INFORMIX-Data Director for Web

Programmer's Guide

Version 1.1
March 1998
Part No. 000-5034

Published by INFORMIX® Press

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

ACKNOWLEDGMENTS

Documentation Team: Kim Bostrom, Sandra Farkas, Jan Strother

Contributors: John Gaffney, Scott Grant, Daniel Howard, Paul Jasper, Karin Moore, Simon Morgan, Bob Nowacki, Sean Riley, Simon So, Scott Stark

Table of Contents

Introduction

In This Chapter

This chapter introduces the *INFORMIX-Data Director for Web Programmer's Guide*. Read this chapter for an overview of the information provided in this manual and for an understanding of the conventions used throughout.

About This Manual

The *INFORMIX-Data Director for Web Programmer's Guide* explains how to use the Data Director for Web Application Programming Interface (Data Director for Web API). Using the Data Director for Web API, you design applications that manage AppPages (HTML pages that include AppPage tags to dynamically execute SQL statements and format the results) and Web content stored in the Informix Dynamic Server database.

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 Data Director for Web API.

Organization of This Manual

This manual includes the following chapters:

- This introduction provides an overview of the manual, describes the documentation conventions used, and explains the generic style of this documentation.
- Chapter 1, "Overview of the Data Director for Web API," provides an overview of the architecture and functionality of the Data Director for Web API.

- Chapter 2, "Using the Data Director for Web API," provides examples of common tasks you can program using the Data Director for Web API.
- Chapter 3, "Data Director for Web API Class Reference," provides reference information about classes and methods in the Data Director for Web API.
- Chapter 4, "Error Handling and Troubleshooting," provides information about error handling in the Data Director for Web API. It also provides information to help you troubleshoot problems you might encounter while developing an application using the API.
- Appendix A, "Data Director for Web Schema Definitions," provides the schema definitions for the Data Director for Web API.
- Appendix B, "Data Director for Web API Examples," provides the full code for the examples in Chapter 2, "Using the Data Director for Web API."

A comprehensive index directs you to areas of particular interest.

Types of Users

To use this product, you should be familiar with the following products and technologies:

- Data Director for Web
- Web Integration Option (Informix Web DataBlade module or INFORMIX-Universal Web Connect)
- The C++ programming language and the Microsoft Foundation Classes (MFC)
- Object-oriented concepts and programming techniques

Documentation Conventions

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

The following conventions are covered:

- Typographical conventions
- Icon conventions

Typographical Conventions

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

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
italics	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
`monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
♦	This symbol indicates the end of product- or platform-specific information.

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

Icon Conventions

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

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

Additional Documentation

The Data Director for Web documentation set includes printed manuals, on-line manuals, and on-line help.

This section describes the following parts of the documentation set:

- Printed documentation
- On-line documentation

Printed Documentation

The following printed manuals are included in the Data Director for Web documentation set:

- *INFORMIX-Data Director for Web User's Guide*. This manual introduces Data Director for Web, including the development environment and features. It explains how to use Data Director for Web to build Informix-based Web applications and manage Informix-based Web sites.
- *INFORMIX-Data Director for Web Programmer's Guide*. This manual explains how to use the Data Director for Web Application Programming Interface (Data Director for Web API) to design applications that manage AppPages (HTML pages that include AppPage tags to dynamically execute SQL statements and format the results) and Web content stored in the Informix Dynamic Server database.

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

- If you have never used Structured Query Language (SQL), read the *Informix Guide to SQL: Tutorial*. It provides a tutorial on SQL as it is implemented by Informix products. It also describes the fundamental ideas and terminology for planning and implementing a relational database.
- A companion volume to the *Tutorial*, the *Informix Guide to SQL: Reference*, includes details of the Informix system catalog tables, describes Informix and common environment variables that you should set, and describes the column data types that Informix database servers support.
- *Informix Web DataBlade Module User's Guide*. This manual introduces the Web DataBlade module and explains how to use the Web DataBlade module and its associated features to develop applications that dynamically retrieve data from the Informix Dynamic Server with Universal Data Option database.

Informix Error Messages might also be useful if you do not want to look up your error messages on-line.

On-Line Documentation

Different types of on-line documentation are available:

- Universal Web Connect guides
- On-line help
- Release notes and documentation notes

Universal Web Connect Guides

Universal Web Connect on-line guides introduce Universal Web Connect and explain how to use Universal Web Connect and its associated features to develop applications that dynamically retrieve data from the Informix Dynamic Server database.

On-Line Help

The Data Director for Web on-line help facility provides a detailed, context-sensitive explanation of program features. Use it to explore each menu item. To invoke the on-line help, press `F1` whenever you are using any Data Director for Web tool.

Release Notes and Documentation Notes

In addition to the Informix set of manuals, the following on-line files supplement the information in this manual.

On-Line File	Purpose
Release notes	Describes any special actions required to install, configure, and use Data Director for Web on your computer. Additionally, this file contains information about any known problems and their workarounds.
Documentation notes	Describes features not covered in the manuals or modified since publication.

On-line files are located in the **$INFORMIXDIR\ddw** directory.

Informix Welcomes Your Comments

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

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

Write to us at the following address:

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

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

doc@informix.com

Or, send a facsimile to Technical Publications at:

650-926-6571

We appreciate your feedback.

Chapter

1

Overview of the Data Director for Web API

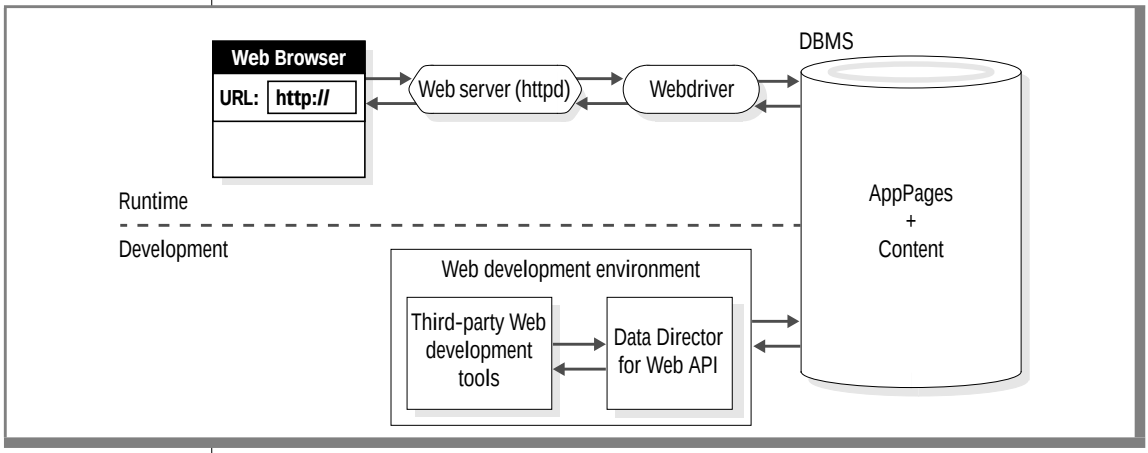
In This Chapter

The Data Director for Web Application Programming Interface (Data Director for Web API) is a Microsoft Foundation Classes (MFC) extension library designed to enable client applications to access AppPages and Web content stored in an Informix database under the Data Director for Web schema. You interface with the database through the API without constructing SQL statements or having a specific knowledge of the schema.

Architecture

The Data Director for Web API models Web site content stored in the Data Director for Web schema using C++ classes. You can view the API as a high-level abstraction layer, containing C++ objects that model data in the schema, interacting with a hidden connectivity layer that manages communication to the database. Figure 1-1 on page 1-3 illustrates the API database interaction model.

