MX

macromedia®

DREAMWEAVER® MX 2004

Dreamweaver API Reference

Acknowledgments

Senior Management: Sheila McGinn

Project Management: Robert Berry

Writing: Robert Berry and David Jacowitz

Editing Management: Lisa Stanziano

Editing: Mary Kraemer

Production Management: Patrice O'Neill

Media Design and Production: Adam Barnett, Aaron Begley, Chris Basmajian, John Francis, Jeff Harmon

Special thanks to Jay London, Jeff Schang, Lori Hylan-Cho, Hisami Scott, Sam Mathews, Jake Cockrell, Russ Helfand, Randy Edmunds, George Comninos, Rosana Francescato, Charles Nadeau, and the entire Dreamweaver engineering and QA teams.

First Edition: November 2003

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

CHAPTER 1
Introduction

The *Dreamweaver API Reference* describes two application programming interfaces (APIs) that let you perform various supporting tasks when developing Macromedia Dreamweaver MX 2004 extensions and adding program code to your Dreamweaver web pages. These two APIs are the utility API and the JavaScript API. The utility API contains subsets of related functions that let you perform specific types of tasks. The utility API includes the following API subsets:

- The File I/O API, which lets you read and write files on the local file system
- The HTTP API, which lets you send and receive information from a web server
- The Design Notes API, which lets you store and retrieve notes about Dreamweaver documents
- The Fireworks Integration API, which lets you communicate with Macromedia Fireworks
- Flash Integration, which contains information about adding Flash elements to the Dreamweaver user interface (UI) and details on the Flash Objects API (which lets you build objects that create Macromedia Flash content)
- The Database API, which lets you access information stored in databases and manage database connections
- The Database Connectivity API, which lets you create a new connection type and corresponding dialog boxes for new or existing server models
- The JavaBeans API, which retrieves class names, methods, properties, and events for JavaBeans that you have defined
- The Source Control Integration API, which lets you write shared libraries to extend the Dreamweaver Check In/Check Out feature

The extensive JavaScript API lets you perform a diverse set of smaller tasks, many of which are tasks that a user would perform when creating or editing Dreamweaver documents. These API functions are grouped by the parts of the Dreamweaver UI that they affect. For example, the JavaScript API includes Workspace functions, Document functions, Design functions, and so on. These functions let you perform tasks such as opening a new document, getting or setting a font size, finding the occurrence of a search string in HTML code, making a toolbar visible, and much more.

Background

This book assumes that you are familiar with Dreamweaver, HTML, XML, JavaScript programming and, if applicable, C programming. If you are writing extensions for building web applications, you should also be familiar with server-side scripting on at least one platform, such as Active Server Pages (ASP), ASP.net, PHP: Hypertext Preprocessor (PHP), ColdFusion, or Java Server Pages (JSP).

Extending Dreamweaver

To learn about the Dreamweaver framework and the API that enables you to build Dreamweaver extensions, see *Extending Dreamweaver*. *Extending Dreamweaver* describes the API functions that Dreamweaver calls to implement the objects, menus, floating panels, server behaviors, and so on, that comprise the various features of Dreamweaver. You can use those APIs to add objects, menus, floating panels, or other features to the product. *Extending Dreamweaver* also explains how to customize Dreamweaver by editing and adding tags to various HTML and XML files to add menu items or document types, and so on.

Additional resources for extension writers

To communicate with other developers who are involved in writing extensions, you might want to join the Dreamweaver extensibility newsgroup. You can access the website for this newsgroup at www.macromedia.com/go/extending_newsgrp/.

New functions in Dreamweaver MX 2004

The following new functions have been added to the Dreamweaver MX 2004 JavaScript API. The headings designate the chapters and sections that contain the new functions:

Workspace

The following Insert Bar, Results window, and Toolbar functions have been added to the Workspace chapter.

Insert object

- “`dom.insertFlashElement()`” on page 161
- “`dreamweaver.objectPalette.getMenuDefault()`” on page 162
- “`dreamweaver.objectPalette.setMenuDefault()`” on page 162
- “dreamweaver.reloadObjects()” on page 163

Results window

“`dreamweaver.showResults()`” on page 172

Toolbar

- “`dom.getToolbarItemValue()`” on page 203
- “`dom.setToolbarItemAttribute()`” on page 205

Document

The following new Global document functions have been added to the Document chapter.

