF` to disable AppPage caching. The default value is `OFF`.
cache_directory	Yes	Specifies the full pathname of the directory on the Web server computer in which cached AppPages and large objects are placed If this variable is not set, neither AppPages nor large objects are cached.
cache_page_buckets	No	Specifies the number of subdirectories per AppPage created under the directory specified by **cache_directory** The default is one subdirectory per AppPage. Set this variable only if you intend on caching AppPages that might have over 1000 different versions.
cache_page_life	No	Specifies the length of time after which an AppPage is refreshed from the database Set **cache_page_life** in units of seconds (`s` or `S`), hours (`h` or `H`), or days (`d` or `D`). For example, the value `5d` indicates five days.
cache_admin	No	Specifies the name of the Cache Administration AppPage The Cache Administration AppPage is not stored in the database, but is an internal AppPage managed by Webdriver. When **MIval** is set to this value, Webdriver invokes this AppPage so you can add, delete, purge, or view cache entries in the **cache_directory** directory. The default value is `cacheadmin`.

(1 of 2)

Webdriver Variable	Mandatory?	Description
cache_admin_password	No	Specifies that cache administration requests are processed only if the password entered in the Cache Administration AppPage matches this value.
cache_page_timestamp	No	Specifies that Webdriver, when invoking an AppPage for which AppPage caching has been enabled, adds timestamp information at the bottom of the page. The timestamp is enclosed in an HTML comment and thus is only seen if a user views the HTML source of the AppPage in their browser. The default value is `OFF`. To enable this feature, set this Webdriver variable to `ON`.
cache_page_debug	No	Specifies that Webdriver invokes AppPages that contain deferred sections (delimited with the MIDEFERRED tag) without returning an error, even if AppPage caching has *not* been enabled. This Webdriver variable is used to debug problems with partial AppPage caching. The **cache_page_debug** Webdriver variable can be set to two values: `show_defer` and `execute_defer`. When set to `show_defer` and you invoke an AppPage with a deferred section, Webdriver returns the deferred section in its original form. If the Webdriver variable is set to `execute_defer`, Webdriver executes the deferred section when you invoke the AppPage. For detailed information on the **cache_page_debug** Webdriver variable, refer to "Debugging Problems with Partial AppPage Caching" on page 9-27.

(2 of 2)

To enable AppPage caching for a particular AppPage, refer to "Enabling AppPage Caching for a Particular AppPage" on page 9-13.

When you set the **cache_directory** Webdriver variable to a directory in which cached large objects and AppPages are placed, and you set **cache_page** to `ON`, Webdriver places an AppPage for which caching has been enabled in its disk cache the first time the AppPage is retrieved. Subsequent retrievals of that AppPage are made from Webdriver's disk cache.

Set the **cache_page_life** Webdriver variable to the length of time you want AppPages to remain in the disk cache. Specify **cache_page_life** in units of seconds (`s` or `S`), hours (`h` or `H`), or days (`d` or `D`). Each time an AppPage is retrieved, if the time stamp indicates that the AppPage is older than the value of **cache_page_life**, the AppPage is refreshed from the database.

If you intend to cache AppPages that might have more than 1000 different versions, you should set the **cache_page_buckets** Webdriver variable to create subdirectories under the initial AppPage directory. An AppPage might have more than 1000 versions if you pass a large number of variables to it.

Important: *If you modify the setting for **cache_page_buckets**, the algorithm used to locate the different versions of the AppPage in the subdirectories changes. Therefore, you should remove all AppPages from the subdirectories if you change the value for **cache_page_buckets**. The algorithm for creating the subdirectories does not always create them in sequential order.*

Example of Setting AppPage Caching for a Webdriver Configuration

The following table shows sample settings for the AppPage caching Webdriver variables to enable AppPage caching for a Webdriver configuration.

Webdriver Variable	Sample Value
cache_page	`ON`
cache_directory	`/bigdisk/AppPageCache`
cache_page_buckets	`30`
cache_page_life	`1h`
cache_admin	`cacheadmin`
cache_admin_password	`topsecret`

AppPage caching is enabled for this configuration because the **cache_page** variable is set to `ON`. The directory that holds the cached AppPages is **/bigdisk/AppPageCache**. This directory has a maximum of 30 subdirectories for the AppPage. A particular AppPage remains cached for a maximum of one hour. The Cache Administration AppPage is called **cacheadmin**, and the password to use this page to administer AppPage caching is `topsecret`.

Use the Web DataBlade Module Administration Tool to set these Webdriver variables for your Web DataBlade module configuration. For more information on setting Webdriver variables, refer to Chapter 3, "Configuring Webdriver."

The Cache Administration AppPage

The **cache_admin** Webdriver variable specifies the name of the Cache Administration AppPage. If you do not set this Webdriver variable in your Webdriver configuration, the name of the Cache Administration AppPage is **cacheadmin**. Use the Cache Administration AppPage to enable and disable AppPage caching for a particular AppPage, as described later on in this section. You also use the Cache Administration AppPage to purge AppPages from the cache, view the AppPages for which you have enabled AppPage caching, and analyze AppPage caching statistics.

The Cache Administration AppPage is not stored in a database table, but rather is a "virtual" AppPage dynamically created and managed by Webdriver. However, you invoke the AppPage the same way you invoke all other AppPages.

The following example shows how to invoke the Cache Administration AppPage using its default name (**cacheadmin**):

```
http://domain:port/sales/?MIval=cacheadmin
```

In the example, *domain* refers to the name of your Web server computer, *port* refers to the port number of your Web server process, and `/sales` is a URL prefix that corresponds to a Webdriver mapping.

If you have previously set the **cache_admin** Webdriver variable to another value in your Webdriver configuration, then specify that value instead of `cacheadmin`.

Figure 9-1 shows the Cache Administration AppPage.

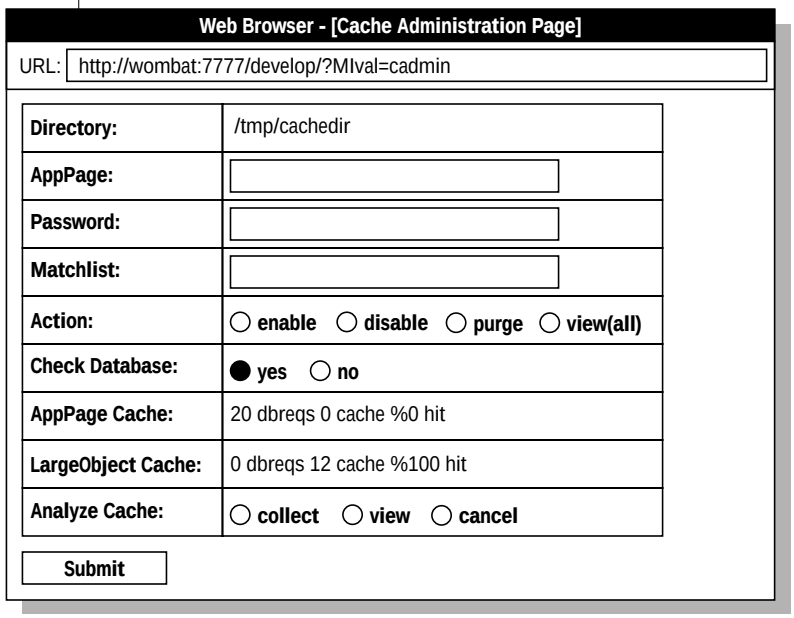


Figure 9-1 *Cache Administration AppPage*

The figure shows that the AppPage cache directory has been set to **/tmp/cachedir**.

The following sections describe how to use the Cache Administration AppPage to perform cache administration tasks.

Enabling AppPage Caching for a Particular AppPage

This section describes how to enable AppPage caching for a particular AppPage.

The section is written with the assumption that you have already set the necessary Webdriver variables to enable AppPage caching for your Webdriver configuration, as described in "Setting AppPage Caching for a Webdriver Configuration" on page 9-9.

To enable AppPage caching for an individual AppPage

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.
2. Enter the name of the AppPage for which you want to enable AppPage caching in the **AppPage** text box.
3. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.
4. Select **enable** from the **Action** group.
5. Select **yes** from the **Check Database** group if you want Webdriver to make sure that the AppPage for which you are enabling caching exists in the database.

   If you select **no**, Webdriver does not verify that the AppPage actually exists.
6. Click **Submit**.

Disabling AppPage Caching for a Particular AppPage

This section describes how to disable AppPage caching for an AppPage for which caching is currently enabled.

To disable AppPage caching for an AppPage

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.
2. Enter the name of the AppPage for which you want to disable AppPage caching in the **AppPage** text box.

3. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.

4. Select **disable** from the **Action** group.

5. Select **yes** from the **Check Database** group if you want Webdriver to make sure that the AppPage for which you are disabling caching exists in the database.

   If you select **no**, Webdriver does not verify that the AppPage actually exists.

6. Click **Submit**.

Removing AppPages from the Disk Cache

This section describes two options for removing AppPages from the disk cache:

- Removing all versions of an AppPage
- Removing a particular version of an AppPage

When you remove an AppPage from the disk cache, AppPage caching is still enabled for the AppPage, and the next request for the particular AppPage creates a new cached AppPage.

Webdriver caches AppPages based on the values of the variables passed into the AppPage. This means that if an AppPage is invoked many times with different values for the variables, Webdriver caches many different versions of the AppPage in its disk cache. When you remove AppPages from the disk cache, you can remove all versions or just a particular version, as described in the following two sections.

Removing All Versions of an AppPage from the Disk Cache

If you change a section of an AppPage that is common to all versions of the cached AppPage, then you should remove all versions of the AppPage from the disk cache so users start to invoke the newly updated AppPage.

To remove all versions of an AppPage from the disk cache

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.
2. Enter the name of the AppPage for which you want remove all versions from the disk cache in the **AppPage** text box.
3. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.
4. Select **purge** from the **Action** group.
5. Click **Submit**.

Removing a Particular Version of an AppPage from the Disk Cache

If only a particular version of a cached AppPage has changed, you only need to remove that version of the AppPage from the disk cache; you do not need to remove all versions.

For example, assume you pass a product ID into an AppPage and thus the versions of the cached AppPages are based on the product ID. If the information for only one of the products has changed, you do not need to remove the cached AppPages that correspond to the unchanged products.

To remove a particular version of an AppPage from the disk cache

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.

2. Enter the name of the AppPage for which want remove all versions from the disk cache in the **AppPage** text box.

3. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.

4. Enter the name / value pair that specifies the particular version of the AppPage in the **Matchlist** text box.

   For example, if you want to remove the cached version of an AppPage identified by the name / value pair `product_id=6`, enter this name value pair in the **Matchlist** text box.

5. Select **purge** from the **Action** group.

6. Click **Submit**.

Viewing the List of AppPages That Have Caching Enabled

This section describes how to view the list of AppPages that have AppPage caching enabled.

To view a list of AppPages that have AppPage caching enabled

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.

2. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.

3. Select **view(all)** from the **Action** group.

4. Click **Submit**.

Caching AppPages Retrieved with the POST Method

By default, Webdriver never caches AppPages that are retrieved with the POST method, even if you have explicitly enabled AppPage caching for that AppPage.

You can, however, bypass this default behavior by setting a hidden variable in the form, as described in the following procedure.

To cache an AppPage that is retrieved with the POST method

1. In your definition of the form that specifies the POST method, add a hidden variable called **MIdriver** and set its value to `allow_cache`, as shown in the following example:

   ```
   <INPUT TYPE=hidden NAME=MIdriver VALUE=allow_cache>
   ```

2. Prefix all other variables in the form with the keyword **defer.** This specifies that the variables are actually deferred variables.

3. In the AppPage that is retrieved with the POST method, be sure that all variables passed to it from the form are referenced in a deferred section. You specify a deferred section in an AppPage with the MIDEFERRED tag.

If you follow this procedure, the nondeferred section of the AppPage retrieved with the POST method can be cached.

For detailed information on specifying deferred sections in an AppPage with the MIDEFERRED tag, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Using the MIFUNC Tag to Dynamically Manage AppPage Caching from Within an AppPage

The section "Using AppPage Caching" on page 9-8 describes how to manage AppPage caching by using the Cache Administration AppPage. This section provides an example, consisting of three AppPages, of using the MIFUNC AppPage tag to manage AppPage caching dynamically from within an AppPage.

Typically you use the MIFUNC AppPage tag to execute user-defined NSAPI or ISAPI functions in your AppPages. However, by specifying the `INTERNAL=cache_admin` attribute, you can also execute a select list of internal Webdriver functions in your AppPages to help you manage AppPage caching dynamically.

For a detailed description of using the INTERNAL attribute with the MIFUNC AppPage tag and for a full list of available options, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*. The description of the following example assumes that you are familiar with the MIFUNC discussion in the *IBM Informix Web DataBlade Module Application Developer's Guide*.

AppPage 1: Setting Up the Example

The following **/c_setup.html** AppPage sets up the example:

```
<?MIFUNC INTERNAL=cache_admin cache_mode=enable app_page=c_displ
  message=display>
<?MIVAR>$cache_admin.display,<?/MIVAR>
<?MIERROR TAG=MIVAR>$cache_admin.display, Cache Table exists<?/MIERROR>
<?MISQL SQL="create table ctesttab (ticker_symbol varchar(7),
  name varchar(255), price float);">
<?/MISQL>
<?MISQL
SQL="insert into ctesttab values ('IFMX','Informix Corp',97.5);
insert into ctesttab values ('IBM','International Business Machines', 10);
insert into ctesttab values ('PSFT','Peoplesoft',12);">
<?/MISQL>Cache table created
<?/MIFUNC>
```

This AppPage first enables AppPage caching for the **/c_displ.html** AppPage (described later on in this section), creates the **ctesttab** table, and inserts sample data into the table.