Figure 1-1
Informix-Based Web Development Environment



API Objects

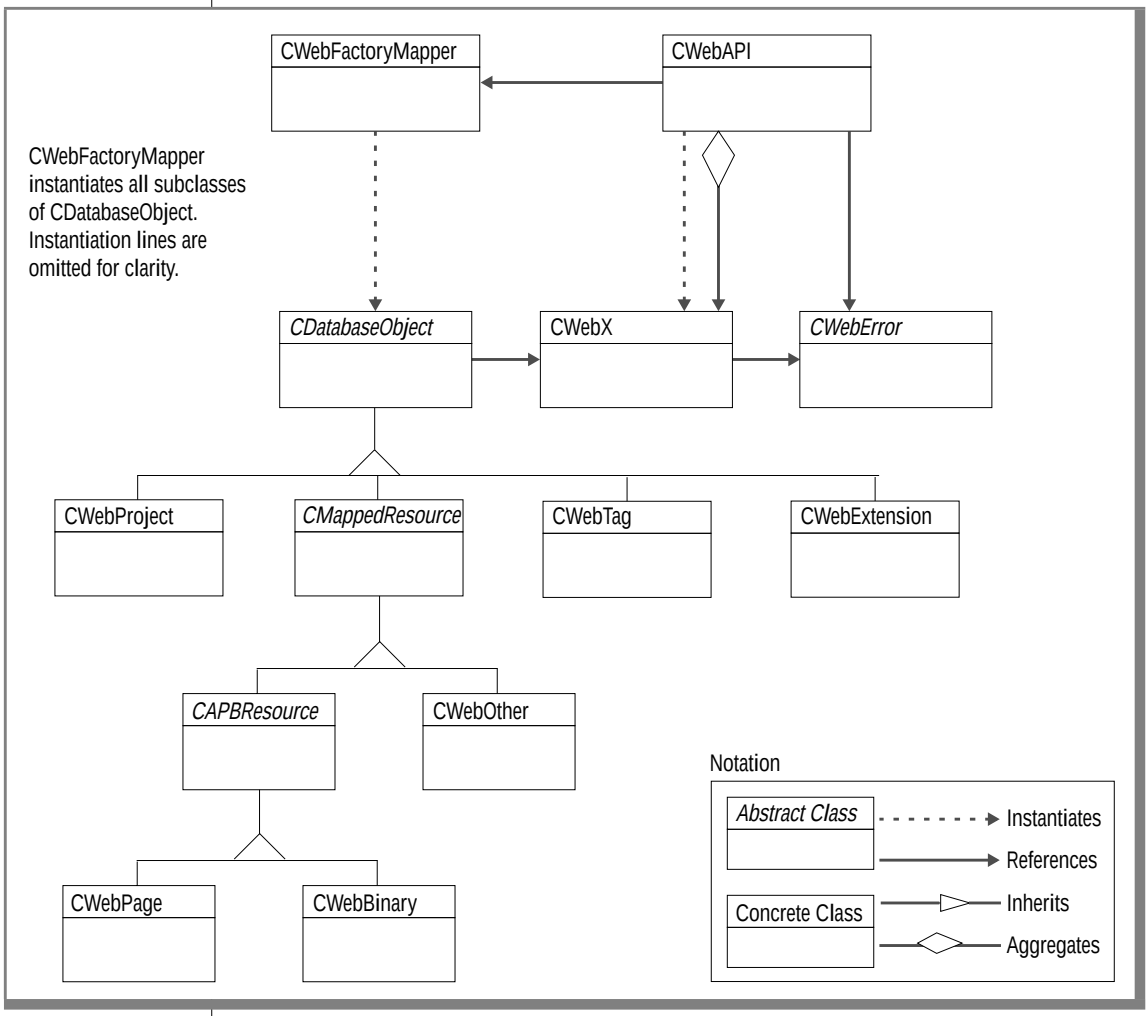
The API provides a complete object model of all tables and columns defined in the Data Director for Web schema. Tables in the Data Director for Web schema are modeled using objects (C++ classes) in the API. Your application interacts with the database through these objects. You modify column values by invoking the methods defined for each object and committing the object to the database.

The following objects are modeled in the API:

- **Resources.** Web site content, such as AppPages and images, are *resources*. A resource is modeled in the API using a subclass of the **CDatabaseObject** abstract base class. The built-in resources are:
  - **AppPage.** An *AppPage* is an HTML page that can contain Web Integration Option tags, known as *AppPage tags*. By default, an AppPage is stored in the **wbPages** table and modeled in the API using the **CWebPage** class.
  - **Binary.** A *binary* is an object like an image or an audio clip. By default, a binary is stored as a binary large object in the **wbBinaries** table and modeled in the API using the **CWebBinary** class.
  - **Dynamic tags.** A *dynamic tag* is a dynamically expanded AppPage fragment that can be easily shared among multiple AppPages. Dynamic tags are stored in the **wbTags** table and modeled in the API using the **CWebTag** class.
  - **Other.** This resource describes all objects not stored in the Data Director for Web tables. The resource is stored as a text or binary object in a user table and the source table information is stored in the **wbExtensions** table. The resource is modeled using the **CWebOther** class. A resource of this type cannot belong to a project.
- **Projects.** A *project* is a collection of related resources—such as AppPages, images, and dynamic tags—associated with a particular Web application. A resource can belong to one or more projects, but is not required to belong to any project. The information specific to a given project is stored in the **wbProjects** table and modeled in the API using the **CWebProject** class.
- **Extensions.** An *extension* specifies mapping information for resources. You add a new MIME type by defining an extension for it, including the name of the table in which the objects reside. Generally, objects should reside in the **wbPages** and **wbBinaries** tables, but existing database tables with resource content can also be used. Tables other than **wbPages** and **wbBinaries** can be associated with only a single extension. Extensions are stored in the **wbExtensions** table and modeled in the API using the **CWebExtension** class.

Figure 1-2 on page 1-6 illustrates the Data Director for Web API class hierarchy.

Figure 1-2
Data Director for Web API Class Hierarchy



Using the API, you can add, delete, modify, and list projects and resources associated with an Informix-based Web site within a client application. For example, your client application can:

- create a new AppPage.

- associate an AppPage with a project.
- modify the attributes of the AppPage.

Database Connectivity

The Data Director for Web API supports TCP/IP connectivity through LIBDMI (for the Web DataBlade module) or ESQL/C (for Universal Web Connect).

A database connection is made when you invoke the **Connect** method. The database can be a local or remote connection to the client. You provide a database server name, database name, user name, and password. The database connection is maintained for the life of the API instance and is closed when the instance is destroyed.

Each instance of the API maintains only one database connection. You can implement multiple connections to different databases by creating multiple instances of the API.

Database Model

Null values are expressed using the string constant `NULL`.

Key data, or columns that are defined to be primary keys in a table, are object attributes that can be set at object creation but are read-only thereafter. You cannot change key attributes of an object once the object is created.

Associations between tables are stored as any other attribute, inside an object. You can update these associations with the appropriate methods; however, you must ensure that associative attributes are updated with values that represent associations to valid objects.

Memory Management

You must manage memory in certain situations. In C++ terms, this refers to object creation (using the **new** operator) and object destruction (using the **delete** operator).

Using the API, you manage memory in the following three situations:

- Creating an instance of the API
- Managing lists and objects in lists
- Creating new API objects

Creating an Instance of the API

The first step when using the API is to create a new instance of the API class **CWebAPI** using the class constructor:

```
CWebAPI(
        CWebFactoryMapper* factory,
        CWebError* errorHandler,
        const CString& LOCacheDirectory);
```

The constructor takes pointers to the objects **CWebFactoryMapper** and **CWebError** as two of its arguments. **CWebAPI** does not delete these classes when it is destroyed. You must destroy all instances of **CWebFactoryMapper** and **CWebError** when destroying **CWebAPI** (or a derived subclass).

You can use the same instance of **CWebFactoryMapper** for multiple instances of the API. This is not true for the error handler, **CWebError**. You must create a new instance of **CWebError** (or its derived subclass) for each instance of **CWebAPI**.

Managing Lists and Objects in Lists

To retrieve a list of objects from the database, you invoke a method on **CWebAPI**, passing a pointer to a list (**CPtrList ***). The API populates the list with new instances of API objects. You must create and destroy the list.

You are also responsible for the objects in the list. A helper method has been provided on the API that destroys all objects in a list. The method on the class **CWebAPI** looks like:

```
static void DestroyContentOfList(CPtrList* list)
```

Important: *Exercise care when storing list object pointers in local variables. If you destroy the list, the local variables point to objects that no longer exist.*

Creating New API Objects

When you create a new instance of an API object using one of the API object-creation routines, you must destroy the object (using the **delete** operator) when it is no longer needed:

```
// Delete a project.

if (!newProject->Delete())
{
        // Perform some error processing.
}

// Now that processing is completed, destroy the
// project object.

delete newProject;
```

Destroying an object does not remove it from the database. To remove an object from the database, invoke the **Delete** method on the object.

Chapter

2

Using the Data Director for Web API

In This Chapter

This chapter provides examples of some of the more common tasks you perform using the Data Director for Web API. The examples are designed to be generic, allowing you to extend them to meet your needs. For the sake of narrative, longer examples are broken into smaller code fragments. You can view the examples in their entirety in Appendix B.

Database Connectivity

This section provides examples of connecting to and disconnecting from a database, and of retrieving database and server information.

Connecting to a Database

Connecting to a database is a three-step process:

1. Instantiate the necessary objects.
2. Instantiate the API.
3. Connect to the database.

Instantiating the Necessary Objects

To create an instance of the API, you must pass in instances of an error-handler class and a factory-mapper class:

```
// Create an instance of an error-handler class.

error = new CWebError();

// Create an instance of a factory-mapper class.

factory = new CWebFactoryMapper();
```

Instantiating the API

Having created the necessary objects, you can instantiate the API. In addition to the error-handler and factory-mapper classes, you must also declare the location of the Data Director for Web page-caching directory:

```
// Create a new instance of the API with the factory-mapper
// and error-handler classes.  Also, specify the cache
// directory as D:\APBDump.  Note the need to use double
// slashes to pass a single slash.

api = new CWebAPI(factory, error,"D:\\APBDump");
```

It is necessary to use a double backslash in the directory path for the parser to recognize a single backslash.

Opening the Connection

After the new instance of the API has been created, attempt to connect to the database, supplying the server name, database name, user name, and password:

```
// Connect to a database using the parameters:
//
// Server:  server
// Database:    db
// User:    foo
// Password:    guest

if ( ! api->Connect("server",
     "db",
     "foo","guest!"))
{
     TRACE0("Failed to connect to database db as foo \n");
     return ;
}
```

If the connection fails, perform the necessary error handling. **TRACE0** is a Microsoft built-in method that prints an error message to the screen.