Global document

- "`dom.hideInfoMessagePopup()`" on page 268
- "`dreamweaver.showTargetBrowsersDialog()`" on page 276
- "`dom.showInfoMessagePopup()`" on page 269

Design

The following new CSS, HTML, and Table Editing functions have been added to the Design chapter.

CSS

- "`dreamweaver.cssRuleTracker.newRule()`" on page 347
- "`dreamweaver.cssRuleTracker.editSelectedRule()`" on page 347
- "`dreamweaver.cssStylePalette.getMediaType()`" on page 350
- "`dreamweaver.cssStylePalette.setMediaType()`" on page 352

Table editing

- "`dom.getShowTableWidths()`" on page 371
- "`dom.removeColumnWidth()`" on page 375
- "`dom.setShowTableWidths()`" on page 375

Code

The following new Code view functions have been added to the Code chapter.

Code view

- "`dom.getAutoValidationCount()`" on page 409
- "`dom.source.getValidationErrorsForOffset()`" on page 415

Removed functions

The following functions have been removed from the Dreamweaver MX 2004 API because the associated features have been removed from the product.

Document

In the Document chapter, the following conversion function has been removed:

Conversion function

`dom.convertTo30()`

Page content

In the Page Content chapter, the following Timeline functions have been removed:

Timeline functions

- `dreamweaver.timelineInspector.addBehavior()`
- `dreamweaver.timelineInspector.addFrame()`
- `dreamweaver.timelineInspector.addKeyframe()`
- `dreamweaver.timelineInspector.addObject()`
- `dreamweaver.timelineInspector.addTimeline()`
- `dreamweaver.timelineInspector.changeObject()`
- `dreamweaver.timelineInspector.getAutoplay()`
- `dreamweaver.timelineInspector.getCurrentFrame()`
- `dreamweaver.timelineInspector.getLoop()`
- `dreamweaver.timelineInspector.recordPathOfLayer()`
- `dreamweaver.timelineInspector.removeBehavior()`
- `dreamweaver.timelineInspector.removeFrame()`
- `dreamweaver.timelineInspector.removeKeyframe()`
- `dreamweaver.timelineInspector.removeObject()`
- `dreamweaver.timelineInspector.removeTimeline()`
- `dreamweaver.timelineInspector.renameTimeline()`
- `dreamweaver.timelineInspector.setAutoplay()`
- `dreamweaver.timelineInspector.setCurrentFrame()`
- `dreamweaver.timelineInspector.setLoop()`

Design

In the Design chapter, the following HTML style functions have been removed:

HTML style functions

- `dom.applyHTMLStyle()`
- `dreamweaver.htmlStylePalette.deleteSelectedStyle()`
- `dreamweaver.htmlStylePalette.duplicateSelectedStyle()`
- `dreamweaver.htmlStylePalette.editSelectedStyle()`
- `dreamweaver.htmlStylePalette.getSelectedStyle()`
- `dreamweaver.htmlStylePalette.getStyles()`
- `dreamweaver.htmlStylePalette.newStyle()`
- `dreamweaver.htmlStylePalette.setSelectedStyle()`

Code

In the Code chapter, the following JavaScript Debugger and Tag Inspector functions have been removed:

JS Debugger functions

- `dreamweaver.debugDocument()`
- `dreamweaver.startDebugger()`
- `dreamweaver.getIsAnyBreakpoints()`
- `dreamweaver.removeAllBreakpoints()`

Tag Inspector functions

- `dreamweaver.tagInspector.deleteTags()`
- `dreamweaver.tagInspector.deleteTagsEnabled()`
- `dreamweaver.tagInspector.editTagName()`
- `dreamweaver.tagInspector.editTagNameEnabled()`
- `dreamweaver.tagInspector.tagAfter()`
- `dreamweaver.tagInspector.tagAfterEnabled()`
- `dreamweaver.tagInspector.tagBefore()`
- `dreamweaver.tagInspector.tagBeforeEnabled()`
- `dreamweaver.tagInspector.tagInside()`
- `dreamweaver.tagInspector.tagInsideEnabled()`

Enablers

In the Enablers chapter, the following Enabler functions have been removed:

- `dreamweaver.timelineInspector.canAddFrame()`
- `dreamweaver.timelineInspector.canAddKeyFrame()`
- `dreamweaver.timelineInspector.canChangeObject()`
- `dreamweaver.timelineInspector.canRemoveBehavior()`
- `dreamweaver.timelineInspector.canRemoveFrame()`
- `dreamweaver.timelineInspector.canRemoveKeyFrame()`
- `dreamweaver.timelineInspector.canRemoveObject()`

Other

The `"answers"` and `"sitespring"` arguments have been removed from the `dw.getFloaterVisibility()` and `dw.setFloaterVisibility()` functions.