AppPage 2: Displaying Information

The following **/c_displ.html** AppPage shows how to display information from the **ctesttab** table based on the item you select from the list box:

```
<?MIBLOCK COND=$(XST,$symbol)>
<?MISQL SQL="select * from ctesttab where ticker_symbol='$symbol';">$1, $2,
$3<?/MISQL>
<?MIELSE>
<form method="post" action=<?MIVAR>$WEB_HOME<?/MIVAR>>
<td valign=top align=middle>
<input type="hidden" name=MIval value=/c_displ.html>
<input type="hidden" name=MIdriver value=allow_cache>
<?MIVAR NAME=symbol>IFMX<?/MIVAR>
<TD><STRONG>Enter symbol to display</STRONG></TD>
<TD>
<?APB2_SELECT_LIST QRY="select ticker_symbol from ctesttab;"
  NAMEVAL="symbol" DEFVAL=$(REPLACE,$symbol,',')>
</TD>
</td>
<INPUT TYPE="SUBMIT" VALUE="SUBMIT">

<?MIVAR>
<BR><A HREF=$WEB_HOME?MIval=/c_update.html>Update a stock</A>
<?/MIVAR>

</form>
<?/MIBLOCK>
```

The AppPage calls itself when you click the **Submit** button.

Remember that the **/c_setup.html** AppPage enables AppPage caching for the **/c_displ.html** AppPage. This means that the first time you invoke the **/c_displ.html** AppPage, Webdriver retrieves the AppPage from the database. However, subsequent times that you invoke the AppPage using the same item from the list box, Webdriver uses the cached AppPage rather than retrieving it from the database.

The **MIdriver** hidden variable in the **/c_displ.html** AppPage enables AppPage caching even though it recursively retrieves itself with the POST method. For more information on the **MIdriver** hidden variable, refer to "Caching AppPages Retrieved with the POST Method" on page 9-18.

AppPage 3: Updating Information

Finally, the following **/c_update.html** AppPage, invoked from the **/c_displ.html** AppPage, shows how to update a value in the database table:

```
<?MIBLOCK COND=$(AND,$(XST,$new_value),$(XST,$symbol))>
<?MISQL SQL="update ctesttab set price=$new_value where
ticker_symbol='$symbol';"><?/MISQL>
<?MIVAR NAME=match>symbol=$symbol<?/MIVAR>
<?MIFUNC INTERNAL=cache_admin cache_mode=purge matchlist=$match
  message=display app_page=c_displ>
<?MIVAR>$cache_admin.display<?/MIVAR><?/MIFUNC>
<?/MIBLOCK>

<form method="post" action=<?MIVAR>$WEB_HOME<?/MIVAR>
<?MIVAR NAME=symbol>IFMX<?/MIVAR>
<TD><STRONG>Enter symbol to update</STRONG></TD>
<TD>
<?APB2_SELECT_LIST QRY="select ticker_symbol from ctesttab;"
  NAMEVAL="symbol" DEFVAL=$(REPLACE,$symbol,','')>
</TD>

<td valign=top align=middle>
<input type="hidden" name=MIval value=/c_update.html>
new value
<input type="text" name="new_value" size=4>
</td>
<INPUT TYPE="SUBMIT" VALUE="SUBMIT">
<PRE><?MIVAR>
<BR><A HREF=$WEB_HOME?MIval=/c_displ.html>Display stock</A>
<?/MIVAR>
</form>
```

The AppPage calls itself when you click the **Submit** button.

The **/c_update.html** AppPage first determines whether you have already used the **/c_update.html** AppPage to update a row in the **ctesttab** table. If you have, the AppPage uses the MIFUNC AppPage tag to purge the relevant cached **/c_displ.html** AppPage instance from the disk cache. This means that the next time you invoke the **/c_displ.html** AppPage by specifying the item in the list box that you updated in the **ctesttab** table, Webdriver retrieves the AppPage from the database. This is because a cached entry for the AppPage does not exist in the cache directory because it was purged with the MIFUNC tag. This is correct behavior since the data on a cached **/c_displ.html** AppPage would be out of date, and Webdriver *must* go to the database to retrieve the latest data.

Analyzing AppPage Caching

You can use the Cache Administration AppPage to analyze the effectiveness of your AppPage caching strategy. Caching statistics compare the number of times an AppPage has been retrieved from the database against the number of times an AppPage has been read from the disk cache.

You can either analyze statistics for all AppPages since the Web server was started or analyze statistics for a particular AppPage.

Analyzing Caching for All AppPages

The text box labeled **AppPage Cache** on the Cache Administration AppPage displays the following three statistics (since the Web server was started):

- **dbreqs**, the number of times Webdriver retrieved any AppPage from the database
- **cache**, the number of times Webdriver read any AppPage from the disk cache
- **hit**, Webdriver's hit rate of reading AppPages from disk, displayed as a percentage

A higher hit rate means that Webdriver is reading more AppPages from the disk cache, which typically translates into higher performance.

The text box labeled **Large Object Cache** shows similar caching statistics for cached large objects, such as images and video clips.

Analyzing Caching for a Particular AppPage

If you are interested in analyzing the number of times Webdriver read a particular AppPage from disk rather than the number of times Webdriver retrieved it from the database, then you can collect caching statistics for just that AppPage.

Enabling Collection of Caching Statistics for an AppPage

Although you can enable the collection of caching statistics for AppPages for which you have not currently enabled AppPage caching, the results are not very interesting since Webdriver *always* retrieves these AppPage from the database. For this reason it makes sense to only enable the collection of caching statistics for AppPages for which you have enabled AppPage caching.

To enable the collection of caching statistics for an AppPage

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.

2. Enter the name of the AppPage for which you want to enable the collection of caching statistics.

3. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.

4. Select **collect** in the **Analyze Cache** group.

5. Click **Submit**.

Viewing Caching Statistics for an AppPage

This section describes how to view caching statistics for all AppPage for which you have enabled the collection of caching statistics.

To view caching statistics for an AppPage

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.

2. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.

3. Select **view** from the **Analyze Cache** group.

4. Click **Submit**.

   The following caching statistics are displayed for each AppPage for which you have enabled the collection of statistics:

   - **Cache (READ)**, the number of times Webdriver has read the AppPage from the disk cache
   - **Cache (WRITE)**, the number of times Webdriver has written the AppPage to the disk cache
   - **WebExplode()**, the number of times Webdriver has retrieved the AppPage from the database

Disabling Collection of Caching Statistics for all AppPages

You can disable the collection of caching statistics only for *all* AppPages which are currently enabled to collect statistics; you cannot cancel caching statistics for a single AppPage.

To cancel caching statistics for all AppPages

1. Invoke in your browser the Cache Administration AppPage identified by the **cache_admin** Webdriver variable.

   For information on the Cache Administration AppPage and details on how to invoke it in your browser, refer to "The Cache Administration AppPage" on page 9-12.

2. Enter the password specified by the **cache_admin_password** Webdriver variable in the **Password** text box.

   If you have not set the **cache_admin_password** Webdriver variable for your Webdriver configuration, leave the **Password** text box blank.

3. Select **cancel** from the **Analyze Cache** group.

4. Click **Submit**.

Partial AppPage Caching

Some AppPages are not good candidates for AppPage caching because, although most of the contents of the AppPage are static, there is a small amount of dynamic content that should not be cached. The dynamic contents of these types of AppPages must be executed by the **WebExplode()** function each time the AppPage is requested. These AppPages are good candidates for partial AppPage caching.

To enable partial AppPage caching, first set the standard AppPage caching Webdriver variables as described in the section "Setting AppPage Caching for a Webdriver Configuration" on page 9-9. Then use the MIDEFERRED tag in the AppPage to identify the dynamic content.

An example of dynamic content is a reference to a session variable. AppPages in which most of the content is static except for a reference to a session variable can be partially cached as long as the reference to the session variable is enclosed within MIDEFERRED tags.

For detailed information on using the MIDEFERRED tag, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

How Partial AppPage Caching Works

When the **WebExplode()** function first processes an AppPage that contains an MIDEFERRED tag, it processes everything *but* the content inside the MIDEFERRED tag and tells Webdriver that further processing of the deferred section is needed.

When it receives this partially processed AppPage, Webdriver caches it in the cache directory, adding the extension **.def** to the file to indicate it contains deferred content. Webdriver then sends the cached page back to the **WebExplode()** function to process just the deferred content. Subsequent requests for the AppPage are also passed back to the **WebExplode()** function, although only the deferred section is processed, not the entire AppPage.

Using Variables with the MIDEFERRED Tag

During AppPage caching, Webdriver uses variables sent to an AppPage as a key to create the name of the file stored in the cache directory. Static AppPages always have variables set to the same value, so the cached AppPage is always easily found.

Variables in the dynamic content of the MIDEFERRED tag, however, can change each time the AppPage is called. These types of variables, therefore, should not be part of the key used to create and find files in the AppPage cache. To specify to Webdriver that a variable should not be used in the key, prefix the variable with the **defer** keyword.

Important: *If you have enabled session management and your AppPage contains a session variable (a variable that is prefixed with the* ***session*** *keyword) in the nondeferred section of the AppPage, the AppPage is never cached. A warning is written to the Webdriver log instead.*

Debugging Problems with Partial AppPage Caching

When you are developing an AppPage that contains a deferred section, it is often useful, for debugging purposes, to be able to invoke the AppPage in a browser and see what Webdriver returns in the deferred section without actually enabling AppPage caching for the AppPage. If you actually enable caching for the AppPage while you are developing the AppPage, changes to the AppPage are not reflected. By default, however, Webdriver returns an error if you try to invoke an AppPage that contains a deferred section without having previously enabled AppPage caching for the AppPage.

To work around this behavior, set the **cache_page_debug** Webdriver variable in your Webdriver configuration. The Webdriver variable can be set to two values: `show_defer` and `execute_defer`.

Set the **cache_page_debug** Webdriver variable to `show_defer` if you want Webdriver to return the AppPage with the deferred section in its original form. This means that in the returned AppPage, the **WebExplode()** function has not executed the section of the AppPage between the MIDEFERRED tags.

Set the **cache_page_debug** Webdriver variable to `execute_defer` if you want the **WebExplode()** function to execute the section between the MIDEFERRED tags. In other words, you want to see the AppPage as if AppPage caching has been enabled for the AppPage.

Large Object Caching

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.

Setting Large Object Caching

To set large object caching, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table.

Webdriver Variable	Mandatory?	Description
cache_directory	Yes	Specifies the directory on the Web server computer in which cached large objects are placed. If not set, large objects are not cached.
cache_buckets	No	Specifies the number of subdirectories per database created under the directory specified by **cache_directory**. The default is one subdirectory per database.
cache_maxsize	No	Specifies the maximum size in bytes of large objects to be cached. The default is `64 KB`.

For detailed information on using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

When you set the **cache_directory** Webdriver 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 **cache_buckets** Webdriver variable, one or more subdirectories of **cache_directory** 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 **cache_directory** directory by specifying a higher number for **cache_buckets**. The subdirectories created are named ***database_name***0, ***database_name***1, 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* ***cache_buckets****, 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* ***cache_buckets****. The algorithm for creating the database subdirectories does not always create them in sequential order.*

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 might be named as follows:

```
/bigdisk/LOcache/webdb0/na6b7c8d9m2m2l14k2g9q1p686f626a65637
   44g18l4m1ft48h9bde74ga2b9dia2begeg
```

Example of Setting Large Object Caching

The following table shows an example of enabling large object caching with sample Webdriver variable settings.

Webdriver Variable	Sample Value
cache_directory	`/bigdisk/LOCache`
cache_buckets	`30`
cache_maxsize	`1024000`

Large object caching is enabled for this configuration because the **cache_directory** Webdriver variable is set to a value, `/bigdisk/LOCache`. This is the directory that holds the cached large objects. The directory has a maximum of 30 subdirectories for the database. The maximum size of large objects that can be stored in the large object cache, specified by the **cache_maxsize** Webdriver variable, is 1 MB.

Use the Web DataBlade Module Administration Tool to set these Webdriver variables for your Web DataBlade module configuration. For more information on setting Webdriver variables, refer to Chapter 3, "Configuring Webdriver."

Analyzing Caching Statistics for Large Objects

You can use the Cache Administration AppPage to analyze the effectiveness of your large object caching strategy. Caching statistics compare the number of times a large object has been retrieved from the database against the number of times a large object has been read from the disk cache.

Refer to "The Cache Administration AppPage" on page 9-12 for instructions on how to invoke the Cache Administration AppPage in your browser.

The text box labeled **LargeObject Cache** on the Cache Administration AppPage displays the following three statistics since the Web server was started:

- **dbreqs**, the number of times Webdriver retrieved any large object from the database
- **cache**, the number of times Webdriver read any large object from the disk cache
- **hit**, Webdriver's hit rate of reading large objects from disk, displayed as a percentage

A higher hit rate means that Webdriver is reading more large objects from the disk cache, which typically translates into higher performance.