Retrieving Database Connection Information

To retrieve database connection information, invoke the appropriate API methods:

```
// Retrieve the connection parameters.

CString server = api->GetServer();
CString database = api->GetDatabase();
CString user = api->GetUser();

TRACE0("user " + user + " is connected to server:" +
     server + " and database:" + database + "\n");
```

Disconnecting from a Database

To disconnect from a database, you destroy the instance of the API. Once the API instance has been destroyed, the factory-mapper and error-handler instances can also be destroyed.

```
// Delete the API instance.

delete api;

// Delete the error-handler instance.

delete error;

// Delete the factory-mapper instance.

delete factory;
```

It is important to destroy objects once they are no longer necessary, to avoid memory leaks in your application.

Managing Resources

This section contains examples of common tasks that you perform on Data Director for Web resources. The examples here use projects; however, these examples are extended in Appendix B to apply to extensions, dynamic tags, and AppPages.

Managing Project Lists

There are three basic steps to manipulating a project list:

1. Create the list.
2. Populate the list.
3. Process the list.

After you have processed the list, you must destroy the list to release the memory.

Creating and Populating the List

First, create the new list object, then populate it using the API method **GetWebProjects**:

```
// Create a new, empty project list.
CPtrList* projects = new CPtrList();

// Fetch the projects from the database.
if (! api->GetWebProjects(projects))
{
    // Perform some error handling.
}
```

The list is populated with project objects representing all of the projects stored in the database.

Processing the List

After you have retrieved the projects, you can traverse the list to process it. Begin by finding the first project in the list, using the **GetHeadPosition** method, then iterate through the list, using the **GetNext** method:

```
// Loop through the projects and do some processing.

POSITION pos = projects->GetHeadPosition();
while( pos != NULL )
{
    CWebProject* project = (CWebProject*) projects->GetNext( pos );

    // Do some processing.
}
```

Destroying the List

After you have finished processing the projects in the list, destroy the objects in the list and then destroy the list to release the memory:

```
// Finished processing projects, so destroy the project
// objects and the projects list.

CWebAPI::DestroyContentOfList(projects);
delete projects;
```

Creating a Project

Creating a new Data Director for Web resource not only creates an instance of the resource but also inserts a new row, containing the minimum necessary information for that resource, into the database. This ensures that an existing API object corresponds to a row in the database. For a project, the minimum required attribute is the project name.

To create a new project, invoke the **CreateWebProject** method with a project name:

```
// Create a new project.

CWebProject* newProject = api->CreateWebProject("NEWPROJECT");

if (newProject == NULL)
{
    // Perform some error processing.
}
```

Updating a Project

After you have created a project, update it with information not required at creation by invoking the appropriate methods to set the information in the project object:

```
// Modify a project.

newProject->SetDescription("modified description");
newProject->SetOwner();
newProject->SetLastDeployedAndDeployedBy();
newProject->SetDeployedDb("modified db");
newProject->SetDeployedServer("modified server");
newProject->SetDeployedProject("modified project");
```

Use the **Update** method to commit the information to the database:

```
// Commit the changes to the database.

if (!newProject->Update())
{
    // Perform some error processing.
}
```

If **Update** fails, the updated row is not written to the database.

Deleting a Project

To remove a row from the database, use the **Delete** method, then destroy the instance of the local object:

```
// Delete a project.

if (!newProject->Delete())
{
    // Perform some error processing.
}
```

After the local instance is destroyed, to avoid memory leaks, remember to destroy all other objects that are no longer necessary:

```
// Now that processing is completed, destroy the
// project object.

delete newProject;
```

Locking and Unlocking a Project

You can lock a project to prevent other users from deleting or updating its resources. To lock a project, invoke the **Lock** method:

```
// Lock a project.

if (!newProject->Lock())
{
    // Perform some error processing.
}
```

To unlock the project and allow delete / update operations, invoke the **Unlock** method:

```
// Unlock the project.

if (!newProject->Unlock())
{
    // Perform some error processing.
}
```

Executing Web Integration Option Functions

Web Integration Option functions allow you to perform a variety of useful tasks. The following examples show how to invoke these functions from within the API. For further information on the functions themselves, refer to *Informix Web DataBlade Module User's Guide* or the Universal Web Connect online documentation.

EncodeWebConfFile

The method **EncodeWebConfFile** returns the contents of the Web Integration Option configuration file as a string, with all nonalphanumeric characters replaced with their ASCII hexadecimal representation. This method is commonly used to pass the contents of the Web DataBlade module **web.cnf** file to the **ExecuteWebExplode** method:

```
// EncodeWebConfFile
TRACE0("Encoding conf file d:\\web.conf\n");
CString env = api->EncodeWebConfFile("D:\\Web.conf");
```

It is necessary to use a double backslash in the directory path for the parser to recognize a single backslash.

Tip: *It is not necessary to invoke the* ***EncodeWebConfFile*** *method prior to invoking the* ***ExecuteWebExplode*** *method if you are using Universal Web Connect.*

ExecuteWebExplode

The **WebExplode** function expands AppPage tags and dynamically retrieves SQL results. The method **ExecuteWebExplode** runs the **WebExplode** function:

```
CWebPage* page;

page = (CWebPage *)api->GetWebResource("foo", "/", "html");

// ExecuteWebExplode


CString explodedPage = api->ExecuteWebExplode(page->GetContentFile(),env);
```

For the Web DataBlade module, **WebExplode** retrieves its environment from the Web Integration Option configuration file, **web.cnf**. Therefore, ***env*** contains the results of the **EncodeWebConfFile** method from the preceding page.

For Universal Web Connect, the configuration file values are already available in the **WebExplode** environment, so ***env*** can be an empty string or can contain additional user-supplied name/value pairs to **WebExplode**.

ExecuteWebRelease

The **WebRelease** function returns the version string of the Web Integration Option. The method **ExecuteWebRelease** returns the version string to the application:

```
// ExecuteWebRelease
CString release = api->ExecuteWebRelease();
TRACE0("Web release version:" + release + "\n" );
```

ExecuteWebUnHTML

The **WebUnHTML** function returns a copy of the input string with all special HTML characters replaced with their entity references for display by a Web browser. The method **ExecuteWebUnHTML** runs the **WebUnHTML** function:

```
// ExecuteWebUnHTML

CString unhtmledPage = api->ExecuteWebUnHTML(page->GetContentFile());
```

ExecuteWebURLEncode

The **WebURLEncode** function takes a Uniform Resource Locator (URL) and returns it as an ASCII string with all nonalphanumeric characters replaced with their ASCII hexadecimal representations. The **ExecuteWebURLEncode** method is invoked directly on the URL:

```
// ExecuteWebURLEncode
CString src = "this is a (test) of {URL} *encoding*!!";

TRACE0("Encoding string" + src + "\n" );
CString encodedsrc = api->ExecuteWebURLEncode(src);
TRACE0("Encoded string:" + encodedsrc + "\n" );
```

SplitResourceName

The method **SplitResourceName** takes the full pathname of a resource and separates it into its constituent identifier, path, and extension:

```
// Split a resource into its component parts.

CString ID;
CString path;
CString ext;


CString resource = "/runtime/first.html";
api->SplitResourceName(resource,&path,&ID,&ext);
TRACE0("Resource:" + resource + " path:" + path + " ID:" + ID
+ " ext:" + ext + "\n");
```

These component parts can be used, for example, to insert a new resource into the Data Director for Web schema.

Tip: *A Data Director for Web API resource pathname does not use a trailing slash.*

Chapter

3

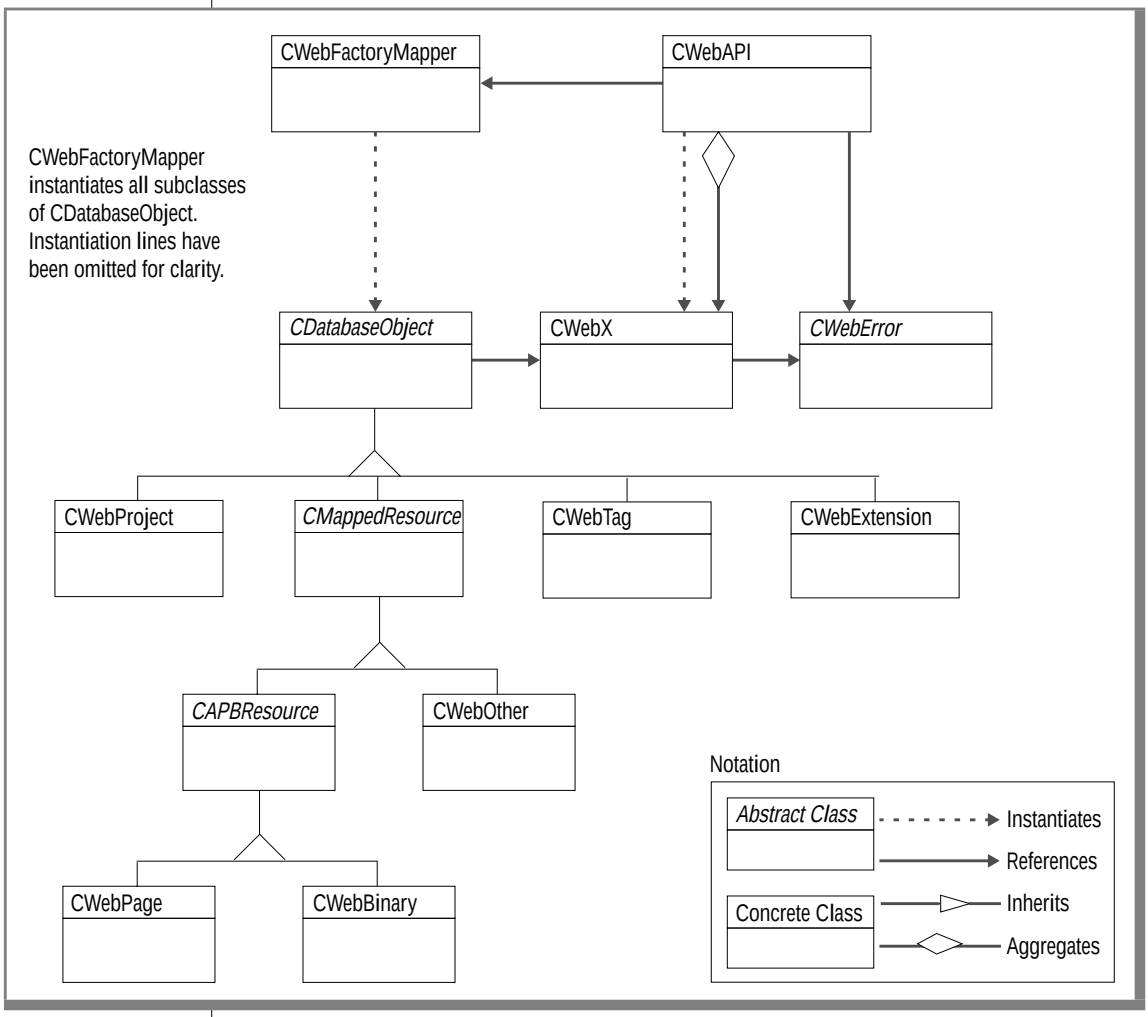
Data Director for Web API Class Reference