Documentation changes

Extending Dreamweaver MX has been divided into two books: *Extending Dreamweaver* and the *Dreamweaver API Reference*. *Extending Dreamweaver* describes how to build various types of Dreamweaver extensions, including the functions that you must write to create each type. It also describes how to customize Dreamweaver by modifying some of its configurable HTML and XML files. The *Dreamweaver API Reference* describes the two APIs that let you perform various supporting tasks in your Dreamweaver extensions.

The *Extending Dreamweaver* book is designed to serve the user who wants to learn how to build a Dreamweaver extension. The *Dreamweaver API Reference* is designed to serve the experienced Dreamweaver programmer who wants to quickly locate the right function to accomplish a particular task. Dividing the material into two books also clarifies the distinction between the extension API functions that an extension author must code, and which Dreamweaver calls, and the JavaScript and Utility API functions that a programmer can call to accomplish various tasks from within an extension.

The *Dreamweaver API Reference* includes the following improvements for extension authors:

- Reorganization of functions into chapters.

  The JavaScript API functions are now grouped first by chapter, according to the related Dreamweaver context (Workspace, Document, Design, and so on), and then by topic according to the specific functionality (Keyboard functions, Menu functions, Window functions, and so on).
- Deprecated functions now "inline."

  Deprecated functions were previously in their own section. Now, deprecated functions remain in their original topic to provide context, and are denoted with the suffix "(deprecated)" in the function name.
- Enabler functions chapter.

  The Enabler functions have been organized into their own chapter with cross references to their relevant API functions.

Errata

A current list of known issues can be found in the Extensibility section of the Dreamweaver Support Center (www.macromedia.com/go/extending_errata).

Conventions used in this guide

The following typographical conventions are used in this guide:

- `Code` font indicates code fragments and API literals, including class names, method names, function names, type names, scripts, SQL statements, and both HTML and XML tag and attribute names.
- *`Italic code`* font indicates replaceable items in code.
- The continuation symbol (¬) indicates that a long line of code has been broken across two or more lines. Due to margin limits in this book's format, what is otherwise a continuous line of code must be split. When copying the lines of code, eliminate the continuation symbol and type the lines as one line.
- Curly braces ({ }) that surround a function argument indicate that the argument is optional.
- Function names that have the prefix `dreamweaver.`*`funcname`* can be abbreviated to `dw.`*`funcname`* when you are writing code. This manual uses the full `dreamweaver.` prefix when defining the function and in the index. Many examples use the `dw.` prefix, however.

The following naming conventions are used in this guide:

- You—the developer who is responsible for writing extensions
- The user—the person using Dreamweaver

PART I
Utility APIs

Learn about the Macromedia Dreamweaver MX 2004 utility functions that you can use to access local and web-based files, work with Macromedia Fireworks MX 2004, and Macromedia Flash MX 2004 objects, manage database connections, create new database connection types, access JavaBeans fscomponents, and integrate Dreamweaver with various source control systems.

CHAPTER 2
The File I/O API

Macromedia Dreamweaver MX 2004 includes a C shared library called DWfile, which gives authors of objects, commands, behaviors, data translators, floating panels, and Property inspectors the ability to read and write files on the local file system. This chapter describes the File I/O API and how to use it.

For general information on how C libraries interact with the JavaScript interpreter in Dreamweaver, see "C-Level Extensibility" in *Extending Dreamweaver.*

Accessing configuration folders

On Microsoft Windows 2000 and Windows XP, and Mac OS X platforms, users have their own copies of configuration files. Whenever Dreamweaver writes to a configuration file, Dreamweaver writes it to the user's Configuration folder. Similarly, when Dreamweaver reads a configuration file, Dreamweaver searches for it first in the user's Configuration folder and then in the Dreamweaver Configuration folder. DWfile functions use the same mechanism. In other words, if your extension reads or writes a file in the Dreamweaver Configuration folder, your extension also accesses the user's Configuration folder. For more information about configuration folders on multiuser platforms, see *Extending Dreamweaver.*

The File I/O API

All functions in the File I/O API are methods of the `DWfile` object.