Using Session Variables to Improve Performance

You can improve the performance of your application by using session variables. Session variables are variables that can be accessed on any AppPage without having to explicitly pass them to an AppPage. Session variables can reduce the number of database queries and thereby improve the overall performance of your application.

For example, a shopping cart application could use session variables to improve performance. Assume that an AppPage initially uses the MISQL tag to retrieve product information from a database table. A user of the application might then choose one of the products to buy and add it to their shopping cart. The application stores the product, as well as all the relevant information about the product, in session variables. The user might not immediately buy the product, but continue shopping, and thereby invoke more AppPages. Once the user wants to buy the product, the application does not need to retrieve the product information from the database once again, since all the relevant information is stored in the session variables.

For detailed information on how to enable session management and how to use session variables, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Session Management and AppPage Caching

This section describes an additional application development step you must perform to ensure that AppPages are cached correctly if you have enabled both session management and AppPage caching. In cached AppPages that also use session management, you must replace every instance of the **WEB_HOME** anchor variable with the system dynamic tag of the same name, WEB_HOME**.**

As described in the section "AppPages That Are Not Cached" on page 9-4, AppPages that reference a session variable in a static, nondeferred section are never cached.

AppPages in which most of the content is static except for a reference to a session variable can be partially cached as long as the reference to the session variable is enclosed within MIDEFERRED tags. Refer to "Partial AppPage Caching" on page 9-25 for information on partial AppPage caching.

If your AppPages contain only static content or the reference to the session variable occurs in a deferred section of the AppPage delimited by the MIDEFERRED tags, then the AppPage can be cached. However, you must replace every instance of the **WEB_HOME** anchor variable with the system dynamic tag of the same name, WEB_HOME.

For example, assume you use the ANCHOR tag to invoke APB in an AppPage in the following way:

```
<A HREF=<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=/APB20/apb.html>APB</A>
```

To ensure that the AppPage is cached when appropriate, replace <?MIVAR>$WEB_HOME<?/MIVAR> with the **WEB_HOME** system dynamic tag, as shown in the following example:

```
<A HREF=<?WEB_HOME>?MIval=/APB20/apb.html>APB</A>
```

Chapter

10

Globalizing Your Web DataBlade Module Application

In This Chapter

The following sections describe multibyte support:

Overview of Globalization

While most Western languages require one byte per alphabetic character, many other languages are represented by more complicated characters, requiring more than one byte. These are called *multibyte character sets*. Support of multibyte character sets, as well as single-byte character sets, is known as *multibyte support*.

The IBM Informix Web DataBlade module includes support for multibyte character sets, which contain more than 256 characters. This support enables you to write and execute AppPages and database content in a multibyte format, where a single character might take more than one byte of storage space.

Using Locale Variables

When a database application runs in a client/server environment, the client application and database server might reside on different computers. These computers might have different language support. To ensure that these parts of the database application communicate locale information, you must set the **CLIENT_LOCALE** and **DB_LOCALE** Informix environment variables.

Locale variables identify the language, territory, and code set. The default locale variables are set to English.

To set the **CLIENT_LOCALE** or **DB_LOCALE** Informix environment variable, add a line to Setvar section of the **web.cnf** file. The following example shows how to specify Shift JIS or EUC as the client locale:

```
<Setvar>
CLIENT_LOCALE ja_jp.sjis-s
</Setvar>

<Setvar>
CLIENT_LOCALE ja_jp.ujis
</Setvar>
```

The **DB_LOCALE** Informix environment variables must be set to specify a nondefault locale. By setting the **DB_LOCALE** variable, any database you create with your Web application will have the corresponding database locale. Be sure that the database locale is compatible with the client locale.

For more information on the **CLIENT_LOCALE** and **DB_LOCALE** Informix environment variables, see the *IBM Informix Guide to GLS Functionality*.

AppPage Builder and Globalization

Your browser needs to know when a code set is being used. For the AppPage Builder (APB) application, use the **MI_WEBENCODE** user-defined variable for your Webdriver configuration to supply this information. The variable is then embedded in the APB_HEADER and APB_ERROR user dynamic tags. The following example shows how the **MI_WEBENCODE** variable appears within these dynamic tags.

```
<?MIVAR COND=$(XST,$MI_WEBENCODE)>$(URLDECODE,$MI_WEBENCODE)<?/MIVAR>
```

To set the **MI_WEBENCODE** user-defined variable, use the Web DataBlade Module Administration Tool. Specify as its value the same code set used in the **CLIENT_LOCALE** Informix environment variable.

The **MI_WEBENCODE** user-defined variable does not appear on the Web DataBlade Module Administration Tool variables list. You must manually add this variable. See "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29 for information about setting variables with the Web DataBlade Module Administration Tool.

WebURLDecode() and WebURLEncode() Functions

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

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

Output from the **WebURLDecode()** and **WebURLEncode()** functions varies depending on the code set used in the **CLIENT_LOCALE** Informix environment variable, as follows:

- **WebURLDecode().** This function expects the URL-encoded string passed as the argument to consist of the characters encoded in **CLIENT_LOCALE** variable.

- **WebURLEncode().** This function expects the characters in the **CLIENT_LOCALE** variable and URL-encodes them.

For more information on the **WebURLDecode()** and **WebURLEncode()** functions, see the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Chapter

11

Deploying Web DataBlade Module Applications

In This Chapter

This chapter provides information about moving your Web DataBlade module application from a development environment to a production environment. It includes the following topics:

Overview of Deployment

Once you have finished developing and testing your Web DataBlade module application, you are ready to deploy it and use it in a production environment. Deployment of a Web DataBlade module application typically includes the following tasks:

- Moving the Web DataBlade module application from a development database to a production database
- Using a production Web server on a different computer from the computer on which the Informix database server is installed

The following two sections describe the issues you should be aware of when you perform the two tasks.

Moving Applications from Development to Production

Web DataBlade module applications are typically made up of AppPages stored in one or more tables, Web DataBlade module information stored in system tables, and application data stored in one or more user tables.

You can either move each type of data separately from the development database to the production database, or you can move the entire contents of the development database to the production database.

Moving Each Type of Data Separately

When you move each type of data separately, you control what data exists in the production database at any one time. If you move *all* the data in the database, you might move temporary tables you created during development, out-of-date data, and other data you probably do not want to include in your production database. Moving each type of data separately, however, is more complex because you must execute a separate command for each table.

Before you begin moving data, be sure you register the Web DataBlade module and install the Web DataBlade Module Administration Tool in the production database.

You should probably *not* install AppPage Builder (APB) in the production database, since APB is typically used only during development and can pose a security risk if present in a production database.

You might need to move the following three types of data from a development database to a production database:

- The AppPages and large objects that make up your Web application

  If you used APB to create your Web application, AppPages are stored in the **wbPages** table and large objects are stored in the **wbBinaries** table.

  If you added a new extension type to the **wbExtensions** table, and then created additional tables to store objects with the new extension, be sure you move the data in these new tables as well.

- The contents of the **wbExtensions** table

  For detailed information about the schema of the **wbExtensions** table, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

- Web DataBlade module data stored in the following system tables:
  - **WebConfigs**
  - **WebCMImages**
  - **WebEnvVariables**
  - **WebTags** or **wbTags**, whichever you use to store your dynamic tags
  - **WebUdrs**

  For detailed information about the schemas of these system tables, refer to Appendix A, "Web DataBlade Module System Tables."

- Application data stored in user tables

Use the standard SQL statements UNLOAD and RELOAD to move the different types of data from your development database to your production database.

For more information on the UNLOAD and RELOAD statements, refer to the *IBM Informix Guide to SQL: Syntax*.

Moving Data All at Once

To move the entire contents of the development database to the production database, you need execute only one command: **onpload**. However, *all* data is moved, including temporary or out-of-date data.

The **onpload** utility is part of the High-Performance Loader (HPL) feature of the Informix database server. The HPL allows you to efficiently unload and reload large quantities of data to or from an Informix database. For more information on using the HPL, refer to the *Guide to the High-Performance Loader* for your database server.

Accessing the New Production Database

Once you have moved all the relevant data from the development database to the production database, you are ready to access the Web DataBlade module application that now resides in the production database. You can either create new Webdriver mappings to access the production database or update the old Webdriver mappings. These methods are described in the next two sections.

Creating New Webdriver Mappings

This section describes how to create new Webdriver mappings to access the production database.

To create new Webdriver mappings

1. Configure the Web DataBlade Module Administration Tool for the production database.

   For detailed information on how to do this, refer to the section "Setting Up the Web DataBlade Module Administration Tool" on page 3-22.

2. Use the Web DataBlade Module Administration Tool to add Webdriver mappings.

   The tool should automatically bring up the Webdriver configurations that were moved from the development database to the production database.

   For detailed information on adding Webdriver mappings, refer to "Adding a New Webdriver Mapping" on page 3-45.

3. Add corresponding URL prefix information to your Web server. Be sure the URL prefix is the same as the name of the Webdriver mapping.

   For details on how to perform this step for the NSAPI, Apache, and ISAPI implementations of Webdriver, refer to Chapter 4, "Using the NSAPI Webdriver," Chapter 5, "Using the Apache Webdriver," and Chapter 6, "Using the ISAPI Webdriver," respectively.

Updating Existing Webdriver Mappings

This section describes how to update the old Webdriver mappings to access the production database.

To update old Webdriver mappings

1. Locate the Webdriver configuration file, typically called **web.cnf**. The **MI_WEBCONFIG** environment variable, usually set in the Web server startup files, points to this file.
2. For each Webdriver mapping you want to change, update the value of the **database** variable in the Map section of the **web.cnf** file from the development database to the production database.
3. Restart your Web server.
4. Invoke the Web DataBlade module application in the production database using the same URL prefix in your browser that you used to invoke the application in the development database.

If you want Webdriver to connect to the production database as a different user than is currently assigned to the Webdriver mapping, use the Web DataBlade Module Administration Tool to change the user name and password for the Webdriver mapping. For detailed information on using the tool, refer to "Invoking and Using the Web DataBlade Module Administration Tool" on page 3-29.

Using a Web Server on a Different Computer

While you are developing a Web DataBlade module application, the Web server is typically on the same computer as the Informix database server. However, when you deploy the application to a production environment, you might want to use an existing production Web server that resides on a different computer from the one on which the Informix database server is installed. This section describes what you need to do so that the Web DataBlade module application continues to work correctly in this type of production environment.

A particular Web server can point to only one version of the Web DataBlade module. This means that if you want to register two different versions of the Web DataBlade module into two different databases in the same Informix database server, each database must be accessed by *different* Web servers.

Important: *You can register only one version of the Web DataBlade module in a particular database.*

To deploy a Web DataBlade module application on a computer different from the computer on which the Web server resides

1. Install IBM Informix Connect or IBM Informix Client Software Developer's Kit on the Web server machine.

   IBM Informix Connect consists of the runtime versions of Informix client products. IBM Informix Client Software Developer's Kit consists of the development versions of the client products.

   For detailed information on installing either product, refer to the *IBM Informix Client Products Installation Guide* for the platform on which your Web server is installed.

2. Set up the network connection between the Web server computer and the computer on which the Informix database server is installed by updating the **sqlhosts** file on the Web server computer.

   For detailed information on setting up a network connection, refer to the *Administrator's Guide* for your database server.

3. Move the Webdriver configuration file, typically called **web.cnf**, to the Web server computer.

4. Set the **MI_WEBCONFIG** environment variable to point to the full pathname of the **web.cnf** file.

5. If you are using the CGI Webdriver, move the executable to the directory on the Web server computer set up to contain CGI programs.

   You should also put the **web.cnf** file in the same directory.

6. If you are using the NSAPI, the Apache, or the ISAPI Webdriver, for instructions on how to set up the particular Webdriver on the computer that runs the Web server, refer to Chapter 4, Chapter 5, or Chapter 6, respectively.

Chapter

12

Debugging and Troubleshooting

In This Chapter

This chapter describes how to enable Webdriver tracing to troubleshoot your Web DataBlade module applications. For more information on debugging applications during the development stage, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Enabling Webdriver Tracing

To enable Webdriver tracing for all Webdriver configurations defined for a Web DataBlade module installation, set the **debug_file** and **debug_level** Webdriver variables in the Global section of the **web.cnf** file. For information on updating the **web.cnf** file, refer to "The Webdriver Configuration File (web.cnf)" on page 3-9.

The following table describes each variable.

Variable	Description
debug_level	Enables Webdriver tracing to the log file specified by the **debug_file** variable Refer to the table on page 12-4 for a full list of possible values for this variable.
debug_file	Specifies the full pathname of the log file to which Webdriver messages are written

You can also set the **debug_level** Webdriver variable for a particular Webdriver configuration using the Web DataBlade Module Administration Tool. The value of the **debug_level** Webdriver variable, if set for your Webdriver configuration, overrides the value of the variable in the **web.cnf** file.

For detailed information on using the Web DataBlade Module Administration Tool to set Webdriver variables, refer to Chapter 3, "Configuring Webdriver."

Possible Trace Settings for the debug_level Webdriver Variable

The following table lists the possible trace settings for the **debug_level** Webdriver variable.