In This Chapter

This chapter describes the classes provided by the Data Director for Web API. Attributes have been omitted because all attributes are private and cannot be directly accessed by client applications.

Figure 3-1, previously seen on page 1-6, illustrates the Data Director for Web API class hierarchy.

Figure 3-1
Data Director for Web API Class Hierarchy



CWebAPI

The **CWebAPI** class is the main controlling object through which most database access takes place. This class must be instantiated before a database connection can be made.

Public Constructors

The constructor for the **CWebAPI** class instantiates the API:

```
CWebAPI(
        CWebFactoryMapper* factory,
        CWebError* errorHandler,
        const CString& LOCacheDirectory);
```

To open a connection to a database, invoke the **Connect** method.

Public Interface

The following are the public methods on the **CWebAPI** class.

AddAPBResourceToProject

```
BOOL AddAPBResourceToProject(const CWebProject& project,
                             const CAPBResource& resource);
```

Adds the Data Director for Web resource *resource* to *project*. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

AddWebTagToProject

```
BOOL AddWebTagToProject(const CWebProject& project,
                             const CWebTag& tag);
```

Adds the dynamic tag *tag* to *project*. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

CacheExtensionMappings

```
BOOL CacheExtensionMappings();
```

Refreshes the internal cache of extension objects with the extensions stored in the database.

Connect

```
BOOL Connect(const CString& server,
             const CString& database,
             const CString& user,
             const CString& password);
```

Invoked on a **CWebAPI** object, **Connect** opens a connection to a database, using the parameters supplied. Returns FALSE if the connection fails; otherwise returns TRUE.

CreateWebExtension

```
CWebExtension* CreateWebExtension(const CString& extension);
```

Creates and returns a new extension object called *extension*.

CreateWebProject

```
CWebProject* CreateWebProject(const CString& name);
```

Creates and returns a new project object called *name*.

CreateWebResource

```
CMappedResource* CreateWebResource(const CString& ID,
                         const CString& path,
                         const CString& extension);
```

Creates and returns a new resource object identified by *ID*, *path*, and *extension*.

CreateWebTag

```
CWebTag* CreateWebTag(const CString& ID);
```

Creates a new dynamic tag object identified by *ID*.

DestroyContentOfList

```
static void DestroyContentOfList(CPtrList* list);
```

A static method that removes all API objects from *list*.

EncodeWebConfFile

```
CString EncodeWebConfFile(const CString& WebConfFileName);
```

Returns the URL-encoded contents of the Web Integration Option configuration file *WebConfFileName*. This method is most commonly used to pass the contents of a **web.cnf** file to **WebExplode** for the Web DataBlade module.

ExecuteWebExplode

```
CString ExecuteWebExplode(const CString& localFileName,
                          const CString& variables);
```

Runs **WebExplode** over the contents of *localFileName*. Returns the exploded (expanded) HTML.

ExecuteWebLint

```
CString ExecuteWebLint(const CString& localFileName,
                       int level);
```

Runs **WebLint** over the contents of *localFileName* with an error-checking level of *level*. Returns text describing any errors encountered.

ExecuteWebRelease

```
CString ExecuteWebRelease();
```

Returns Web Integration Option release information to the client.

ExecuteWebUnHTML

```
CString ExecuteWebUnHTML(const CString& localFileName);
```

Returns the contents of the file *localFileName* with all special HTML characters replaced by their entity references.

ExecuteWebURLEncode

```
CString ExecuteWebURLEncode(const CString& text);
```

Returns the string *text* as a URL-encoded string.

GetAPBResources

```
BOOL GetAPBResources(const CWebExtension& extension,
                     CPtrList* resources);
```

Populates the *resources* list with Data Director for Web resources of the type *extension* stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

```
BOOL GetAPBResources(const CWebProject& project,
                     const CWebExtension& extension,
                     CPtrList* resources);
```

Populates the *resources* list with Data Director for Web resources of the type *extension* associated with *project*. Returns FALSE if an error occurs; otherwise returns TRUE.

```
BOOL GetAPBResources(CString& pathWildCard,
                     const CWebExtension& extension,
                     CPtrList* resources);
```

Populates the *resources* list with all Data Director for Web resources of the type *extension* having a path like *pathWildCard*. Returns FALSE if an error occurs; otherwise returns TRUE.

GetAPBResourcesNotInAnyProject

```
BOOL GetAPBResourcesNotInAnyProject(
                      const CWebExtension& extension,
                      CPtrList* resources);
```

Populates the *resources* list with all Data Director for Web resources of the type *extension* not associated with any project. Returns FALSE if an error occurs; otherwise returns TRUE.

GetConnection

```
const CWebX& GetConnection() const;
```

Returns the low-level connectivity object for the API.

GetDatabase

```
CString GetDatabase() const;
```

Returns the name of the current database.

GetErrorHandler

```
const CWebError* GetErrorHandler() const;
```

Returns the error-handling object for the API.

GetOtherResources

```
BOOL GetOtherResources(const CWebExtension& extension,
                       CPtrList* resources);
```

Populates a list with all non-Data Director for Web resources stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

```
BOOL GetOtherResources(const CString& pathWildCard,
                       const CWebExtension& extension,
                       CPtrList* resources);
```

Populates a list with all non-Data Director for Web resources having a path like *pathWildCard* stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

GetServer

```
CString GetServer() const;
```

Returns the name of the current database server.

GetSuperTypesForProject

```
BOOL GetSuperTypesForProject(CString& strProjName,
                             CStringArray& arrNames);
```

Populates the *arrNames* array with the names of all supertypes associated with the project called *strProjName*.

GetSuperTypesNotInProjects

```
BOOL GetSuperTypesNotInProjects(CStringArray& arrNames);
```

Populates the *arrNames* array with the names of all supertypes for extensions that are not associated with any project.

GetUser

```
CString GetUser() const;
```

Returns the name of the current database user.

GetWebExtension

```
CWebExtension* GetWebExtension(const CString& extension);
```

Returns an extension object for the specified extension. Returns NULL if an error occurs (for example, if the specified extension does not exist in the database).

GetWebExtensions

```
BOOL GetWebExtensions(CPtrList* extensions);
```

Populates the *extensions* list with pointers to all extensions stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

```
BOOL GetWebExtensions(const CString& superType,
                      CPtrList *extensions);
```

Populates the *extensions* list with pointers to all extensions of type *superType* stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

GetWebProject

```
CWebProject* GetWebProject(const CString& name);
```

Returns the project object for the supplied project name. Returns NULL if an error occurs (for example, if the specified project does not exist).

GetWebProjects

```
BOOL GetWebProjects(CPtrList* projects);
```

Populates the *projects* list with the names of all projects stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

GetWebResource

```
CMappedResource* GetWebResource(const CString& ID,
                                const CString& path,
                                const CString& name);
```

Returns a Web resource identified by *ID*, *path*, and *name*.

GetWebTag

```
CWebTag* GetWebTag(const CString& ID);
```

Returns the dynamic tag object for the supplied tag *ID*. Returns NULL if an error occurs (for example, if the specified dynamic tag does not exist).

GetWebTags

```
BOOL GetWebTags(CPtrList* tags);
```

Populates the *tags* list with all dynamic tags stored in the database. Returns FALSE if an error occurs; otherwise returns TRUE.

```
BOOL GetWebTags(const CWebProject& project,
                CPtrList* tags);
```

Populates the *tags* list with all dynamic tags associated with *project*. Returns FALSE if an error occurs; otherwise returns TRUE.

GetWebTagsNotInAnyProject

```
BOOL GetWebTagsNotInAnyProject(CPtrList* tags);
```

Populates the *tags* list with all dynamic tags not associated with any project. Returns FALSE if an error occurs; otherwise returns TRUE.