DWfile.copy()

Availability

Dreamweaver 3.

Description

This function copies the specified file to a new location.

Arguments

originalURL, *copyURL*

- The *originalURL* argument, which is expressed as a file:// URL, is the file you want to copy.
- The *copyURL* argument, which is expressed as a file:// URL, is the location where you want to save the copied file.

Returns

A Boolean value: `true` if the copy succeeds; `false` otherwise.

Example

The following code copies a file called myconfig.cfg to myconfig_backup.cfg:

```
var fileURL = "file:///c|/Config/myconfig.cfg";
var newURL ="file:///c|/Config/myconfig_backup.cfg";
DWfile.copy(fileURL, newURL);
```

DWfile.createFolder()

Availability

Dreamweaver 2.

Description

This function creates a folder at the specified location.

Arguments

folderURL

- The *folderURL* argument, which is expressed as a file:// URL, is the location of the folder you want to create.

Returns

A Boolean value: `true` if the folder is created successfully; `false` otherwise.

Example

The following code tries to create a folder called tempFolder at the top level of the C drive and displays an alert box that indicates whether the operation was successful:

```
var folderURL = "file:///c|/tempFolder";
if (DWfile.createFolder(folderURL)){
  alert("Created " + folderURL);
}else{
  alert("Unable to create " + folderURL);
}
```

DWfile.exists()

Availability

Dreamweaver 2.

Description

This function tests for the existence of the specified file.

Arguments

fileURL

- The *fileURL* argument, which is expressed as a file:// URL, is the requested file.

Returns

A Boolean value: `true` if the file exists; `false` otherwise.

Example

The following code checks for the mydata.txt file and displays an alert message that tells the user whether the file exists:

```
var fileURL = "file:///c|/temp/mydata.txt";
if (DWfile.exists(fileURL)){
  alert(fileURL + " exists!");
}else{
  alert(fileURL + " does not exist.");
}
```

DWfile.getAttributes()

Availability

Dreamweaver 2.

Description

This function gets the attributes of the specified file or folder.

Arguments

fileURL

- The *fileURL* argument, which is expressed as a file:// URL, is the file or folder for which you want to get attributes.

Returns

A string that represents the attributes of the specified file or folder. If the file or folder does not exist, this function returns a `null` value. The following characters in the string represent the attributes:

- `R` is read only.
- `D` is folder.
- `H` is hidden.
- `S` is system file or folder.

Example

The following code gets the attributes of the mydata.txt file and displays an alert box if the file is read only:

```
var fileURL = "file:///c|/temp/mydata.txt";
var str = DWfile.getAttributes(fileURL);
if (str && (str.indexOf("R") != -1)){
  alert(fileURL + " is read only!");
}
```

DWfile.getModificationDate()

Availability

Dreamweaver 2.

Description

This function gets the time when the file was last modified.

Arguments

`fileURL`

- The *`fileURL`* argument, which is expressed as a file:// URL, is the file for which you are checking the last modified time.

Returns

A string that contains a hexadecimal number that represents the number of time units that have elapsed since some base time. The exact meaning of time units and base time is platform-dependent; in Windows, for example, a time unit is 100ns, and the base time is January 1st, 1600.

Example

It's useful to call the function twice and compare the return values because the value that this function returns is platform-dependent and is not a recognizable date and time. The following code example gets the modification dates of file1.txt and file2.txt and displays an alert message that indicates which file is newer:

```
var file1 = "file:///c|/temp/file1.txt";
var file2 = "file:///c|/temp/file2.txt";
var time1 = DWfile.getModificationDate(file1);
var time2 = DWfile.getModificationDate(file2);
if (time1 == time2){
  alert("file1 and file2 were saved at the same time");
}else if (time1 < time2){
  alert("file1 older that file2");
}else{
  alert("file1 is newer than file2");
}
```

DWfile.getCreationDate()

Availability

Dreamweaver 4.

Description

This function gets the time when the file was created.

Arguments

fileURL

- The *fileURL* argument, which is expressed as a file:// URL, is the file for which you are checking the creation time.