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
4	Logs Webdriver query requests to the database server, such as calls to the **WebExplode()** function or authorization requests
8	Logs large object requests
16	Logs AppPage headers
32	Logs large object headers
64	Logs client file upload information
128	Logs information as AppPages are added and retrieved from the disk cache
256	Logs request variables
512	Logs information similar to the information logged by the NSAPI driver (CGI only)
1024	Logs connection pool information
2048	Logs session management information, such as persistent variables being updated and new sessions being created
4096	Logs parameters sent to the **WebExplode()** function in a decoded format

(1 of 2)

Trace Value	Information Displayed
8192	Logs parameters sent to the **WebExplode()** function in an encoded format
16384	Time stamps each request of Webdriver
32768	Logs callback messages

(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 **debug_file** variable in the Global section of the **web.cnf** file. If the trace file does not exist, it is created. If the trace file exists, new messages are appended to it.

Example of Setting the debug_level Webdriver Variable

The following example shows a Global section of a **web.cnf** file:

```
<Global>
dbconnmax      10
anchorvar      WEB_HOME
debug_file     /disk1/webdriver.log
debug_level    4
maxcharsize    2
</Global>
```

In the example, Webdriver tracing messages are written to the file **/disk1/webdriver.log**. The only type of Webdriver messages written to this file are query requests to the database server, such as calls to the **WebExplode()** function or authorization requests.

Using the Webdriver Diagnostic Page

The information in this section applies only when you use the Netscape NSAPI Webdriver. Your browser shows the same information that would be sent to the Webdriver log had you set the **debug_level** to 1024, as shown in "Enabling Webdriver Tracing" on page 12-3.

Webdriver provides a virtual diagnostic web page that gives information about connections between Webdriver and the database. To use the diagnostic web page, set the **internal_diags** variable for your configuration in the **webconfigs** table using the Web DataBlade Module Administration Tool. The value that you set it to can be any short string that makes sense to you. You are essentially naming a prefix for a dynamic page. For example:

```
set internal_diags diag
```

Then in your url, request *your_full_url_path*/**diagCP** to see connection pool information. For example, *your_full_url_path* could be one of the examples below:

```
http://host.domain/$WEB_HOME
http://host.domain/database-name
```

Errors While Retrieving Pages from the DataBase

There are some problems that may occur when Webdriver connects to **WebExplode()** and retrieves an exploded page from the database. This section discusses some of these problems, likely diagnoses, and their solutions or workarounds.

Symptom	Possible Diagnosis	Solution
When the browser requests a page, it immediately displays Error 25588. This occurs intermittently during busy times and may succeed when you retry.	The **dbconnmax** Webdriver variable was not set; it defaults to 16 and all connections were occupied.	Increase **dbconnmax**. This variable is set in Global section of the **web.cnf** file.
When the browser requests a page, after normal wait time, a popup window appears saying "Document contains no data."	The browser is requesting a URL with the **http** protocol (the URL begins with http: / /) and the web server is serving **https** (or vice versa).	Correct the URL.
The browser returns error 404 "Not found," when specifying a URL beginning with $WEB_HOME.	In your web server configuration, the URL mapping for Webdriver is not set up correctly. You may have put the mapping in the wrong place in the file. For example, when you use the Netscape server, it is an error to put the NameTrans entry for $WEB_HOME after the document root entry in **obj.conf**.	Put the URL mapping for Webdriver in the correct place.

Executing SQL Statements Greater Than 32 KB

By default, the database server returns an error if you try to execute an SQL statement larger than 32 KB within an MISQL AppPage tag. An example of an SQL statement that could be larger than 32 KB is an INSERT statement that inserts a large AppPage into a table.

To work around this problem, use the PREPARE attribute to specify the name of a variable that contains the large chunk of text (such as the entire AppPage that is going to be inserted) and then put a question mark (?) in the SQL statement to show where this text should be substituted once the SQL statement is actually executed. For more information about the PREPARE attribute, see the chapter called "Using Tags in AppPages" in the *IBM Informix Web DataBlade Module Application Developer's Guide*.

For example, assume you store movie descriptions in a table with the following schema:

```
CREATE TABLE movies
(
        id              INTEGER,
        name            VARCHAR(30),
        description     LVARCHAR
);
```

The following AppPage snippet shows how to use the MIVAR AppPage tag to first store text data in a variable called *the_text* and then use the PREPARE attribute of the MISQL tag to insert the text into the table. (For simplicity, the sample text is short; in reality you should only use the PREPARE attribute for text that is larger than 32 KB.)

```
<?MIVAR NAME=the_text>The movie was great. <?/MIVAR>
<?MISQL PREPARE=the_text
        SQL="insert into movies values (1, 'Casablanca', ?);"><?/MISQL>
```

To use more than one placeholder in the SQL statement, use a vector variable to store the text, as in:

```
<?MIVAR NAME=the_text[1]>Citizen Kane<?/MIVAR>
<?MIVAR NAME=the_text[2]>A movie classic.<?/MIVAR>
<?MISQL PREPARE=the_text
        SQL="insert into movies values (1, ?, ?);"><?/MISQL>
```

Important: *You should use the PREPARE attribute of the MISQL AppPage tag only for SQL statements that are larger than 32KB, since use of the attribute might cause the SQL statement to execute more slowly.*

Chapter

13

Web DataBlade Module Utilities

In This Chapter

Web DataBlade module utilities are programs that you run at the operating system level that perform a specific task. This chapter describes the following Web DataBlade module utilities:

- The **cm_schema_create** utility
- The **cm_schema_load** utility
- The **createAPB20_DDW20schema** utility
- The **loadAPB20application** utility
- The **webconfig** utility
- The **webpwcrypt** utility
- The **websetup** utility

The cm_schema_create Utility

The **cm_schema_create** utility creates the system tables that make up the Web DataBlade Module Administration Tool.

In particular, the **cm_schema_create** utility creates the following four tables:

- **WebCMPages**. Stores the Web DataBlade Module Administration Tool AppPages.
- **WebConfigs**. Stores all Webdriver configurations.
- **WebEnvVariables**. Stores all the Webdriver variables that can be included in a Webdriver configuration.
- **WebCMImages**. Stores the graphics used in the Web DataBlade Module Administration Tool.

Important: *You typically do not run the* ***cm_schema_create*** *utility manually because the* ***websetup*** *utility, used to initially configure the Web DataBlade module for your database, calls it automatically. The description of the* ***cm_schema_create*** *utility is provided in case you need to manually install the Web DataBlade Module Administration Tool in your database.*

The **cm_schema_create** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/admtool**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Usage

To use the **cm_schema_create** utility, execute the following command at the operating system prompt:

```
cm_schema_create database sbspace
```

The *database* argument is the name of the database for which you want to create Web DataBlade Module Administration Tool system tables. The *sbspace* argument is the name of an sbspace.

For example, to create the Web DataBlade Module Administration Tool system tables in a database called **my_db**, using the sbspace **sbsp1**, execute the following command at the operating system prompt:

```
cm_schema_create my_db sbsp1
```

You must have previously created the sbspace with the **onspaces** command. The Web DataBlade Module Administration Tool uses the sbspace to store the **object** column of the **WebCMPages** table, which is of data type HTML.

Typically, the owner of the database or the **informix** user executes the **cm_schema_create** utility.

After you execute the **cm_schema_create** utility to create the Web DataBlade Module Administration Tool system tables, execute the **cm_schema_load** utility to load data into the system tables. For detailed information on using the **cm_schema_load** utility, refer to "The cm_schema_load Utility" on page 13-5.

The cm_schema_load Utility

The **cm_schema_load** utility loads data into the Web DataBlade Module Administration Tool system tables created by the **cm_schema_create** utility. In particular, the **cm_schema_load** utility loads:

- The Web DataBlade Module Administration Tool AppPages into the **WebCMPages** system table
- The **admin**, **apb**, and **ddw** default Webdriver configurations into the **WebConfigs** system table
- All the Webdriver variables that can be included in a Webdriver configuration, along with their default and possible values, into the **WebEnvVariables** system table
- The Web DataBlade Module Administration Tool images into the **WebCMImages** system table

Important: *You typically do not run the **cm_schema_load** utility manually because the **websetup** utility, used to initially configure the Web DataBlade module for your database, calls it automatically. The description of the **cm_schema_load** utility is provided in case you need to manually install the Web DataBlade Module Administration Tool in your database.*

The **cm_schema_load** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/admtool**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Usage

To use the **cm_schema_load** utility, execute the following command at the operating system prompt:

```
cm_schema_load database
```

The *database* argument is the name of the database into which you want to load the Web DataBlade Module Administration Tool data.

For example, to load the Web DataBlade Module Administration Tool data into a database called **my_db**, execute the following command at the operating system prompt:

```
cm_schema_load my_db
```

Typically, the owner of the database or the **informix** user executes the **cm_schema_load** utility.

You must have previously run the **cm_schema_create** utility to create the Web DataBlade Module Administration Tool system tables before you run the **cm_schema_load** utility. For detailed information on using the **cm_schema_create** utility, refer to "The cm_schema_create Utility" on page 13-3.

The createAPB20_DDW20schema Utility

The **createAPB20_DDW20schema** utility creates the system tables that make up the AppPage Builder (APB) application.

The **createAPB20_DDW20schema** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/apb2**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Important: *You typically do not run the **createAPB20_DDW20schema** utility manually, because the **websetup** utility, used to initially install the Web DataBlade module for your database, calls it automatically if you choose at that time to install APB. The description of the **createAPB20_DDW20schema** utility is provided in case you need to manually install APB in your database.*

Usage

To use the **createAPB20_DDW20schema** utility, execute the following command at the operating system prompt:

```
createAPB20_DDW20schema database sbspace
```

The *database* argument is the name of the database for which you want to create the APB system tables. The *sbspace* argument is the name of an sbspace.

For example, to create the APB system tables in a database called **my_db**, using the sbspace **sbsp1**, execute the following command:

```
createAPB20_DDW20schema my_db sbsp1
```

You must have previously created the sbspace with the **onspaces** command. APB uses the sbspace to store HTML and BLOB columns, such as the **object** column in both the **wbPages** and **wbBinaries** tables.

Typically, the owner of the database or the **informix** user executes the **createAPB20_DDW20schema** utility.

After you execute the **createAPB20_DDW20schema** utility to create the APB system tables, execute the **loadAPB20application** utility to load data into the system tables. For detailed information on using the **loadAPB20application** utility, refer to "The loadAPB20application Utility," following.

The loadAPB20application Utility

The **loadAPB20application** utility loads data into the APB system tables.

The **loadAPB20application** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/apb2**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Important: *You typically do not run the **loadAPB20application** utility manually, because the **websetup** utility, used to initially configure the Web DataBlade module for your database, calls it automatically if you choose at that time to install APB. The description of the **loadAPB20application** utility is provided in case you need to manually install APB in your database.*

Usage

To use the **loadAPB20application** utility, execute the following command at the operating system prompt:

```
loadAPB20application database
```

The *database* argument is the name of the database into which you want to load the APB data.

Typically, the owner of the database or the **informix** user executes the **loadAPB20application** utility.

For example, to load the APB data into a database called **my_db**, execute the following command:

```
loadAPB20application my_db
```

You must have previously run the **createAPB20_DDW20schema** utility to create the APB system tables before you run the **loadAPB20application** utility. For detailed information on using the **createAPB20_DDW20schema** utility, refer to "The createAPB20_DDW20schema Utility" on page 13-6.

The webconfig Utility

Use the **webconfig** utility to add new Webdriver mappings to the **web.cnf** file and convert **web.cnf** files created before Version 4.0 of the Web DataBlade module into current **web.cnf** files.

You typically use the **webconfig** utility to add a special Webdriver mapping to the **web.cnf** file used to invoke the Web DataBlade Module Administration Tool. You add subsequent Webdriver mappings with the Web DataBlade Module Administration Tool; you do not add them with the **webconfig** utility.

You can also use the **webconfig** utility to convert **web.cnf** files created before Version 4.0 of the Web DataBlade module into current **web.cnf** files and Webdriver configurations. In versions of the Web DataBlade module earlier than 4.0, the **web.cnf** file contained definitions for all Webdriver variables. The **web.cnf** file in Version 4.0 and later of the Web DataBlade module contains only the Webdriver variables needed to connect to the database; all other Webdriver variables are stored in the database in Webdriver configurations. The **webconfig** utility can migrate the information from old **web.cnf** files to new **web.cnf** files and Webdriver configurations.

The **webconfig** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/utils**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

For detailed information on Webdriver mappings, Webdriver configurations, Webdriver variables, and the **web.cnf** file, refer to Chapter 3, "Configuring Webdriver."

Usage

You must set the **MI_WEBCONFIG** environment variable to the full pathname of the **web.cnf** file before you run the **webconfig** utility. For example, if the **web.cnf** file is located in the **/local1/webserver** directory, then the following UNIX C shell command sets the **MI_WEBCONFIG** variable:

```
setenv MI_WEBCONFIG /local1/webserver/web.cnf
```

You must execute the **webconfig** utility as the user who starts the Web server processes. Similarly, the owner of the **web.cnf** file should be the user who starts the Web server processes. If you choose to run the Web server processes as the user **nobody**, the **root** user should run the **webconfig** utility.

You must restart the Web server after you run the **webconfig** utility or update the **web.cnf** file manually. The Web server reads the **web.cnf** file only once and then caches the information. Therefore, if you have updated the **web.cnf** file, either by running the **webconfig** utility or by editing the file manually, you must restart the Web server so it will reread the file.