IsWebBlade

```
BOOL IsWebBlade();
```

Determines whether the current connection is to the Web DataBlade module or to Universal Web Connect. Returns FALSE if the connection is to Universal Web Connect; otherwise returns TRUE.

RemoveAPBResourceFromProject

```
BOOL RemoveAPBResourceFromProject(const CWebProject& project,
                        const CAPBResource& resource);
```

Removes the Data Director for Web resource *resource* from *project*. Returns FALSE if an error occurs; otherwise returns TRUE.

RemoveWebTagFromProject

```
BOOL RemoveWebTagFromProject(const CWebProject& project,
                        const CWebTag& tag);
```

Removes the dynamic tag *tag* from *project*. Returns FALSE if an error occurs; otherwise returns TRUE.

SplitResourceName

```
static void SplitResourceName(
                    const CString& fullResourceName,
                    CString* ID,
                    CString* path,
                    CString* extension);
```

Splits *fullResourceName* into its constituent *ID*, *path*, and *extension*.

CWebBinary

This class, derived from the **CAPBResource** class, defines the behavior for all Data Director for Web objects, excluding AppPages.

Public Interface

The following are the public methods on the **CWebBinary** class.

CanBeLocked

```
BOOL CanBeLocked();
```

Checks whether or not the **CWebBinary** object can be locked. A **CWebBinary** object cannot be locked if it is explicitly locked or if any project to which it belongs is locked. Returns `TRUE` if the **CWebBinary** object can be locked; otherwise returns `FALSE`.

Delete

```
virtual BOOL Delete();
```

Deletes the binary resource from the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

DeleteLeaveInProjects

```
virtual BOOL DeleteLeaveInProjects();
```

Deletes the binary resource from the database but does not remove its project associations from the **wbResProjects** table. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

This method is intended to be used when the binary resource is immediately replaced with another of the same identifier, extension, and path, where the existing project associations must be preserved.

GetContentFile

```
virtual CString GetContentFile();
```

Returns the filename of the binary resource content.

GetDescription

```
CString GetDescription() const;
```

Returns the description of the binary resource.

GetExtension

```
CString GetExtension() const;
```

Returns the extension of the binary resource.

GetHeight

```
int GetHeight() const;
```

Returns the height of the binary resource.

GetID

```
CString GetID() const;
```

Returns the identifier of the binary resource.

GetLastChanged

```
CString GetLastChanged() const;
```

Returns the date and time that the binary resource was last changed.

GetLastChangedBy

```
CString GetLastChangedBy() const;
```

Returns the name of the user who last changed the binary resource.

GetLastLocked

```
CString GetLastLocked() const;
```

Returns the date and time that the binary resource was last locked.

GetLastLockedBy

```
CString GetLastLockedBy() const;
```

Returns the name of the user who last locked the binary resource.

GetPath

```
CString GetPath() const;
```

Returns the path of the binary resource.

GetReadLevel

```
int GetReadLevel() const;
```

Returns the runtime security level for read access to the binary resource.

GetSize

```
long GetSize();
```

Returns the size, in bytes, of the **CWebBinary** object.

GetWidth

```
int GetWidth() const;
```

Returns the width of the binary resource.

Lock

```
BOOL Lock();
```

Locks the binary resource, preventing write access. Returns FALSE if an error occurs; otherwise returns TRUE.

Rename

```
CWebBinary *Rename(CWebBinary *newbin);
```

Creates a copy of a **CWebBinary** object in the newly created object *newbin*. After invoking **Rename**, you must explicitly invoke **Update** to save *newbin* to the database, and delete both the old and new **CWebBinary** objects. The behavior of any method called on the old **CWebBinary** object after **Rename** has been called is undefined.

SetContentFile

```
virtual void SetContentFile(const CString& localFileName);
```

Sets the content of the resource to the local file *localFileName*.

SetCurrentVersion

```
void SetCurrentVersion(int version);
```

Sets the version number of the binary resource to *version*.

SetDescription

```
void SetDescription(const CString& description);
```

Sets the description of the binary resource to *description*.

SetHeight

```
void SetHeight(int height);
```

Sets the height of the binary resource to *height*.

SetReadLevel

```
void SetReadLevel(int level);
```

Sets the runtime security level for read access to the binary resource to *level*.

SetWidth

```
void SetWidth(int width);
```

Sets the width of the binary resource to *width*.

Unlock

```
BOOL Unlock();
```

Releases the lock on the binary resource. Returns FALSE if an error occurs; otherwise returns TRUE.

Update

```
virtual BOOL Update();
```

Writes the binary resource object to the database. Returns FALSE if an error occurs; otherwise returns TRUE.

CWebError

The **CWebError** class is responsible for error handling in the API. To customize error handling, create a new error-handling class that inherits from this class.

Public Interface

The following are the public methods on the **CWebError** class.

LogMessage

```
virtual void LogMessage(const CString& object,
                        const CString& method,
                        const CString& message);
```

Displays a message box with the *object* and *method* in which the error occurred, and the error text *message*.

LogSQL

```
virtual void LogSQL(const CString &server,
                    const CString database,
                    const CString &user,
                    const CString &sql);
```

Called every time SQL is issued by an API object with which this **CWebError** object is associated.

CWebExtension

This class, derived from the **CDatabaseObject** class, defines the behavior for Web extensions.

Public Interface

The following are the public methods on the **CWebExtension** class.

Delete

```
virtual BOOL Delete();
```

Deletes the extension object from the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

GetContentColumn

```
CString GetContentColumn() const;
```

Returns the name of the column in which the extension content is stored.

GetExtension

```
CString GetExtension() const;
```

Returns the file extension associated with the extension object.

GetIDColumn

```
CString GetIDColumn() const;
```

Returns the name of the column in which the resource identifier for the extension is stored.

GetName

```
CString GetName() const;
```

Returns the name of the extension.

GetPathColumn

```
CString GetPathColumn() const;
```

Returns the name of the column in which the extension path is stored.

GetRetrievalMethod

```
CWebContentManager::EWebRetrievalMethod GetRetrievalMethod() const;
```

Returns the retrieval method for the extension. The retrieval method indicates how Webdriver handles the type at runtime.

GetSourceTable

```
CString GetSourceTable() const;
```

Returns the name of the source table in which the extension content is stored.

GetSubType

```
CString GetSubType() const;
```

Returns the MIME subtype for the extension.

GetSuperType

```
CString GetSuperType() const;
```

Returns the MIME supertype for the extension.

SetContentColumn

```
void SetContentColumn(const CString& contentColumn);
```

Sets the name of the column in which the extension content is stored to *contentColumn*.

SetIDColumn

```
void SetIDColumn(const CString& IDColumn);
```

Sets the name of the column in which the resource identifier for the extension is stored to *IDColumn*.

SetName

```
void SetName(const CString& name);
```

Sets the name of the extension to *name*.

SetPathColumn

```
void SetPathColumn(const CString &pathColumn);
```

Sets the name of the column in which the extension path is stored to *pathColumn*.

SetRetrievalMethod

```
void SetRetrievalMethod(CWebContentManager::EWebRetrievalMethod
retrievalMethod);
```

Sets the retrieval method for the extension to **CWebContent-Manager::EWebRetrievalMethod**. The retrieval method indicates how Webdriver handles the type at runtime. The retrieval methods are:

APB_EXPLODE	Run WebExplode on the page
APB_TEXT	Retrieve as text without exploding
APB_BLOB	Retrieve as a large object

SetSourceTable

```
void SetSourceTable(const CString& table);
```

Sets the name of the source table in which the extension content is stored to *table*.

SetSubType

```
void SetSubType(const CString& subtype);
```

Sets the MIME subtype for the extension to *subtype*.

SetSuperType

```
void SetSuperType(const CString& supertype);
```

Sets the MIME supertype for the extension to *supertype*.

Update

```
virtual BOOL Update();
```

Writes the extension object to the database. Returns FALSE if an error occurs; otherwise returns TRUE.

CWebOther

This class, derived from the **CMappedResource** class, defines the behavior of a resource object that is mapped in the extensions table but not stored in the Data Director for Web schema.

Public Interface

The following are the public methods on the **CWebOther** class.

GetContentFile

```
virtual CString GetContentFile();
```

Returns the path to a file on the client that contains the content for the non-Data Director for Web resource.

GetExtension

```
CString GetExtension() const;
```

Returns the extension of the non-Data Director for Web resource.

GetID

```
CString GetID() const;
```

Returns the identifier of the non-Data Director for Web resource.

GetPath

```
CString GetPath() const;
```

Returns the path of the non-Data Director for Web resource.

SetContentFile

```
virtual void SetContentFile(const CString& localFileName);
```

Sets the content of the non-Data Director for Web resource to *localFileName*.

CWebPage

This class, derived from the **CAPBResource** class, defines the behavior for Data Director for Web AppPages.

Public Interface

The following are the public methods on the **CWebPage** class.

CanBeLocked

```
BOOL CanBeLocked();
```

Checks whether or not the **CWebPage** object can be locked. A **CWebPage** object cannot be locked if it is explicitly locked or if any project to which it belongs is locked. Returns `TRUE` if the **CWebPage** object can be locked; otherwise returns `FALSE`.

Delete

```
virtual BOOL Delete();
```

Deletes the AppPage from the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

DeleteLeaveInProjects

```
virtual BOOL DeleteLeaveInProjects();
```

Deletes the AppPage from the database but does not remove its project associations from the **wbResProjects** table. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

This method is intended to be used when the AppPage is immediately replaced with another of the same identifier, extension, and path, where the existing project associations must be preserved.