Returns

A string that contains a hexadecimal number that represents the number of time units that have elapsed since some base time. The exact meaning of time units and base time is platform-dependent; in Windows, for example, a time unit is 100ns, and the base time is January 1st, 1600.

Example

You can call this function and the `DWfile.getModificationDate()` function on a file to compare the modification date to the creation date:

```
var file1 = "file:///c|/temp/file1.txt";
var time1 = DWfile.getCreationDate(file1);
var time2 = DWfile.getModificationDate(file1);
if (time1 == time2){
  alert("file1 has not been modified since it was created");
}else if (time1 < time2){
  alert("file1 was last modified on " + time2);
}
```

DWfile.getCreationDateObj()

Availability

Dreamweaver MX.

Description

This function gets the JavaScript object that represents the time when the file was created.

Arguments

fileURL

- The *fileURL* argument, which is expressed as a file:// URL, is the file for which you are checking the creation time.

Returns

A JavaScript `Date` object that represents the date and time when the specified file was created.

DWfile.getModificationDateObj()

Availability

Dreamweaver MX.

Description

This function gets the JavaScript `Date` object that represents the time when the file was last modified.

Arguments

`fileURL`

- The *`fileURL`* argument, which is expressed as a file:// URL, is the file for which you are checking the time of the most recent modification.

Returns

A JavaScript `Date` object that represents the date and time when the specified file was last modified.

DWfile.getSize()

Availability

Dreamweaver MX.

Description

This function gets the size of a specified file.

Arguments

`fileURL`

- The *`fileURL`* argument, which is expressed as a file:// URL, is the file for which you are checking the size.

Returns

An integer that represents the actual size, in bytes, of the specified file.

DWfile.listFolder()

Availability

Dreamweaver 2.

Description

This function gets a list of the contents of the specified folder.

Arguments

folderURL, {constraint}

- The *folderURL* argument is the folder for which you want a contents list, which is expressed as a file:// URL, plus an optional wildcard file mask. Valid wildcards are asterisks (*), which match one or more characters, and question marks (?), which match a single character.
- The *constraint* argument, if it is supplied, must be either `"files"` (return only files) or `"directories"` (return only folders). If it is omitted, the function returns files and folders.

Returns

An array of strings that represents the contents of the folder.

Example

The following code gets a list of all the text (TXT) files in the C:/temp folder and displays the list in an alert message:

```
var folderURL = "file:///c|/temp";
var fileMask = "*.txt";
var list = DWfile.listFolder(folderURL + "/" + fileMask, "files");
if (list){
  alert(folderURL + " contains: " + list.join("\n"));
}
```

DWfile.read()

Availability

Dreamweaver 2.

Description

This function reads the contents of the specified file into a string.

Arguments

fileURL

- The *fileURL* argument, which is expressed as a file:// URL, is the file you want to read.

Returns

A string that contains the contents of the file or a `null` value if the read fails.

Example

The following code reads the mydata.txt file and, if it is successful, displays an alert message with the contents of the file:

```
var fileURL = "file:///c|/temp/mydata.txt";
var str = DWfile.read(fileURL);
if (str){
  alert(fileURL + " contains: " + str);
}
```

DWfile.remove()

Availability

Dreamweaver 3.

Description

This function moves the specified file to the Recycling Bin or Trash.

Arguments

`fileURL`

- The *`fileURL`* argument, which is expressed as a file:// URL, is the file you want to remove.

Returns

A Boolean value: `true` value if the operation succeeds; `false` otherwise.

Example

The following example uses the `DWfile.getAttributes()` function to determine whether the file is read-only and the `confirm()` function to display a Yes/No dialog box to the user:

```
function deleteFile(){
  var delAnyway = false;
  var selIndex = document.theForm.menu.selectedIndex;

  var selFile = document.theForm.menu.options[selIndex].value;
  if (DWfile.getAttributes(selFile).indexOf('R') != -1){
    delAnyway = confirm('This file is read-only. Delete anyway?');
    if (delAnyway){
      DWfile.remove(selFile);
    }
  }
}
```

DWfile.setAttributes()

Availability

Dreamweaver MX.

Description

This function sets the system-level attributes of a particular file.