You can use the following options with the **webconfig** utility.

Option	Description
-addmap	Adds a new Webdriver mapping to the **web.cnf** file Typically, you use this option only to add a special Webdriver mapping that is used to invoke the Web DataBlade Module Administration Tool.
-convert	Converts a **web.cnf** file created before Version 4.0 of the Web DataBlade module into a current **web.cnf** file It also creates a new Webdriver configuration that will contain the Webdriver variables defined in the old **web.cnf** file.
-secure	Enables security for the Web DataBlade Module Administration Tool by adding the **config_password** and **config_user Global** variables to the **web.cnf** file
-p <*mapping_name*>	Specifies the name of the new Webdriver mapping Typically, you use this option only to add the Webdriver mapping that is used to invoke the Web DataBlade Module Administration Tool. The full name of this special Webdriver mapping should be `/dbname/admin`, where *dbname* is the name of the database for which you are adding the Webdriver mapping.
-n <*config_name*>	Specifies the name of the Webdriver configuration that the new Webdriver mapping will use If you are adding the special Webdriver mapping that is used to invoke the Web DataBlade Module Administration Tool, you should specify `-n admin` for this option. When you use this option in conjunction with the **-convert** option, *config_name* refers to the name of the new Webdriver configuration that is created to contain the Webdriver variables that were specified in the old **web.cnf** file.
-d <*database*>	Specifies the name of the database for which you are adding a new Webdriver mapping

(1 of 2)

Option	Description
-u <*user*>	Specifies the user that will be used for all connections to the database with the new Webdriver mapping. The **webconfig** utility prompts you for the user's password. The utility encrypts the password before it adds it to the **web.cnf** file.
-s <*server_name*>	Specifies the name of an Informix database server that the new Webdriver mapping will use, if different from the Informix server specified by the **INFORMIXSERVER** environment variable in the Setvar section of the **web.cnf** file
-f <*old_web.cnf_file*>	Specifies the full pathname of the **web.cnf** file created before Version 4.0 of the Web DataBlade module Always use this option in conjunction with the **-convert** option to convert old **web.cnf** files into new **web.cnf** files.
-verify	Checks the **web.cnf** file If you specify the **-verify** option with the **-p <*mapping_name*>** option, only the specified Webdriver mapping is checked. If you specify the **-verify** option on its own, all Webdriver mappings are checked. The **-verify** option checks whether: ■ The syntax of the Webdriver mapping in the **web.cnf** file is correct. ■ A connection to the database can be made. ■ The **WebConfigs** system table exists in the database and that it contains an entry for the appropriate Webdriver configuration.
-o <*output_file*>	Writes the output to the file called ***output_file*** Use the **-o** option together with the **-addmap** option to create a temporary file with a Map entry when you are unable to write directly to the **web.cnf** file due to lack of necessary permissions.
-i <*input_file*>	Copies the contents of the **input_file** file to the **web.cnf** file You typically use the **-i** option to update the **web.cnf** file as the owner of the **web.cnf** file after another user has used the **-o** option to create a temporary file with a Map entry.

(2 of 2)

The following section provides examples of using the options described in the preceding table.

Examples

The following example shows how to add the special Webdriver mapping that invokes the Web DataBlade Module Administration Tool for the **hr_db** database:

```
webconfig -addmap -p /hr_db/admin -n admin -d hr_db -u hr_user
```

The **webconfig** utility prompts you for the password of the **hr_user** user.

The following example shows how to convert the **web.cnf** file created before Version 4.0 of the Web DataBlade module and currently located in the **/oldfiles** directory into a new **web.cnf** file. The utility also creates a new Webdriver configuration called **new_config** that contains the Webdriver variables originally defined in the old **web.cnf** file. It also creates a new Webdriver mapping called **/newmap**.

```
webconfig -convert -p /newmap -n new_config -f /oldfiles/web.cnf
```

This example does not specify the options **-d** or **-u** because the old **web.cnf** file contains the name of the database and the name of the user with which the connection to the database should be made.

The following example shows how to check the syntax of the `/newmap` Webdriver mapping in the **web.cnf** file, whether a connection to the database defined for the Webdriver mapping can be made, and whether the Webdriver configuration defined for the Webdriver mapping exists in the **WebConfigs** table:

```
webconfig -verify -p /newmap
```

The webpwcrypt Utility

The **webpwcrypt** utility encrypts a user's password.

When you create a new Webdriver mapping with the Web DataBlade Module Administration Tool, you specify the name of a database and the user with which the connection to the database is made. The Web DataBlade Module Administration Tool then writes the name of the database and the user's name to the **web.cnf** file, as well as the user's encrypted password. The Web DataBlade Module Administration Tool automatically encrypts the password using its own encryption key.

If, however, you want to use your own encryption key, you must use the **webpwcrypt** utility to create the encrypted password and update the **web.cnf** file manually.

The **webpwcrypt** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/utils**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Usage

To use the **webpwcrypt** utility, execute the following command at the operating system prompt:

```
webpwcrypt database user key
```

The arguments are described in the following table.

Argument	Description
database	Name of the database being accessed
user	Name of the user accessing the database
key	User-supplied key used in the encryption process (can be any string)

The utility prompts you twice for the user's password and returns it in encrypted form.

Example

The following example shows how to run the **webpwcrypt** utility to encrypt the password of the **webuser** user using the user-supplied key webkey for the **webdb** database:

```
webpwcrypt webdb webuser webkey
```

The utility prompts you for the password of the **webuser** user. In the example, the password is webuserpassword:

```
Enter password for user "webuser":    <enter webuserpassword>
Enter password again:                 <re-enter webuserpassword>
```

The **webpwcrypt** utility returns:

```
password       c47c6e1c91d32affd138212b24277f85
password_key   webkey
```

The websetup Utility

The **websetup** utility configures the Web DataBlade module for your database server.

You execute the **websetup** utility once to configure your Web server to use the Web DataBlade module and once for each database that uses the Web DataBlade module.

The **websetup** utility asks you questions such as whether you want to configure database components, Web components, or both types of components. The **websetup** utility configures the Web DataBlade module for your database based on your answers.

The **websetup** utility defines two types of components as follows:

- **Database components**. If you choose to configure database components, the **websetup** utility installs the Web DataBlade Module Administration Tool and APB in your database and updates the **web.cnf** file with the relevant information.
- **Web components**. If you choose to configure Web components, the **websetup** utility creates the **web.cnf** file, moves it to its permanent location, and updates it with the relevant information. If you are going to use the NSAPI Webdriver, the **websetup** utility updates the Netscape **obj.conf** file with Informix-specific information.

Although you can configure the two types of components separately, Informix recommends that you configure both at the same time in the same **websetup** session.

The **websetup** utility is located in the directory ***INFORMIXDIR*/extend/web.*version*/install**, where *INFORMIXDIR* refers to the main Informix directory and *version* refers to the current version of the Web DataBlade module installed on your computer.

Usage

To use the **websetup** utility, execute the following command at the operating system prompt:

```
websetup
```

If the same user owns both the Web server and the database, run the **websetup** utility as that user.

If different users own the Web server and database, run the **websetup** utility as the **root** user. The **websetup** utility will then have full permission to execute its various tasks as the appropriate user: the owner of the Web server or the owner of the database.

If, however, you are unable to become the **root** user on your computer, you must run the **websetup** utility twice: first as the Web server owner to configure the Web server components and second as the database owner to configure the database components.

Appendix

A

Web DataBlade Module System Tables

This appendix describes the following Web DataBlade module system tables that are created when you register the DataBlade module in your database:

- **WebTags**
- **WebUdrs**

This appendix also describes the following Web DataBlade Module Administration Tool system tables that are created when you install the tool in your database:

- **WebConfigs**
- **WebCMPages**
- **WebCMImages**
- **WebEnvVariables**

The AppPage Builder (APB) system tables are described in the *IBM Informix Web DataBlade Module Application Developer's Guide.*

WebTags

The **WebTags** system table stores both system and user-defined dynamic tag definitions. You add and update user-defined dynamic tags with AppPage Builder (APB). Once you add a new tag to the **WebTags** system table, you can invoke the tag in any AppPage.

The **WebTags** system table is created when you register the Web DataBlade module in your database.

For detailed information on user-defined dynamic tags, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

The following table describes the columns of the **WebTags** system table.

Column Name	Data Type	Description
ID	VARCHAR(40) NOT NULL	Unique identifier for the dynamic tag
parameters	VARCHAR(250)	Ampersand-separated list of parameters to the dynamic tag
class	VARCHAR(40)	Class of dynamic tag For example, you can have `beginning`, `expert`, or any other class name. System dynamic tags have the class name `system`, and cannot be modified in APB. The default class is `user`.
description	VARCHAR(250)	Description of the dynamic tag
content	HTML NOT NULL	Body of the dynamic tag

WebUdrs

The **WebUdrs** system table stores information about UDR tags.

You add and update UDR tags with AppPage Builder (APB).

The **WebUdrs** system table is created when you register the Web DataBlade module in your database.

For detailed information on how to use UDR tags in an AppPage, refer to the *IBM Informix Web DataBlade Module Application Developer's Guide*.

The following table describes the columns of the **WebUdrs** system table.

Column Name	Data Type	Description
ID	VARCHAR(40)	Unique identifier of the routine Specify this identifier when you invoke the routine in an AppPage with the tag <*?udrname...>*. The value in this column does not have to match the corresponding value in the **sysprocedures** system table.
parameters	VARCHAR(250)	Ampersand-separated list of parameters to the routine 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: `param=`.
class	VARCHAR(40)	Class of the routine For example, you can specify `beginning`, `expert`, or any other class name. If you specify the class name `system`, you cannot use AppPage Builder to delete the routine from the **WebUdrs** system table.
description	VARCHAR(250)	Description of the routine
procid	INTEGER	Unique identifier of the routine as specified in the **procid** column of the **sysprocedures** system table The value in the **procid** column of the **WebUdrs** system table must exactly match the corresponding value in the **procid** column in the **sysprocedures** system table for the specified routine.

(1 of 2)

Column Name	Data Type	Description
procname	VARCHAR(128)	Unique name of the routine as specified in the **procname** column of the **sysprocedures** system table The value in the **procname** column of the **WebUdrs** system table must exactly match the corresponding value in the **procname** column in the **sysprocedures** system table for the specified routine.
numargs	INTEGER	Number of arguments of the routine The value in the **numargs** column of the **WebUdrs** system table must exactly match the corresponding value in the **numargs** column in the **sysprocedures** system table for the specified routine.
paramtypes	LVARCHAR	Comma-delimited string that specifies the data type of each argument The number of delimited data types must match the number of arguments specified by the **numargs** column. An example is `html,html,integer`

(2 of 2)

The primary key of the **WebUdrs** system table is the **id** column.

WebConfigs

The **WebConfigs** system table is part of the Web DataBlade Module Administration Tool schema. It stores the Webdriver configurations, along with the Webdriver variables defined for each configuration, that exist in the database.

The table has one row for each Webdriver variable in each Webdriver configuration. Therefore, the primary key of the table is composed of two columns: the Webdriver configuration column and the Webdriver variable column.

The **WebConfigs** system table is created when you install the Web DataBlade Module Administration Tool in your database.

For detailed information on Webdriver configurations, Webdriver variables, and the Web DataBlade Module Administration Tool, refer to Chapter 3, “Configuring Webdriver.”

The following table describes the columns of the **WebConfigs** system table.

Column Name	Data Type	Description
config_name	VARCHAR(80) NOT NULL	The name of the Webdriver configuration Webdriver configurations map to virtual paths on the Web server. The **web.cnf** file defines the mappings between the Web server virtual paths and the Webdriver configurations stored in this system table.
variable_name	VARCHAR(80) NOT NULL	A Webdriver variable that makes up the Webdriver configuration specified by the **config_name** column The value of this column must also be a value of the **variable_name** column of the **WebEnvVariables** system table.
overwrite	CHAR(1) NOT NULL	Specifies whether the Webdriver variable can be overwritten in the URL used to call an AppPage This column can be set to `Y` or `N`.
value	VARCHAR(255)	The value of the Webdriver variable This value must be one of the values listed in the **possible_values** column of the **WebEnvVariables** system table.
disable	CHAR(1)	Specifies whether the Webdriver variable is enabled or disabled This column can be set to `Y` or `N`.
time_stamp	DATETIME YEAR TO SECOND	Indicates when the value of the Webdriver variable was last updated You can use this column to see when the description of a Webdriver variable was last updated.

The primary key of the **WebConfigs** system table, defined by the constraint called **pk_webconfigs**, includes two columns: **config_name** and **variable_name**.

WebCMPages

The **WebCMPages** system table is part of the Web DataBlade Module Administration Tool schema. It stores all the AppPages that make up the Web DataBlade Module Administration Tool.

The **WebCMPages** system table is created when you install the Web DataBlade Module Administration Tool in your database.

For detailed information on the Web DataBlade Module Administration Tool, refer to Chapter 3.

The following table describes the columns of the **WebCMPages** system table.