GetAuthor

```
CString GetAuthor() const;
```

Returns the author of the page.

GetContentFile

```
virtual CString GetContentFile();
```

Returns the filename of the AppPage content.

GetDescription

```
CString GetDescription() const;
```

Returns the description of the AppPage.

GetExtension

```
CString GetExtension() const;
```

Returns the extension of the AppPage.

GetID

```
CString GetID() const;
```

Returns the identifier of the AppPage.

GetKeywords

```
CString GetKeywords() const;
```

Returns the keywords associated with the AppPage.

GetLastChanged

```
CString GetLastChanged() const;
```

Returns the date and time that the AppPage was last changed.

GetLastChangedBy

```
CString GetLastChangedBy() const;
```

Returns the name of the user who last changed the AppPage.

GetLastLocked

```
CString GetLastLocked() const;
```

Returns the date and time that the AppPage was last locked.

GetLastLockedBy

```
CString GetLastLockedBy() const;
```

Returns the name of the user who last locked the AppPage.

GetPath

```
CString GetPath() const;
```

Returns the path of the AppPage.

GetReadLevel

```
int GetReadLevel() const;
```

Returns the runtime security level for read access to the AppPage.

GetSize

```
long GetSize();
```

Returns the size, in bytes, of the **CWebPage** object.

Lock

```
BOOL Lock();
```

Locks the AppPage, preventing write access. Returns FALSE if an error occurs; otherwise returns TRUE.

Rename

```
CWebPage *Rename(CWebPage *newpage);
```

Creates a copy of a **CWebPage** object in the newly created object *newpage*. After invoking **Rename**, you must explicitly invoke **Update** to save *newpage* to the database, and delete both the old and new **CWebPage** objects. The behavior of any method called on the old **CWebPage** object after **Rename** has been called is undefined.

SetAuthor

```
void SetAuthor(const CString& author);
```

Sets the author of the AppPage to *author*.

SetContentFile

```
virtual void SetContentFile(const CString& localFileName);
```

Sets the content of the AppPage to the local file *localFileName*.

SetCurrentVersion

```
void SetCurrentVersion(int version);
```

Sets the version number of the AppPage to *version*.

SetDescription

```
void SetDescription(const CString& description);
```

Sets the description of the AppPage to *description*.

SetKeywords

```
void SetKeywords(const CString& keywords);
```

Sets the keywords associated with the AppPage to *keywords*.

SetReadLevel

```
void SetReadLevel(int level);
```

Sets the runtime security level for read access to the AppPage to *level*.

Unlock

```
BOOL Unlock();
```

Releases the lock on the AppPage. Returns FALSE if an error occurs; otherwise returns TRUE.

Update

```
virtual BOOL Update();
```

Writes the AppPage object to the database. Returns FALSE if an error occurs; otherwise returns TRUE.

CWebProject

This class, derived from the **CDatabaseObject** class, defines the behavior for Data Director for Web projects.

Public Interface

The following are the public methods on the **CWebProject** class.

CanBeLocked

```
BOOL CanBeLocked();
```

Checks whether or not the project can be locked by the current user. A project cannot be locked by the current user if:

- it is already locked by another user.
- a resource belonging to the project is already explicitly locked by another user.
- a resource belonging to the project belongs to another project which is already locked by another user.

Delete

```
virtual BOOL Delete();
```

Deletes the project from the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

GetDeployedDb

```
CString GetDeployedDb() const;
```

Returns the name of the database from which the project was last deployed.

GetDeployedProject

```
CString GetDeployedProject() const;
```

Returns the name of the new project from which the current project was last deployed.

GetDeployedServer

```
CString GetDeployedServer() const;
```

Returns the name of the database server from which the project was last deployed.

GetDescription

```
CString GetDescription() const;
```

Returns the description of the project.

GetLastDeployed

```
CString GetLastDeployed() const;
```

Returns the date and time that the project was last deployed to a different database.

GetLastDeployedBy

```
CString GetLastDeployedBy() const;
```

Returns the name of the user who last deployed the project to a different database.

GetLastLocked

```
CString GetLastLocked() const;
```

Returns the date and time that the project was last locked.

GetLastLockedBy

```
CString GetLastLockedBy() const;
```

Returns the name of the user who last locked the project.

GetName

```
CString GetName() const;
```

Returns the name of the project.

GetOwner

```
CString GetOwner() const;
```

Returns the name of the owner of the project.

IsLocked

```
BOOL IsLocked();
```

Checks whether or not the **CWebProject** object is locked. Returns `TRUE` if the **CWebProject** object is locked; otherwise returns `FALSE`.

Lock

```
BOOL Lock();
```

Locks the project, preventing write access to all resources associated with the project. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

Rename

```
CWebProject *Rename(const CString &name);
```

Creates a copy of a **CWebProject** object in the newly created object *name*. After invoking **Rename**, you must explicitly invoke **Update** to save *name* to the database, and delete both the old and new **CWebProject** objects. The behavior of any method called on the old **CWebProject** object after **Rename** has been called is undefined.

SetDeployedDb

```
void SetDeployedDb(const CString& database);
```

Sets the name of the database from which the project was deployed to *database*.

SetDeployedProject

```
void SetDeployedProject(const CString& project);
```

Sets the name of the project from which the project was deployed to *project*.

SetDeployedServer

```
void SetDeployedServer(const CString& server);
```

Sets the name of the database server from which the project was deployed to *server*.

SetDescription

```
void SetDescription(const CString& description);
```

Sets the description of the project to *description*.

SetLastDeployedAndDeployedBy

```
void SetLastDeployedAndDeployedBy();
```

Sets the **last_deployed** column of the project to the current date and time, and sets the **last_deployed_by** column of the project to the name of the current user.

SetOwner

```
void SetOwner();
```

Sets the owner of the project to the name of the current user.

Unlock

```
BOOL Unlock();
```

Releases the lock on the project, unlocking all resources associated with the project. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

Update

```
virtual BOOL Update();
```

Writes the project object to the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

CWebTag

This class, derived from the **CDatabaseObject** class, defines the behavior for dynamic tags stored in the database.

Public Interface

The following are the public methods on the **CWebTag** class.

CanBeLocked

```
BOOL CanBeLocked();
```

Checks whether or not the **CWebTag** object can be locked. A **CWebTag** object cannot be locked if it is explicitly locked or if any project to which it belongs is locked. Returns `TRUE` if the **CWebTag** object can be locked; otherwise returns `FALSE`.

Delete

```
virtual BOOL Delete();
```

Deletes the dynamic tag from the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

DeleteLeaveInProjects

```
virtual BOOL DeleteLeaveInProjects();
```

Deletes the dynamic tag from the database but does not remove its project associations from the **wbResProjects** table. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

This method is intended to be used when the dynamic tag is immediately replaced with another of the same identifier, extension, and path, where the existing project associations must be preserved.

GetClass

```
CString GetClass() const;
```

Returns the class of the dynamic tag.

GetContentFile

```
CString GetContentFile();
```

Returns the path to a file on the client that contains the code for the dynamic tag.

GetCurrentVersion

```
int GetCurrentVersion() const;
```

Returns the version number for the current dynamic tag.

GetDescription

```
CString GetDescription() const;
```

Returns the description of the dynamic tag.

GetID

```
CString GetID() const;
```

Returns the identifier of the dynamic tag.

GetLastChanged

```
CString GetLastChanged() const;
```

Returns the date and time that the dynamic tag was last changed.

GetLastChangedBy

```
CString GetLastChangedBy() const;
```

Returns the name of the user who last changed the dynamic tag.

GetLastLocked

```
CString GetLastLocked() const;
```

Returns the date and time that the dynamic tag was last locked.

GetLastLockedBy

```
CString GetLastLockedBy() const;
```

Returns the name of the user who last locked the dynamic tag.

GetParameters

```
CString GetParameters() const;
```

Returns the parameters of the dynamic tag.

IsLocked

```
BOOL IsLocked();
```

Checks whether or not the **CWebTag** object is locked. Returns TRUE if the **CWebTag** object is locked; otherwise returns FALSE.

Lock

```
BOOL Lock();
```

Locks the dynamic tag, preventing write access. Returns FALSE if an error occurs; otherwise returns TRUE.

Rename

```
CWebTag *Rename(CWebTag *newtag);
```

Creates a copy of a **CWebTag** object in the newly created object *newtag*. After invoking **Rename**, you must explicitly invoke **Update** to save *newtag* to the database, and delete both the old and new **CWebTag** objects. The behavior of any method called on the old **CWebTag** object after **Rename** has been called is undefined.

SetClass

```
void SetClass(const CString& classtext);
```

Sets the class of the dynamic tag to *classtext*.

SetContentFile

```
void SetContentFile(const CString& localFileName);
```

Sets the content of the dynamic tag to *localFileName*.

SetCurrentVersion

```
void SetCurrentVersion(int version);
```

Sets the version of the dynamic tag to *version*.

SetDescription

```
void SetDescription(const CString& description);
```

Sets the description of the dynamic tag to *description*.

SetParameters

```
void SetParameters(const CString& parameters);
```

Sets the parameters of the dynamic tag to *parameters*.