Arguments

`fileURL`, *`strAttrs`*

- The *`fileURL`* argument, which is expressed as a file:// URL, identifies the file for which you are setting the attributes.
- The *`strAttrs`* argument specifies the system-level attributes for the file that is identified by the *`fileURL`* argument. The following table describes valid attribute values and their meaning:

Attribute Value	Description
`R`	Read only
`W`	Writable (overrides `R`)
`H`	Hidden
`V`	Visible (overrides `H`)

Acceptable values for the `strAttrs` string are `R`, `W`, `H`, `V`, `RH`, `RV`, `WH`, or `WV`.

You should not use `R` and `W` together because they are mutually exclusive. If you combine them, `R` becomes meaningless, and the file is set as writable (`W`). You should not use `H` and `V` together because they are also mutually exclusive. If you combine them, `H` becomes meaningless, and the file is set as visible (`V`).

If you specify `H` or `V` without specifying an `R` or `W` read/write attribute, the existing read/write attribute for the file is not changed. Likewise, if you specify `R` or `W`, without specifying an `H` or `V` visibility attribute, the existing visibility attribute for the file is not changed.

Returns

Nothing.

DWfile.write()

Availability

Dreamweaver 2.

Description

This function writes the specified string to the specified file. If the specified file does not yet exist, it is created.

Arguments

fileURL, *text*, *{mode}*

- The *fileURL* argument, which is expressed as a file:// URL, is the file to which you are writing.
- The *text* argument is the string to be written.
- The *mode* argument, if it is supplied, must be `"append"`. If this argument is omitted, the contents of the file are overwritten by the string.

Returns

A Boolean value: `true` if the string is successfully written to the file; `false` otherwise.

Example

The following code attempts to write the string `"xxx"` to the mydata.txt file and displays an alert message if the write operation succeeds. It then tries to append the string `"aaa"` to the file and displays a second alert if the write succeeds. After executing this script, the mydata.txt file contains the text `xxxaaa` and nothing else.

```
var fileURL = "file:///c|/temp/mydata.txt";
if (DWfile.write(fileURL, "xxx")){
  alert("Wrote xxx to " + fileURL);
}
if (DWfile.write(fileURL, "aaa", "append")){
  alert("Appended aaa to " + fileURL);
}
```

CHAPTER 3
The HTTP API

Extensions are not limited to working within the local file system. Macromedia Dreamweaver MX 2004 provides a mechanism to get information from and send information to a web server by using hypertext transfer protocol (HTTP). This chapter describes the HTTP API and how to use it.

How the HTTP API works

All functions in the HTTP API are methods of the `MMHttp` object. Most of these functions take a URL as an argument, and most return an object. The default port for URL arguments is 80. To specify a port other than 80, append a colon and the port number to the URL, as shown in the following example:

```
MMHttp.getText("http://www.myserver.com:8025");
```

For functions that return an object, the object has two properties: `statusCode` and `data`.

The `statusCode` property indicates the status of the operation; possible values include, but are not limited to, the following values:

- 200: Status OK
- 400: Unintelligible request
- 404: Requested URL not found
- 405: Server does not support requested method
- 500: Unknown server error
- 503: Server capacity reached

For a comprehensive list of status codes for your server, check with your Internet service provider or system administrator.

The value of the `data` property varies according to the function; possible values are specified in the individual function listings.

Functions that return an object also have a callback version. Callback functions let other functions execute while the web server processes an HTTP request. This capability is useful if you are making multiple HTTP requests from Dreamweaver. The callback version of a function passes its ID and return value directly to the function that is specified as its first argument.

The HTTP API

This section details the functions that are methods of the `MMHttp` object.

MMHttp.clearServerScriptsFolder()

Availability

Dreamweaver MX.

Description

Deletes the _mmServerScripts folder—and all its files—under the root folder for the current site, which can be local or remote. The _mmServerScripts folder is located in Configuration/Connections/Scripts/*`server-model`*/_mmDBScripts folder.

Arguments

`serverScriptsfolder`

- The *`serverScriptsfolder`* argument is a string that names a particular folder, relative to the Configuration folder on the application server, from which you want to retrieve and clear server scripts.

Returns

An object that represents the reply from the server. The `data` property of this object is a string that contains the contents of the deleted scripts. If an error occurs, Dreamweaver reports it in the `statusCode` property of the returned object.