Column Name	Data Type	Description
ID	VARCHAR(40) NOT NULL	Unique ID of the AppPage The Webdriver variable **MIval** is set to this value in the URL to call the AppPage.
description	VARCHAR(40) NOT NULL	Description of the AppPage
read_level	INTEGER	Specifies the authorization level of the AppPage Used with AppPage-level security.
object	HTML	The AppPage itself
time_stamp	DATETIME YEAR TO SECOND	Indicates when the AppPage was last updated

The primary key of the **WebCMPages** system table, defined by the constraint called **pk_webcmpages**, is the **ID** column.

WebCMImages

The **WebCMImages** system table is part of the Web DataBlade Module Administration Tool schema. It stores the images used in the Web DataBlade Module Administration Tool.

The **WebCMImages** system table is created when you install the Web DataBlade Module Administration Tool in your database.

For detailed information on the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

The following table describes the columns of the **WebCMImages** system table.

Column Name	Data Type	Description
ID	VARCHAR(40) NOT NULL	Unique ID of the image
description	VARCHAR(250)	Description of the image
height	INTEGER	Height of the image
width	INTEGER	Width of the image
object	BLOB	The image itself

The primary key of the **WebCMPages** system table is the **ID** column.

WebEnvVariables

The **WebEnvVariables** system table is part of the Web DataBlade Module Administration Tool schema. It stores all the Webdriver variables provided by the Web DataBlade module that can be included in a Webdriver configuration.

The **WebEnvVariables** system table is created as part of the installation of the Web DataBlade Module Administration Tool in your database.

For detailed information on Webdriver configurations, Webdriver variables, and the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

The following table describes the columns of the **WebEnvVariables** system table.

Column Name	Data Type	Description
variable_name	VARCHAR(80) NOT NULL	The name of the Webdriver variable Examples of Webdriver variables include **cache_page** and **redirect_url**.
data_type	CHAR(1) NOT NULL	The data type of the Webdriver variable This column can be set to character (C), numeric (N), or time (T). If set to C, the value of the Webdriver variable must be a character string. If set to N, the value of the variable must be numeric. If set to T, the value of the variable must be a time unit in seconds (s or S), hours (h or H), or days (d or D).
default_value	VARCHAR(255)	The default value of the Webdriver variable When you add a Webdriver variable to a Webdriver configuration, this is the default value of the variable.
possible_values	VARCHAR(255)	The comma-separated list of possible values of a Webdriver variable When you add a Webdriver variable to a Webdriver configuration, the Web DataBlade Module Administration Tool uses this column to validate the value you assign the variable.
time_stamp	DATETIME YEAR TO SECOND	Indicates when the definition of the Webdriver variable was last updated

The primary key of the **WebEnvVariables** system table, defined by the **pk_webenvvariables** constraint, is the **variable_name** column.

Appendix

B

Web DataBlade Module Variables

This appendix provides the full list of Webdriver and **WebExplode()** variables. The appendix is organized into the following sections:

Use the Web DataBlade Module Administration Tool to set the Webdriver and **WebExplode()** variables that are stored in the database as part of your Webdriver configuration.

Many Webdriver variable names changed in Version 4.0 of the Web DataBlade module. This appendix also provides, where applicable, the old name of the Webdriver variable.

For detailed information about using the Web DataBlade Module Administration Tool, refer to Chapter 3, "Configuring Webdriver."

Webdriver Variables Stored in the web.cnf File

This section describes the Webdriver variables that are stored in the Global, Setvar, and Map sections of the **web.cnf** file.

The Global Section of the web.cnf File

The following table lists all the variables you can set in the Global section of the **web.cnf** file

Variable	Mandatory?	Description
dbconnmax	No	Specifies the maximum number of connections to the database. The default value is `16`
anchorvar	Yes	Specifies the name of the anchor variable used when an AppPage calls another AppPage This variable is mandatory. For the NSAPI and Apache Webdrivers, **anchorvar** should always be set to `WEB_HOME`, with a trailing forward slash (/). For the ISAPI Webdriver, the variable should be set to `WEB_HOME/drvisapi.dll`. For the CGI Webdriver, the variable should be set to `WEB_HOME/webdriver`. Since **anchorvar** is always set to **WEB_HOME**, you can always use **WEB_HOME** as an anchor variable in any AppPage.
driverdir	No	Specifies the directory that Webdriver uses to internally coordinate its interaction with the Web server The default value of this variable is `/tmp`. This variable is only used by the Apache and CGI implementations of Webdriver.
debug_file	No	Specifies the full pathname of the log file to which Webdriver messages are written
debug_level	No	Enables Webdriver tracing to the log file specified by the **debug_file** variable You can override the value of the **debug_level** variable in the Global section of the **web.cnf** file by setting it in your Webdriver configuration using the Web DataBlade Module Administration Tool.

(1 of 2)

Variable	Mandatory?	Description
maxcharsize	No	When set to a value greater than `1`, *each* character sent to the **WebExplode()** function is URL-encoded. If this variable is not set, Webdriver URL-encodes only special characters (such as `&`) before sending it to the **WebExplode()** function. It is recommended that you set this variable to a value greater than `1` *only* if you are using a multibyte character set. This is because you might see a degradation in performance if Webdriver is forced to URL-encode every character before sending it to the **WebExplode()** function. You can override the value of this variable for your Webdriver mapping by adding it as a Webdriver variable to the appropriate Webdriver configuration.
config_user	No	The name of the user who is allowed to use the Web DataBlade Module Administration Tool Add this variable to the **web.cnf** file only with the **webconfig** utility.
config_password	No	The password of the **config_user** user Add this variable to the **web.cnf** file only with the **webconfig** utility.

dbconntimeout	No	Sets the maximum time (in seconds) that a Webdriver connection to the database is allowed to be idle
		
		Webdriver automatically closes any database connections that have been idle for longer than the value of **dbconntimeout**.
		
		The following sample shows how to set **dbconntimeout** to 120 seconds:
		
		`<GLOBAL>`
		`debug_file /tmp/driver.log`
		`debug_level -1`
		`dbconntimeout 120`
		`dbconnmax 128`
		`anchorvar WEB_HOME/`
		`</GLOBAL>`
		
		If no database request is made on a connection for 2 minutes, then Webdriver closes the connection.

(2 of 2)

The Setvar Section of the web.cnf File

You set Informix environment variables in the Setvar section of the **web.cnf** file.

The following Informix environment variables are discussed in the *IBM Informix Web DataBlade Module Administrator's Guide*:

- **INFORMIXSERVER**
- **INFORMIXDIR**

For a complete list of the Informix environment variables you can set in the Setvar section of the **web.cnf** file, refer to *IBM Informix Guide to SQL: Reference*.

Important: *Do not set the Informix environment variables DBDATE and DBCENTURY in your* ***web.cnf*** *file. Their settings will be ignored. Instead, set them in your environment before you register the DataBlade module in your database.*

The Map Section of the web.cnf File

The following table lists all the variables that can be included in the Map section of the **web.cnf** file.

Map Variable	Mandatory?	Description
database	Yes	The name of the database to which Webdriver connects when a URL prefix specifies this Webdriver mapping
user	Yes	The name of the user who connects to the database specified by the **database** variable
password	Yes	The encrypted password of the user specified by the **user** variable
password_key	Yes	The key that Webdriver uses to decrypt the password specified by the **password** variable

(1 of 2)

Map Variable	Mandatory?	Description
server	No	The Informix database server to use when making the connection to the database If this variable is not set, the connection is made using the **INFORMIXSERVER** database server.
config_name	Yes	The name of the Webdriver configuration to use The Webdriver configuration is stored in the **WebConfigs** system table in the database specified by the **database** variable.
config_security	No	When set to `ON`, security is enabled for this Webdriver mapping, which means that only the user specified by the **config_user** variable in the Global section of the **web.cnf** file can use this Webdriver mapping. The **config_security** variable should appear only in Webdriver mappings used to invoke the Web DataBlade Module Administration Tool.

(2 of 2)

Webdriver Variables Stored in the Database

This section describes the Webdriver variables that are stored in the database as part of a Webdriver configuration. These include both schema-related Webdriver variables and feature-related Webdriver variables.

Managing Webdriver Connections to the Database

To modify the behavior of Webdriver connections to the database for specific Webdriver configurations, use the Web DataBlade Module Administration Tool to set the Webdriver variables described in the following table.

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
connection_life	**MI_WEBRECONNECT**	No	Specifies the life of a connection, or in other words, the maximum number of requests (an integer value) that Webdriver makes to the database before the connection is shut down and reestablished The default value is 100. You should set this Webdriver variable to another value only under the guidance of Technical Support.
connection_wait	**MI_WEBDBCONNWAIT**	No	Specifies the amount of time, in milli-seconds, that Webdriver yields and waits to establish a connection if Webdriver was unable to make the initial connection due to the maximum number of database connections having already been reached The maximum number of Webdriver connections to the database server is specified by the **dbconnmax** Webdriver variable in the Global section of the **web.cnf** file.

(1 of 4)

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
connect_as_user	**MI_USER_REMOTE**	No	When set to ON, specifies that Webdriver should establish the connection to the database as the user specified by the **REMOTE_USER** Web browser variable and not as the user specified in the Map section of the **web.cnf** file By default, if this Webdriver variable is not set, Webdriver always establishes connections to the database as the user specified by the **user** Webdriver variable in the appropriate Map section of the **web.cnf** file. This Webdriver variable applies only to the NSAPI, ISAPI, and Apache implementation of Webdriver. In addition, you can only use this Webdriver variable if you have enabled user authentication for the corresponding Web server.
connect_user_max	**MI_USER_DBCONNMAX**	No	Specifies the maximum number of connections that Webdriver establishes as the user specified by the **REMOTE_USER** Web browser variable The default value of this Webdriver variable is 1. The **connect_user_max** Webdriver variable can only be set in conjunction with the **connect_as_user** Webdriver variable. This Webdriver variable applies only to the NSAPI, ISAPI, and Apache implementation of Webdriver. In addition, you can only use this Webdriver variable if you have enabled user authentication for the corresponding Web server.
query_timeout	**MI_WEBQRYTIMEOUT**	No	Specifies the maximum number of seconds that Webdriver allows a query to run before Webdriver interrupts the query

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
keepalive	**MI_WEBKEEPALIVE**	No	Specifies the interval in seconds at which Webdriver checks the Web browser connection If the browser is no longer connected because a STOP or CANCEL signal has been sent by the browser, the running query is interrupted, and the Web server is freed to execute the next query request. This variable applies only to the NSAPI, ISAPI, and Apache implementation of Webdriver.
init_sql	**MI_WEBINITIALSQL**	No	Specifies that Webdriver should send initial SQL statements to the database server when Webdriver makes a connection to the database Set this Webdriver variable to one or more SQL statements, separated by semicolons and terminated by a carriage return. Do not include quotes. For example, if you want to set the isolation level of the connection to the database to dirty read, set the **init_sql** Webdriver variable to the value `SET ISOLATION TO DIRTY READ;`

(3 of 4)

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
max_html_size	**MI_WEBMAXHTMLSIZE**	No	Specifies the largest AppPage, in bytes, that Webdriver sends to the browser. AppPages larger than this size are not sent to the browser The default value for this Webdriver variable is 128 KB. The maximum value is 2^{32} KB.
maxcharsize	**New in Version 4.0**	No	When set to a value greater than `1`, *each* character sent to the **WebExplode()** function is URL-encoded. If this variable is not set, Webdriver URL-encodes only special characters (such as `&`) before sending it to the **WebExplode()** function. It is recommended that you set this variable to a value greater than `1` *only* if you are using a multibyte character set. This is because you might see a degradation in performance if Webdriver is forced to URL-encode every character before sending it to the **WebExplode()** function. You can specify the **maxcharsize** variable in the Global section of the **web.cnf** file if you want to specify globally that characters should be URL-encoded. By adding the variable to a Webdriver configuration, however, you can control this behavior for a single Webdriver configuration and not for the whole database server.

(4 of 4)

Using Server-Side Includes in AppPages with the Apache or NSAPI Webdriver

To use server-side includes in your AppPages with the DYNAMIC option to the PARSE-HTML variable-processing function, you must use the Web DataBlade Module Administration Tool to set the Webdriver variable described in the following table.

Webdriver Variable	Mandatory?	Description
parse_html_directory	Yes	Specifies the full pathname of the directory on the Web server computer where Webdriver temporarily stores the AppPage to be subsequently read by the Web server Webdriver does not create this directory, so be sure the directory exists before you use server-side includes in an AppPage.

Resetting User Name/Password Combinations

To reset user name / password combinations so users can change their passwords within a Web application, use the Web DataBlade Module Administration Tool to set the Webdriver variable listed in the following table.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
auth_cache	**MI_WEBAUTHCACHE**	Yes	Allows you to reset user name and password combinations so users can change their passwords within an application You can set the **auth_cache** Webdriver variable to three values: on, `off`, and `check`. The default value is on. If you set the variable to on, Webdriver always uses the password value in the Web server cache. If you set the variable to `off`, Webdriver always uses the password value in the database. If you set the variable to `check`, if the value in the Web server cache is different from the Web browser value, Webdriver updates the Web server cache with the password value in the database.

Enabling NSAPI, ISAPI, and Apache Security

To use the security features of the Netscape Web server, Microsoft Internet Information Server, or Apache Web Server, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table.