Unlock

```
BOOL Unlock();
```

Releases the lock on the dynamic tag. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

Update

```
virtual BOOL Update();
```

Writes the dynamic tag to the database. Returns `FALSE` if an error occurs; otherwise returns `TRUE`.

Chapter

4

Error Handling and Troubleshooting

In This Chapter

This chapter discusses error handling and troubleshooting when developing applications using the Data Director for Web API.

Error Handling

The API uses the **CWebError** class **LogMessage** method to log messages. If an error occurs, the API passes to **LogMessage** the parameter's object name, method name, and error message.

An instance of **CWebError** is passed into the API when the API is constructed. To configure error handling, create a new error-handler class that inherits from **CWebError** and pass the new class into the API.

The default behavior for the **LogMessage** method on **CWebError** is to display a message box on the screen, with the object, method, and error message displayed. You can override this behavior by defining your own **LogMessage** method in a new error-handler subclass.

Troubleshooting

All API classes that inherit from **CDatabaseObject** have a **Trace** method defined. You should not encounter problems with the API unless data has been seriously corrupted or the connection has been dropped.

To check that API objects contain valid data, invoke the **Trace** method. The **Trace** method produces a screen dump of all the internal variables in a class.

When troubleshooting an application, remember that the two most common problems are:

- connection errors caused by invalid connection information (such as requesting a connection to a nonexistent database or supplying an invalid user name / password).
- runtime errors caused by passing invalid parameters (such as passing in a null list pointer).

Appendix

A

Data Director for Web Schema Definitions

This appendix contains the schema definitions for the tables in the Data Director for Web schema for both Universal Web Connect and the Web DataBlade module.

wbPages

The **wbPages** table stores AppPages and any other text format resources, such as plain text files or HTML pages. Resources other than AppPages have extensions that map to types not expanded by **WebExplode**.

Column	UWC Type	WB Type	Description
ID	VARCHAR(30)	VARCHAR(30)	The name of the AppPage
path	VARCHAR(178)	VARCHAR(178)	The file system path to the AppPage
extension	VARCHAR(12)	VARCHAR(12)	The file system extension of the AppPage
description	VARCHAR(254)	VARCHAR(254)	The description of the AppPage
author	VARCHAR(30)	VARCHAR(30)	The author of the AppPage
keywords	VARCHAR(254)	VARCHAR(254)	The keywords associated with the AppPage
current_version	INTEGER	INTEGER	The version number of the AppPage
last_changed	DATETIME	DATETIME	When the AppPage was last modified
last_changed_by	VARCHAR(30)	VARCHAR(30)	The user who last changed the AppPage
read_level	INTEGER	INTEGER	The runtime security level required to read the AppPage

(Sheet 1 of 2)

Column	UWC Type	WB Type	Description
last_locked	DATETIME	DATETIME	When the AppPage was last locked
last_locked_by	VARCHAR(30)	VARCHAR(30)	The user who locked the AppPage
object	TEXT IN %s	HTML	The content of the AppPage

(Sheet 2 of 2)

wbBinaries

The **wbBinaries** table stores all nontext resources.

Column	UWC Type	WB Type	Description
ID	VARCHAR(30)	VARCHAR(30)	The name of the resource
path	VARCHAR(178)	VARCHAR(178)	The file system path to the resource
extension	VARCHAR(12)	VARCHAR(12)	The file system extension of the resource
description	VARCHAR(254)	VARCHAR(254)	The description of the resource
height	INTEGER	INTEGER	The height of the resource, used to build an IMG or EMBED tag
width	INTEGER	INTEGER	The width of the resource, used to build an IMG or EMBED tag
current_version	INTEGER	INTEGER	The version number of the resource
last_changed	DATETIME	DATETIME	When the resource was last modified
last_changed_by	VARCHAR(30)	VARCHAR(30)	The user who last changed the resource
read_level	INTEGER	INTEGER	The runtime security level required to read the resource

(Sheet 1 of 2)

Column	UWC Type	WB Type	Description
last_locked	DATETIME	DATETIME	When the resource was last locked
last_locked_by	VARCHAR(30)	VARCHAR(30)	The user who locked the resource
object	BYTE IN %s	BLOB	The content of the resource

(Sheet 2 of 2)

wbTags

The **wbTags** table stores Web Integration Option dynamic tags.

Column	UWC Type	WB Type	Description
ID	VARCHAR(30)	VARCHAR(30)	The name of the tag
description	VARCHAR(254)	VARCHAR(254)	The description of the tag
parameters	VARCHAR(254)	VARCHAR(254)	The parameters passed to the tag
class	VARCHAR(64)	VARCHAR(64)	The class of the tag
current_version	INTEGER	INTEGER	The version number of the tag
last_changed	DATETIME	DATETIME	When the tag was last modified
last_changed_by	VARCHAR(30)	VARCHAR(30)	The user who last changed the tag
last_locked	DATETIME	DATETIME	When the tag was last locked
last_locked_by	VARCHAR(30)	VARCHAR(30)	The user who locked the tag
object	TEXT IN %s	HTML	The content of the tag
wizData	BYTE IN %s	BLOB	The properties of the tag(wizard-generated tags only)

wbExtensions

The **wbExtensions** table stores type-mapping information.

Column	UWC Type	WB Type	Description
extension	VARCHAR(12)	VARCHAR(12)	The file extension
name	VARCHAR(30)	VARCHAR(30)	The name of the extension
source_table	VARCHAR(18)	VARCHAR(18)	The name of the table in which the resource is stored
super_type	VARCHAR(18)	VARCHAR(18)	The MIME supertype of the extension
sub_type	VARCHAR(18)	VARCHAR(18)	The MIME subtype of the extension
ID_column	VARCHAR(18)	VARCHAR(18)	The name of the column containing the resource ID
content_column	VARCHAR(18)	VARCHAR(18)	The name of the column containing the resource content
path_column	VARCHAR(18)	VARCHAR(18)	The name of the column containing the resource path information
retrieval_method	INTEGER	INTEGER	The retrieval method used by Webdriver when retrieving the type

wbProjects

The **wbProjects** table stores the project information for all projects in the database.

Column	UWC Type	WB Type	Description
name	VARCHAR(30)	VARCHAR(30)	The name of the project
description	VARCHAR(254)	VARCHAR(254)	The description of the project
owner	VARCHAR(30)	VARCHAR(30)	The owner of the project
last_locked	DATETIME	DATETIME	When the project was last locked
last_locked_by	VARCHAR(30)	VARCHAR(30)	The user who locked the project
last_deployed	DATETIME	DATETIME	When the project was last deployed
last_deployed_by	VARCHAR(30)	VARCHAR(30)	The user who last deployed the project
deployed_db	VARCHAR(18)	VARCHAR(18)	The database from which the project was last deployed
deployed_server	VARCHAR(18)	VARCHAR(18)	The database server from which the project was last deployed
deployed_project	VARCHAR(30)	VARCHAR(30)	The name of the project from which this project was last deployed
preview	VARCHAR(30)	VARCHAR(30)	The preview configuration name for the project

wbResProjects

A project can have many resources associated with it, and a resource can belong to many projects. The **wbResProjects** table manages the many-to-many relationship between projects and resources.

Column	UWC Type	WB Type	Description
project_name	VARCHAR(30)	VARCHAR(30)	The name of the project
ID	VARCHAR(30)	VARCHAR(30)	The ID of the resource
path	VARCHAR(178)	VARCHAR(178)	The path of the resource
extension	VARCHAR(12)	VARCHAR(12)	The extension of the resource

wbPreviews

The **wbPreviews** table contains the preview configurations for each project.

Column	UWC Type	WB Type	Description
ID	VARCHAR(30)	VARCHAR(30)	The ID of the preview configuration
object	TEXT IN %s	HTML	The preview configuration settings

Appendix

B

Data Director for Web API Examples

This appendix contains the complete example code found in Chapter 2, “Using the Data Director for Web API.”

Connect

```
// Create an instance of an error-handler class.

error = new CWebError();

// Create an instance of a factory-mapper class.

factory = new CWebFactoryMapper();

// Create a new instance of the API with the factory-mapper
// and error-handler classes.  Also, specify the cache
// directory as D:\APBDump.  Note the need to use double
// slashes to pass a single slash.

api = new CWebAPI(factory, error,"D:\\APBDump");


// Connect to a database using the parameters:
//
// Server:  server
// Database:    db
// User:    foo
// Password:    guest

if ( ! api->Connect("server",
    "db",
    "foo","guest!"))
{
    TRACE0("Failed to connect to database db as foo \n");
    return ;
}
```

Connection Information

```
// Retrieve the connection parameters.

CString server = api->GetServer();
CString database = api->GetDatabase();
CString user = api->GetUser();

TRACE0("user " + user + " is connected to server:" +
    server + " and database:" + database + "\n");
```

Disconnect

```
// Delete the API instance.

delete api;

// Delete the error-handler instance.

delete error;

// Delete the factory-mapper instance.

delete factory;
```