Example

The following code, in a menu command file inside the Configuration/Menus folder, removes all the files from the _mmServerScripts folder when it is called from a menu:

```
<!-- MENU-LOCATION=NONE -->
<html>
<head>
<TITLE>Clear Server Scripts</TITLE>
<SCRIPT SRC="ClearServerScripts.js"></SCRIPT>
<SCRIPT LANGUAGE="javascript">
</SCRIPT>
<body onLoad="MMHttp.clearServerScriptsFolder()">
</body>
</html>
```

MMHttp.clearTemp()

Description

This function deletes all the files in the Configuration/Temp folder, which is located in the Dreamweaver application folder.

Arguments

None.

Returns

Nothing.

Example

The following code, when saved in a file within the Configuration/Shutdown folder, removes all the files from the Configuration/Temp folder when the user quits Dreamweaver:

```
<html>
<head>
<title>Clean Up Temp Files on Shutdown</title>
</head>
<body onLoad="MMHttp.clearTemp()">
</body>
</html>
```

MMHttp.getFile()

Description

This function gets the file at the specified URL and saves it in the Configuration/Temp folder, which is located in the Dreamweaver application folder. Dreamweaver automatically creates subfolders that mimic the folder structure of the server; for example, if the specified file is at www.dreamcentral.com/people/index.html, Dreamweaver stores the index.html file in the People folder inside the www.dreamcentral.com folder.

Arguments

`URL, {prompt}`, *`{saveURL}`*, *`{titleBarLabel}`*

- The *`URL`* argument is an absolute URL on a web server; if http:// is omitted from the URL, Dreamweaver assumes HTTP protocol.
- The *`prompt`* argument, which is optional, is a Boolean value that specifies whether to prompt the user to save the file. If *`saveURL`* is outside the Configuration/Temp folder, a *`prompt`* value of `false` is ignored for security reasons.
- The *`saveURL`* argument, which is optional, is the location on the user's hard disk where the file should be saved, which is expressed as a file:// URL. If prompt is a `true` value or *`saveURL`* is outside the Configuration/Temp folder, the user can override *`saveURL`* in the Save dialog box.
- The *`titleBarLabel`* argument, which is optional, is the label that should appear in the title bar of the Save dialog box.

Returns

An object that represents the reply from the server. The `data` property of this object is a string that contains the location where the file is saved, which is expressed as a file:// URL. Normally, the `statusCode` property of the object contains the status code that is received from the server. However, if a disk error occurs while Dreamweaver is saving the file on the local drive, the `statusCode` property contains an integer that represents one of the following error codes if the operation is not successful:

- 1: Unspecified error
- 2: File not found
- 3: Invalid path
- 4: Number of open files limit reached
- 5: Access denied
- 6: Invalid file handle
- 7: Cannot remove current working folder
- 8: No more folder entries
- 9: Error setting file pointer
- 10: Hardware error
- 11: Sharing violation
- 12: Lock violation
- 13: Disk full
- 14: End of file reached

Example

The following code gets an HTML file, saves all the files in the Configuration/Temp folder, and then opens the local copy of the HTML file in a browser:

```
var httpReply = MMHttp.getFile("http://www.dreamcentral.com/¬
people/profiles/scott.html",
false);
if (httpReply.statusCode == 200){
  var saveLoc = httpReply.data;
  dw.browseDocument(saveLoc);
}
```

MMHttp.getFileCallback()

Description

This function gets the file at the specified URL, saves it in the Configuration/Temp folder inside the Dreamweaver application folder, and then calls the specified function with the request ID and reply result. When saving the file locally, Dreamweaver automatically creates subfolders that mimic the folder structure of the server; for example, if the specified file is at www.dreamcentral.com/people/index.html, Dreamweaver stores the index.html file in the People folder inside the www.dreamcentral.com folder.

Arguments

`callbackFunction`, *`URL`*, *`{prompt}`*, *`{saveURL}`*, *`{titleBarLabel}`*

- The *`callbackFunction`* argument is the name of the JavaScript function to call when the HTTP request is complete.
- The *`URL`* argument is an absolute URL on a web server; if http:// is omitted from the URL, Dreamweaver assumes HTTP protocol.
- The *`prompt`* argument, which is optional, is a Boolean value that specifies whether to prompt the user to save the file. If *`saveURL`* argument specifies a location outside the Configuration/Temp folder, a *`prompt`* value of `false` is ignored for security reasons.
- The *`saveURL`* argument, which is optional, is the location on the user's hard disk where the file should be saved, which is expressed as a file:// URL. If prompt is a `true` value or *`saveURL`* is outside the Configuration/Temp folder, the user can override *`saveURL`* in the Save dialog box.
- The *`titleBarLabel`* argument, which is optional, is the label that should appear in the title bar of the Save dialog box.