Variable Name	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
MIusertable	Same	Yes	Name of the table that contains user access information
MIusername	Same	Yes	Name of the VARCHAR column in the user access table (**MIusertable**) that contains the name of the database user
MIuserpasswd	Same	Yes	Name of the VARCHAR column of the user access table (**MIusertable**) that contains the password of the database user
MIuserlevel	Same	Yes	Name of the INTEGER column of the user access table (**MIusertable**) that contains the access level of the database user
MIpagelevel	Same	Yes	Name of the INTEGER column of the table that stores your AppPage that contains the access level of the AppPage
MIusergroup	Same	No	Name of the INTEGER column of the user access table (**MIusertable**) that contains the group access level of the user
iis_nt_user	**MI_WEBNTUSER**	Yes	(ISAPI Webdriver only) Name of a valid Windows NT user

(1 of 2)

Variable Name	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
iis_nt_password	**MI_WEBNTPASSWORD**	Yes	(ISAPI Webdriver only) Password of a valid Windows NT user
redirect_url	**MI_WEBREDIRECT**	No	URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
auth_crypt_udr	New in Version 4.0	No	Enables password encryption when set to `ON` If password encryption is enabled, Webdriver encrypts the password entered by the user and compares it to the encrypted password in the **MIusertable** table. If they match, then the user is authenticated. If set to `OFF` (default value), then Webdriver does not encrypt the password.

(2 of 2)

Enabling Basic AppPage-Level Security

To configure AppPage-level authorization, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
MIpagelevel	Same	Yes	Specifies the name of the INTEGER column of the table that stores AppPages that contains the access level of the AppPage
MI_WEBACCESSLEVEL	Same	Yes	Specifies the access level of all users for a particular Webdriver configuration
redirect_url	**MI_WEBREDIRECT**	No	Specifies the URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
error_page	**MI_WEBERRORPAGE**	No	Set to the value of the AppPage that contains error handling routines

Customizing the Query to Retrieve Large Objects

To customize the query that Webdriver uses to retrieve large objects, add the Webdriver variables described in the following table to your Webdriver configuration using the Web DataBlade Module Administration Tool.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
lo_query_string	**MI_WEBLOQUERY**	Yes	Contains the SQL statement that is used to query the database for a large object Use standard C language variable syntax '%s' to specify a parameter string.
lo_query_params	**MI_WEBLOPARAMS**	Yes	Specifies the variables that are substituted for the parameters in the SQL statement specified by the **lo_query_string** variable You *must* use the variable name **MIvalObj** to specify the name of the large object you want to retrieve.
lo_error_zerorows	**MI_WEBLOZEROROWS**	No	Specifies the integer error number that Webdriver should return if the SQL statement that Webdriver uses to retrieve large objects, specified by the **lo_query_string** variable, returned zero rows
lo_error_sql	**MI_WEBLOSQLERROR**	No	Specifies the integer error number that Webdriver should return if an SQL error occurs when Webdriver retrieves a large object using the SQL statement specified by the **lo_query_string** variable.

Enabling AppPage Caching

To set AppPage caching for your Webdriver configuration, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table.

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
cache_page	**MI_WEBCACHEPAGE**	Yes	Specifies whether AppPage caching is enabled or disabled Set to `ON` to enable AppPage caching and `OFF` to disable AppPage caching. The default value is `OFF`.
cache_directory	**MI_WEBCACHEDIR**	Yes	Specifies the full pathname of the directory on the Web server computer in which cached AppPages and large objects are placed If this variable is not set, neither AppPages nor large objects are cached.
cache_page_buckets	New in Version 4.0	No	Specifies the number of subdirectories per AppPage created under the directory specified by **cache_directory** The default is one subdirectory per AppPage. Set this variable only if you intend on caching AppPages that might have over 1000 different versions.

(1 of 4)

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
cache_page_life	**MI_WEBPAGELIFE**	No	Specifies the length of time after which an AppPage is refreshed from the database Set **cache_page_life** in units of seconds (`s` or `S`), hours (`h` or `H`), or days (`d` or `D`). For example, the value `5d` indicates five days.
cache_admin	**MI_WEBCACHEADMIN**	No	Specifies the name of the Cache Administration AppPage The Cache Administration AppPage is not stored in the database, but is an internal AppPage managed by Webdriver. When **MIval** is set to this value, Webdriver invokes this AppPage so you can add, delete, purge, or view cache entries in the **cache_directory** directory. The default value is `cacheadmin`.

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
cache_admin_password	**MI_WEBCACHEPASSWORD**	No	Specifies that cache administration requests are processed only if the password entered in the Cache Administration AppPage matches this value
cache_page_timestamp	New in Version 4.0	No	Specifies that Webdriver, when invoking an AppPage for which AppPage caching has been enabled, adds time-stamp information at the bottom of the page The time stamp is enclosed in an HTML comment and thus is only seen if a user views the HTML source of the AppPage in their browser. The default value is `OFF`. To enable this feature, set this Webdriver variable to `ON`.

(3 of 4)

Webdriver Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
cache_page_debug	New in Version 4.0	No	Specifies that Webdriver invokes AppPages that contain deferred sections (delimited with the MIDEFERRED tag) without returning an error, even if AppPage caching has *not* been enabled This Webdriver variable is used to debug problems with partial AppPage caching. The **cache_page_debug** Webdriver variable can be set to two values: `show_defer` and `execute_defer`. When set to `show_defer` and you invoke an AppPage with a deferred section, Webdriver returns the deferred section in its original form. If the Webdriver variable is set to `execute_defer`, Webdriver executes the deferred section when you invoke the AppPage.

(4 of 4)

Enabling Large Object Caching

To set large object caching, use the Web DataBlade Module Administration Tool to set the Webdriver variables listed in the following table.

Webdriver Variable	Name of Variable in Version 3.3 and Previous	Mandatory?	Description
cache_directory	MI_WEBCACHEDIR	Yes	Specifies the directory on the Web server computer in which cached large objects are placed If not set, large objects are not cached.
cache_buckets	MI_WEBCACHESUB	No	Specifies the number of subdirectories per database created under the directory specified by **cache_directory** The default is one subdirectory per database.
cache_maxsize	MI_WEBCACHEMAXLO	No	Specifies the maximum size in bytes of large objects to be cached The default is `64 KB`.

Enabling Webdriver Tracing

The following table describes each variable for enabling Webdriver tracing.

Variable	Name of Variable in Versions 3.3 and Previous	Description
debug_level	**MI_WEBDRVLEVEL**	Enables Webdriver tracing to the log file specified by the **debug_file** variable
debug_file	New in Version 4.0	Specifies the full pathname of the log file to which Webdriver messages are written

Enabling Use of Session Variables in AppPages

To enable the use of session variables in your AppPages, use the Web DataBlade Module Administration Tool to set the following Webdriver variables.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
session	**MI_WEBSESSION**	Yes	This variable allows you to select the method for binding a session ID to the browser. This variable can have values of `url`, `cookie`, or `auto`. If set to `url`, then the session ID is bound to any dynamic anchor variable contained within the page. Typically, this variable would be **$WEB_HOME**. If set to `cookie`, the session ID is tracked with a variable sent back to the browser as a cookie. If you select `auto`, Webdriver automatically determines which method is best to use.
session_home	**MI_WEBSESSIONHOME**	Yes, if using `auto` or `url`	This variable identifies which configuration file variable is used by your application to anchor HREF tags. For example, if your application uses **WEB_HOME** as its anchor, **WEB_HOME** is the value set for this variable. If multiple values are required for this variable, they should be separated by commas.

(1 of 2)

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
session_location	**MI_WEBSESSIONLOC**	Yes	This variable describes how the persistent state is handled. If the session code is going to run within the same process, this variable needs to refer to the full path of the directory to create session state files. This directory must be created and owned by the same user that owns the Web server. If the code is going to run as a separate process, the variable needs to refer to a port and IP address in the form `port@ip-address`.
session_buckets	**MI_WEBSESSIONSUB**	No	This variable is used to define the number of subdirectories that are available to hash the session data if the site is exceptionally large. It is only required if session management is being controlled within the same process. The default is `100`.
session_life	**MI_WEBSESSIONLIFE**	No	This variable is used to define the amount of time a session is allowed to continue. It measures time from the last update to the session stack (if a session stack exists) or time from session creation. Granularity is in seconds (default), hours (`h`) or days (`d`) and uses the same syntax as **cache_page_life**. For more information about AppPage caching, refer to the *IBM Informix Web DataBlade Module Administrator's Guide*.

(2 of 2)

Handling Errors with the MI_DRIVER_ERROR Variable

Set the following Webdriver variables with the Web DataBlade Module Administration Tool to modify the error messages seen by the browser as different types of errors are encountered.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
show_exceptions	**MI_WEBSHOWEXCEPTIONS**	No	Set to on or `off` When on, Webdriver displays the database exception returned by the **WebExplode()** function. When `off`, Webdriver displays the `HTTP/1.0 500 Server error` message. Default is `off`.
redirect_url	**MI_WEBREDIRECT**	No	Set to the URL to redirect users to if they do not have access to the AppPage they attempt to retrieve
error_page	**MI_WEBERRORPAGE**	No	Set to the value of the AppPage that contains error handling routines

Displaying Database Errors in a Browser

To display database errors in your browser, instead of the generic `HTTP/1.0 500 Server error` error, use the Web DataBlade Module Administration Tool to set the following Webdriver variable for your Webdriver configuration.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
show_exceptions	**MI_WEBSHOWEXCEPTIONS**	No	Use the Web DataBlade Module Administration Tool to set the **show_exceptions** variable to on or `off`. When on, Webdriver displays the database exception returned by **WebExplode()**. When `off`, Webdriver displays the `HTTP/1.0 500 Server error` message. Default is `off`.

Managing Cookies

Use the Web DataBlade Module Administration Tool to set the following Webdriver variable to specify the cookies that Webdriver recognizes.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Description
accept_cookie	**MI_WEBACCEPTCKI**	No	Use the Web DataBlade Module Administration Tool to set the **accept_cookie** Webdriver variable to the name of cookies that your Web DataBlade module application uses. All other cookies are ignored by Webdriver. Multiple cookie names are separated by commas. If you do not use this variable, Webdriver assumes all cookies in the browser are part of the Web application.

Uploading Client Files

Use the Web DataBlade Module Administration Tool to set the following Webdriver variable to upload client files.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
upload_directory	**MI_WEBUPLOADDIR**	No	Directory on the Web server machine in which uploaded files are placed Default is `/tmp`.

Passing Image Map Coordinates

Set the MImap 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`.

Two-Pass Query Processing

Use the Web DataBlade Module Administration Tool to set the following Webdriver variable to specify that Webdriver execute a query in two parts.

Variable	Mandatory?	Description
MIqry2pass	No	Specifies a query to be executed in two parts **MIqry2pass** selects an object and then executes a function. Used only in a URL. Default is set to `OFF`.

Using RAW Mode with Webdriver

To enable RAW mode, use the Web DataBlade Module Administration Tool to set the following Webdriver variable in your Webdriver configuration.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
raw_password	**MI_RAWPASSWORD**	Yes	Password to enable RAW mode

Caching Information in the wbExtensions Table

Use the Web DataBlade Module Administration Tool to set the **extensions** Webdriver variable to control whether Webdriver caches the information contained in the **wbExtensions** table.

When you are developing AppPages (and possibly adding new extension types to the **wbExtensions** table) you do not want Webdriver to cache this information, but instead to retrieve it each time it is needed, in case the information has changed.

After you deploy your application to a production environment, improve performance by caching the information in the **wbExtensions** table. To enable extension information caching, set the **extensions** Webdriver variable to cache.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
extensions	**MI_WEBEXTENSIONS**	No	Default setting is nocache. Set to cache to enable caching.

Using the New AppPage Builder or Data Director for Web

If you use the version of AppPage Builder included in Version 4.12.UC1, or later, of the IBM Informix Web DataBlade module, or you use Data Director for Web, to develop AppPages, you must set the **schema_version** Webdriver variable to the value wb. This Webdriver variable is automatically included in the **apb2** and **ddw** Webdriver configurations when you install the Web DataBlade Module Administration Tool in your database.

If you use Version 3.32 or earlier of AppPage Builder to develop AppPages, you should *not* include the **schema_version** variable in your Webdriver configuration.

Variable	Name of Variable in Versions 3.3 and Previous	Mandatory?	Content
schema_version	**MI_WEBSCHEMADEF**	No	Set to wb if you are using Version 4.12.UC1 or later of the IBM Informix Web DataBlade module or Data Director for Web.

WebExplode() Variables

This section describes the **WebExplode()** variables. These variables are stored in the database as part of a Webdriver configuration.

Enabling WebExplode() Tracing

Use the Web DataBlade Module Administration Tool to set the following variables for your Webdriver configuration to enable logging of **WebExplode()** function trace information.

Variable	Mandatory?	Content
MI_WEBEXPLEVEL	Yes	Enables **WebExplode()** function tracing
MI_WEBEXPLOG	No	File to which **WebExplode()** messages are written

Managing Dynamic Tags

Use the Web DataBlade Module Administration Tool to set the following dynamic tag **WebExplode()** variables.