Managing Projects

```
// Fetch the project named 'first'.

CWebProject* project;

if ((project = api->GetWebProject("first")) != NULL)
{
    project->Trace();
    delete project;
}


// List all projects in the database.

// Create a new, empty project list.
CPtrList* projects = new CPtrList();

// Fetch the projects from the database.
if (! api->GetWebProjects(projects))
{
    // Perform some error handling.
}
// Loop through the projects and do some processing.

POSITION pos = projects->GetHeadPosition();
while( pos != NULL )
{
    CWebProject* project = (CWebProject*) projects->GetNext( pos );

    // Do some processing.
}

// Finished processing projects, so destroy the project
// objects and the projects list.

CWebAPI::DestroyContentOfList(projects);
delete projects;


// Create a new project.

CWebProject* newProject = api->CreateWebProject("NEWPROJECT");

if (newProject == NULL)
{
    // Perform some error processing.
}


// Modify a project.
```

```
newProject->SetDescription("modified description");
newProject->SetOwner();
newProject->SetLastDeployedAndDeployedBy();
newProject->SetDeployedDb("modified db");
newProject->SetDeployedServer("modified server");
newProject->SetDeployedProject("modified project");



// Commit the changes to the database.

if (!newProject->Update())
{
    // Perform some error processing.
}


// Lock a project.

if (!newProject->Lock())
{
    // Perform some error processing.
}


// Unlock the project.

if (!newProject->Unlock())
{
    // Perform some error processing.
}



// Delete a project.

if (!newProject->Delete())
{
    // Perform some error processing.
}

// Now that processing is completed, destroy the
// project object.

delete newProject;
```

Managing Extensions

```
// Fetch a specific extension (for example, the html extension).

CWebExtension* extension;

if ((extension = api->GetWebExtension("html")) != NULL)
{
    extension->Trace();
    delete extension;
}


// List all extensions in the database.

// create a new empty list

CPtrList* extensions = new CPtrList();

// Fetch the extensions from the database.

if (!api->GetWebExtensions(extensions))
{
    // Perform some error processing.
}

// Loop through the extensions and do some processing.

POSITION pos = extensions->GetHeadPosition();
while( pos != NULL )
{
    CWebExtension* extension = (CWebExtension*) extensions->GetNext( pos );
    // Perform some processing.
}

// Finished with extensions, so destroy the objects.

CWebAPI::DestroyContentOfList(extensions);


// List all the extensions for a particular supertype (in this case 'image').

if (api->GetWebExtensions("image",extensions))
{
    // Perform some error processing.
}

// Loop through extensions for 'image' supertype and do some processing.

pos = extensions->GetHeadPosition();
while( pos != NULL )
```

```
{
    CWebExtension* extension = (CWebExtension*) extensions->GetNext( pos );
    // Perform some processing.
}

// finished with extensions so thrown them away...

CWebAPI::DestroyContentOfList(extensions);


// Create a new extension.

CWebExtension* newExtension = api->CreateWebExtension("au");

if (newExtension == NULL)
{
    // Perform some error handling.
}


// Modify an extension.



newExtension->SetName("mod' ''ified name");
newExtension->SetSourceTable("modified_table");
newExtension->SetSuperType("text");
newExtension->SetSubType("doc");
newExtension->SetIDColumn("ID");
newExtension->SetContentColumn("content");
newExtension->SetRetrievalMethod(CWebContentManager::APB_TEXT);

// Commit changes to the database.

if (!newExtension->Update())
{
    // Perform some error handling.
}


// Delete an extension


if (!newExtension->Delete())
{
    // Perform some error handling.
}

// Finished with extension list and extension object, so delete them.

delete extensions;
delete newExtension;
```

Managing Dynamic Tags

```
// Fetch a particular tag, in this tag the tag named 'SELECTLIST'
CWebTag* tag;

if ((tag = api->GetWebTag("SELECTLIST")) != NULL)
{
    tag->Trace();
    delete tag;
}


// List all tags in the database.

// Create a new empty tag list.

CPtrList* tags = new CPtrList();

// Fetch tags from the database
if (!api->GetWebTags(tags))
{
    // Perform some error processing.
}

// Loop through tag list and do some tag processing.


POSITION pos = tags->GetHeadPosition();
while( pos != NULL )
{
    CWebTag* tag = (CWebTag*) tags->GetNext( pos );
    // Do some processing.
}


// Finshed with tags so destroy them.

CWebAPI::DestroyContentOfList(tags);



// List tags for a particular project.

// Fetch project 'first'.

CWebProject* project;

if ((project = api->GetWebProject("first")) == NULL)
{
    // Perform some error processing.
}
```

```
// Fetch tags for project 'first'.

if(!api->GetWebTags(*project,tags))
{
    // Perform some error processing.
}

// Do some project tag processing.

// Finished with tags, so destroy the tag list.

CWebAPI::DestroyContentOfList(tags);


// Create a new tag.

CWebTag* newTag = api->CreateWebTag("NEWTag");

if (newTag == NULL)
{
    // Perform some error processing.
}



// Modify tag.

newTag->SetDescription("modified '' description");
newTag->SetParameters("ONE,TWO,THREE");
newTag->SetClass("query");
newTag->SetContentFile("D:\\tag.html");

// Commit tag to database.

if (!newTag->Update())
{
    // Perform some error processing.
}



// Add a tag to a project.

if (!api->AddWebTagToProject(*project,*newTag))
{
    // Perform some error processing.
}


// Remove tag from project.

if (!api->RemoveWebTagFromProject(*project,*newTag))
{
```

```
        // Perform some error processing.
    }



    // Delete tag.

    if (!newTag->Delete())
    {
        // Perform some error processing.
    }

    // Finished with tag list, project and tag.

    delete tags;
    delete project;
    delete newTag;
```

Managing AppPages

```
// List all AppPages in the database.

// Fetch the 'html' extension object.

CWebExtension* extension = api->GetWebExtension("html");

if (extension == NULL)
{
    // Perform some error processing.
}

// Fetch all the AppPages in the database (all resources with the 'html'
extension).

// Create a new empty AppPages list.

CPtrList* pages = new CPtrList();

// Fetch AppPages from the database.

if (!api->GetAPBResources(*extension,pages))
{
    // Perform some error processing.
}


// Loop through AppPage list and do some AppPage processing.

POSITION pos = pages->GetHeadPosition();
while( pos != NULL )
{
    CAPBResource* page = (CAPBResource*) pages->GetNext( pos );
    // Do some processing.
}


// Finished with AppPage list, so destroy the AppPage objects.


CWebAPI::DestroyContentOfList(pages);



// List all AppPages in a project.

// Fetch project 'first'.

CWebProject* project;
```

```
if ((project = api->GetWebProject("first")) == NULL)
{
    // Perform some error processing.
}

// Fetch AppPages for project 'first'.

if (!api->GetAPBResources(*project,*extension,pages))
{
    // Perform some error processing.
}

// Do some project AppPage processing.

// Finished with AppPages, so destroy the AppPage objects.

CWebAPI::DestroyContentOfList(pages);


// Create a new AppPage.

CWebPage* newPage = (CWebPage*)api->CreateWebResource("NEWPage","/","html");

if (newPage == NULL)
{
    // Perform some error processing.
}



// Lock an AppPage.


if (!newPage->Lock())
{
    // Perform some error processing.
}




// Modify an AppPage.

newPage->SetDescription("modified '' description");
newPage->SetAuthor("Jane Programmer");
newPage->SetKeywords("api");
newPage->SetCurrentVersion(1);
newPage->SetReadLevel(99);
newPage->SetContentFile("D:\\api.html");

// Commit changes to the database.
if (!newPage->Update())
{
    // Perform some error processing.
```

```
}



// UnLock an AppPage.

if (!newPage->Unlock())
{
    // Perform some error processing.
}


// Add an AppPage to a project.


if (!api->AddAPBResourceToProject(*project,*newPage))
{
    // Perform some error processing.
}


// Remove AppPage from a project.

if (!api->RemoveAPBResourceFromProject(*project,*newPage))
{
    // Perform some error processing.
}




// List AppPages not in projects.

if (!api->GetAPBResourcesNotInAnyProject(*extension,pages))
{
    // Perform some error processing.
}

// Do some AppPage processing.

// Finished with AppPages, so destroy the AppPage objects.

CWebAPI::DestroyContentOfList(pages);



// Delete page.


if (!newPage->Delete())
{
    // Perform some error processing.
}
```

```
// Finished with list, project, extension and page, so destroy the objects.

delete pages;
delete project;
delete extension;
delete newPage;
```

Executing Web Integration Option Functions

```
// Split a resource into its component parts.

CString ID;
CString path;
CString ext;


CString resource = "/runtime/first.html";
api->SplitResourceName(resource,&path,&ID,&ext);
TRACE0("Resource:" + resource + " path:" + path + " ID:" + ID + " ext:" + ext +
"\n");


// ExecuteWebRelease
CString release = api->ExecuteWebRelease();
TRACE0("Web release version:" + release + "\n" );



// ExecuteWebURLEncode
CString src = "this is a (test) of {URL} *encoding*!!";

TRACE0("Encoding string" + src + "\n" );
CString encodedsrc = api->ExecuteWebURLEncode(src);
TRACE0("Encoded string:" + encodedsrc + "\n" );


// EncodeWebConfFile
TRACE0("Encoding conf file d:\\web.conf\n");
CString env = api->EncodeWebConfFile("D:\\Web.conf");

CWebPage* page;

page = (CWebPage *)api->GetWebResource("foo", "/", "html");

// ExecuteWebExplode


CString explodedPage = api->ExecuteWebExplode(page->GetContentFile(),env);


// ExecuteWebUnHTML

CString unhtmledPage = api->ExecuteWebUnHTML(page->GetContentFile());
```

Index

D

E

G

I

L

R

S

U