Variable	Mandatory?	Description
MI_WEBTAGSTABLE	No	Specifies the database table that the **WebExplode()** function searches for the body of a dynamic tag This variable can be set to the following two values: `webTags` or `wbTags`. The default value if this variable is not set is `webTags`. You *must* set the **MI_WEBTAGSTABLE** variable to `wbTags` in your Webdriver configuration if you developed your Web application using the APB application included in Version 4.0 or later of the Web DataBlade module or Version 2.0 of Data Director for Web.
MI_WEBTAGSSQL	No	Specifies a user-defined SELECT statement that the **WebExplode()** function runs to retrieve the body of a dynamic tag It is recommended that you *never* set the **MI_WEBTAGSSQL** variable in your Webdriver configuration. The variable should only be set for Web applications that were developed with Version 1.1 or earlier of Data Director for Web. The **MI_WEBTAGSTABLE** variable takes precedence over the **MI_WEBTAGSSQL** variable. This means that if you have both variables set in your Webdriver configuration, the **WebExplode()** function searches for the dynamic tag in the table specified by the **MI_WEBTAGSTABLE** variable.
MI_WEBTAGSCACHE	No	Specifies whether the **WebExplode()** function should cache dynamic tags or not This variable should be set to `on` to turn on caching or `off` to turn off caching. The default value is `on`. It is recommended that you turn off dynamic tag caching when you are developing your AppPages to ensure that you always see the latest version of the dynamic tag and not the cached version. When you deploy your application to a production environment, however, you should turn on dynamic tag caching to increase the performance of your Web application.

Limiting Loop Processing With the MIBLOCK Tag

Use the Web DataBlade Module Administration Tool to set the **MI_LOOP_MAX** variable for your Webdriver configuration to set the maximum number of loops executed when you use the FOR, FOREACH, and WHILE attributes of the MIBLOCK AppPage tag. During loop processing, if the maximum number of loops is reached, the **WebExplode()** function raises an exception and stops loop processing.

For example, assume you have set **MI_LOOP_MAX** to `100` in your Webdriver configuration and you execute the following AppPage:

```
<?MIBLOCK WHILE=1>
We are in an infinite loop
<?/MIBLOCK>
```

Although logically the previous MIBLOCK statement results in an infinite loop, processing stops as soon as 100 loops have been executed, and you receive an error in your AppPage.

Variable	Mandatory?	Content
MI_LOOP_MAX	No	Limits the number of loops executed when you use the FOR, FOREACH, and WHILE attributes of the MIBLOCK AppPage tag

Limiting The Number Of Times an AppPage Can Call Itself Recursively

Use the **MI_WEBEXPLODE_DEPTH** variable to set the maximum number of times an AppPage can call itself recursively. You recursively call an AppPage by explicitly executing the **WebExplode()** function on the AppPage with the MISQL tag.

Use the Web DataBlade Module Administration Tool to set the **MI_WEBEXPLODE_DEPTH** variable for your Webdriver configuration.

For example, assume you invoke the following AppPage called **/recurse.html**, passing it the *name*/*value* pair `$DEPTH=10`:

```
<?MIVAR NAME=DEPTH>$(-,$DEPTH,1)<?/MIVAR>
<?MIVAR>DEPTH : $DEPTH<?/MIVAR>

<?MISQL SQL="select WebExplode(object, 'DEPTH=$DEPTH') from wbpages
  where id = 'recurse' and path = '/' and extension = 'html'";> $1
<?/MISQL>
```

This AppPage calls itself recursively. If the **MI_WEBEXPLODE_DEPTH** variable has not been set, then the AppPage calls itself recursively until all database server resources have been used. If, however, you set **MI_WEBEXPLODE_DEPTH** to `100`, the AppPage calls itself 100 times, and then stops.

Variable	Mandatory?	Content
MI_WEBEXPLODE_DEPTH	No	Limits the number of times an AppPage can call itself recursively.

Appendix

C

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Ave
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

> © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

AIX; DB2; DB2 Universal Database; Distributed Relational Database Architecture; NUMA-Q; OS/2, OS/390, and OS/400; IBM Informix®; C-ISAM®; Foundation.2000™; IBM Informix® 4GL; IBM Informix® DataBlade® Module; Client SDK™; Cloudscape™; Cloudsync™; IBM Informix® Connect; IBM Informix® Driver for JDBC; Dynamic Connect™; IBM Informix® Dynamic Scalable Architecture™ (DSA); IBM Informix® Dynamic Server™; IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager); IBM Informix® Extended Parallel Server™; i.Financial Services™; J/Foundation™; MaxConnect™; Object Translator™; Red Brick Decision Server™; IBM Informix® SE; IBM Informix® SQL; InformiXML™; RedBack®; SystemBuilder™; U2™; UniData®; UniVerse®; wintegrate® are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

Glossary

anchor variable Variable in an AppPage whose value is based on the URL prefix used to invoke the AppPage. You do not set the anchor variable in your AppPage; rather, Webdriver automatically generates the value. You can use anchor variables to link one or more AppPages in the same Web application.

WEB_HOME is the Web DataBlade module anchor variable.

Apache Webdriver The implementation of Webdriver that uses the Apache API to connect to databases and execute AppPages.

See also *Webdriver*.

AppPage An HTML page that includes AppPage tags and functions that dynamically execute SQL statements to query the database and format the results.

AppPage Builder (APB) A development tool packaged with the Web DataBlade module that allows you to create and update AppPages. APB is itself a Web DataBlade module application made up of linked AppPages.

AppPage tags Tags that are provided with the Web DataBlade module and are processed by the **WebExplode()** function. The tags identify elements of an HTML page and specify the structure and formatting for that page.

CGI Webdriver The implementation of Webdriver that uses a CGI program to connect to databases and execute AppPages.

See also *Webdriver*.

code set

A set of unique bit patterns that are mapped to the characters contained in a specific natural language, which include the alphabet, digits, punctuation, and diacritical marks. There can be more than one code set for a language: for example, the code sets for the English language include ASCII, ISO8895-1, and Microsoft 1252. You specify the code set that your database server uses when you set the GLS locale.

See also *multibyte code set*, *Global Language Support (GLS)*, *locale*.

deployment

Moving a Web application from a development environment to a production environment.

directive

An entry in a Web server's configuration file, that identifies the steps in the Web server's request-response processes that handle HTTP transactions. Examples of directives in Netscape's **obj.conf** file are NameTrans and Service.

dynamic tag

An HTML tag that allows multiple AppPages to share AppPage segments. For example, a TITLE dynamic tag might contain a standard title AppPage segment common to all the AppPages that make up a particular Web application. Each AppPage then uses the same TITLE dynamic tag for its title.

See also *system dynamic tag*, *user-defined dynamic tag*.

Global Language Support (GLS)

An application environment that allows Informix application-programming interfaces (APIs) and database servers to handle different languages, cultural conventions, and code sets. Developers use the GLS libraries to manage all string, currency, date, and time data types in their code. Using GLS, you can add support for a new language, character set, and encoding by editing resource files, without access to the original source code, and without rebuilding the DataBlade module or client software.

INFORMIXDIR

The Informix environment variable that specifies the directory in which Informix products are installed.

INFORMIX-SERVER

The Informix environment variable that specifies the name of the Informix database server to which you want to connect.

ISAPI Webdriver

The implementation of Webdriver that uses the Microsoft Windows NT Internet Information Server API to connect to databases and execute AppPages.

See also *Webdriver*.

large object A data object that exceeds 255 bytes in length. A large object is logically stored in a table column but physically stored independently of the column, because of its size. Large objects can contain non-ASCII data.

locale A set of files that define the native-language behavior of the program at run-time. The rules are usually based on the linguistic customs of the region or the territory. The locale can be set through an environment variable that dictates output formats for numbers, currency symbols, date, and time as well as collation order for character strings and regular expressions.

See also *Global Language Support (GLS)*.

MI_DRIVER_ERROR A variable, accessible in AppPages, that contains a description of a Web-driver error. By querying the contents of this variable, an error-handling AppPage can determine the exact error that occurred and take appropriate action.

MI_WEBCONFIG An Web DataBlade module environment variable that contains the full pathname of the **web.cnf** file. This variable is used by the NSAPI, ISAPI, and Apache implementations of Webdriver to locate the file when they create connections to an Informix database server.

multibyte code set A code set that is made up of both single-byte and multibyte characters. Examples of multibyte code sets are EUC and Shift JIS.

See also *code set*.

multirepresentational data type A data type whose storage location varies depending on the size of the data.

The Web DataBlade module HTML data type is an example of a multirepresentational data type. The first 7500 bytes of the HTML object are stored in the row; any portion of the HTML object that exceeds 7500 bytes is stored as a smart large object.

NSAPI Webdriver The implementation of Webdriver that uses the Netscape API to connect to databases and execute AppPages.

See also *Webdriver*.

ONCONFIG file The file that contains parameters for configuring the Informix database server. An example of a parameter in the **ONCONFIG** file is **SBSPACENAME**.

processing variable

A variable in an AppPage that contains processing information about the execution of an SQL statement, such as the number of rows or columns returned from a SELECT statement. An AppPage accesses processing variables after an MISQL tag executes its SQL statement.

RAW mode

A way to display an AppPage stored in the database without expanding the AppPage tags. You can also display the variables in an AppPage and identify where variable assignments are made. RAW mode is useful for debugging.

sbspace

A logical storage area that contains one or more chunks that store only smart large object data.

server-side includes

A mechanism for including dynamic text in AppPages. Server-side includes are special command codes that are recognized and interpreted by the Web server; their output is placed in the AppPage before the AppPage is sent to the browser. Server-side includes can be used, for example, to include a date or time stamp in the text of the AppPage.

smart large object

A large object that:

- Is stored in an sbspace, a logical storage area that contains one or more chunks
- Has read, write, and seek properties similar to a UNIX file
- Is recoverable
- Obeys transaction isolation modes
- Can be retrieved in segments by an application

Smart large objects include CLOB and BLOB data types.

sqlhosts file

An Informix file that contains information that lets a client application locate and connect to an Informix database server anywhere on a network.

system dynamic tag

Dynamic tags provided by the Web DataBlade module that allow you to reuse existing HTML to simplify the construction and maintenance of Web applications. Examples of system dynamic tags are CHECKBOXLIST, RADIOLIST, and SELECTLIST.

See also *dynamic tag*, *user-defined dynamic tag*.

UDR tag

See *user-defined routine tag*.

user-defined routine tag

Tag in an AppPage that directly executes an existing user-defined routine and places the output of the execution of the routine within the AppPage.

URL prefix

Part of a URL that client applications send to the Web server to invoke HTML pages, execute CGI programs (such as the CGI Webdriver,) call Web server plug-ins (such as the NSAPI, Apache, or ISAPI Webdriver,) and so on. The Web server interprets the URL prefix to perform the appropriate action depending on how you have configured your Web server.

user-defined dynamic tag

A dynamic tag you create to reuse existing HTML to simplify the construction and maintenance of Web applications.

See also *dynamic tag*, *system dynamic tag*.

variable expression

An expression in an AppPage that starts with a $ character followed by a variable-processing function and two or more variables within parentheses. For example, the variable expression $(+,$NUMA,$NUMB) adds the two variables $NUMA and $NUMB.

See also *variable-processing function*.

variable-processing function

An AppPage function used in a variable expression to evaluate or manipulate variables. For example, the variable-processing function "+" in the variable expression $(+,$NUMA,$NUMB) adds the two variables $NUMA and $NUMB.

See also *variable expression*.

variable vector

A set of variables with the same name that are passed into the AppPage using check boxes or the MULTIPLE attribute of selection lists.

virtual processor

One of the multithreaded processes that make up the Informix database server and are similar to the hardware processors in the computer. For example, in the Web DataBlade module, you must add a WEB virtual processor to use the MIEXEC tag in an AppPage.

walking window

Two or more linked AppPages in which each AppPage displays a subset of the entire set of rows returned from a SELECT statement. You can navigate through the set of returned rows by clicking buttons on the AppPages.

web.cnf file

The default name of the Webdriver configuration file that describes the connection between the Web server and the Informix database server.

Web DataBlade Module Administration Tool

A Web DataBlade module application used to add, update, or delete Webdriver mappings and Webdriver configurations for the database to which you are connected.

Webdriver

A client application that connects to an Informix database, at the request of a Web server, and retrieves AppPages from a table. Webdriver passes the retrieved AppPage to the **WebExplode()** function and returns the resulting HTML to the Web server.

See also *Apache Webdriver*, *CGI Webdriver*, *ISAPI Webdriver*, *NSAPI Webdriver*.

Webdriver configuration

The name given to a set of Webdriver variables and user-defined variables associated with a particular Web DataBlade module application. Webdriver configurations are stored in the **WebCMConfigs** system table in the database.

Webdriver configuration file

See *web.cnf file*.

Webdriver mapping

The name given to the set of Webdriver variables in a single Map section of the **web.cnf** file that Webdriver uses to connect to a particular database. Webdriver mappings have the same name as the corresponding URL prefixes defined for a Web server and the Web DataBlade module application stored in a database.

See also *URL prefix*.

Webdriver variable

A variable that Webdriver uses to connect to a database and to obtain information about a Web DataBlade module application. There are two types of Webdriver variables: those that reside in the **web.cnf** file (collectively known as Webdriver mappings) and those that reside in the database (collectively known as Webdriver configurations).

See also *Webdriver mapping*, *Webdriver configuration*.

WebExplode() function

An Informix database server function that builds dynamic HTML pages based on data stored in a database. The **WebExplode()** function parses AppPages that contain AppPage tags and dynamically builds and executes the SQL statements embedded in the tags. The **WebExplode()** function returns the HTML page to the client application, usually Webdriver.

Index

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W