Returns

An object that represents the reply from the server. The `data` property of this object is a string that contains the location where the file was saved, which is expressed as a file:// URL. Normally the `statusCode` property of the object contains the status code that is received from the server. However, if a disk error occurs while Dreamweaver is saving the file on the local drive, the `statusCode` property contains an integer that represents an error code. See MMHttp.getFile() for a list of possible error codes.

MMHttp.getText()

Availability

Macromedia Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Retrieves the contents of the document at the specified URL.

Arguments

`URL`, *`{serverScriptsFolder}`*

- The *`URL`* argument is an absolute URL on a web server. If http:// is omitted from the URL, Dreamweaver assumes HTTP protocol.

- The *serverScriptsFolder* argument is an optional string that names a particular folder—relative to the Configuration folder on the application server—from which you want to retrieve server scripts. To retrieve the scripts, Dreamweaver uses the appropriate transfer protocol (such as FTP, WebDAV, or Remote File System). Dreamweaver copies these files to the _mmServerScripts subfolder under the root folder for the current site.

If an error occurs, Dreamweaver reports it in the `statusCode` property of the returned object.

MMHttp.getTextCallback()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Retrieves the contents of the document at the specified URL and passes it to the specified function.

Arguments

callbackFunc, *URL*, *{serverScriptsFolder}*

- The *callbackFunc* argument is the JavaScript function to call when the HTTP request is complete.
- The *URL* argument is an absolute URL on a web server; if http:// is omitted from the URL, Dreamweaver assumes HTTP protocol.
- The *serverScriptsFolder* argument is an optional string that names a particular folder—relative to the Configuration folder on the application server—from which you want to retrieve server scripts. To retrieve the scripts, Dreamweaver uses the appropriate transfer protocol (such as FTP, WebDAV, or Remote File System). Dreamweaver retrieves these files and passes them to the function that *callbackFunc* identifies.

If an error occurs, Dreamweaver MX reports it in the `statusCode` property of the returned object.

MMHttp.postText()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Performs an HTTP post of the specified data to the specified URL. Typically, the data associated with a post operation is form-encoded text, but it could be any type of data that the server expects to receive.

Arguments

URL, *dataToPost*, {*contentType*}, {*serverScriptsFolder*}

- The *URL* argument is an absolute URL on a web server; if http:// is omitted from the URL, Dreamweaver assumes HTTP protocol.

- The *dataToPost* argument is the data to post. If the third argument is `"application/x-www-form-urlencoded"` or omitted, *dataToPost* must be form-encoded according to section 8.2.1 of the RFC 1866 specification (available at www.faqs.org/rfcs/rfc1866.html).
- The *contentType* argument, which is optional, is the content type of the data to post. If omitted, this argument defaults to `"application/x-www-form-urlencoded"`.
- The *serverScriptsFolder* argument is an optional string that names a particular folder—relative to the Configuration folder on the application server—to which you want to post the data. To post the data, Dreamweaver uses the appropriate transfer protocol (such as FTP, WebDAV, or Remote File System).

If an error occurs, Dreamweaver reports it in the `statusCode` property of the returned object.

Example

In the following example of an `MMHttp.postText()` function call, assume that a developer has placed the myScripts.cfm file in a folder named DeployScripts, which is located in the Configuration folder on the local computer:

```
MMHttp.postText(
    "http://ultraqa8/DeployScripts/myScripts.cfm",
    "arg1=Foo",
    "application/x-www-form-urlencoded",
    "Configuration/DeployScripts/"
)
```

When Dreamweaver executes this function call, the following sequence occurs:

1. The myScripts.cfm file in the Configuration/DeployScripts folder on the local computer is copied to another folder named DeployScripts, which is a subfolder of the root folder on the ultraqa8 website. To deploy the files, Dreamweaver uses the protocol specified in the site configuration properties.
2. Dreamweaver uses HTTP protocol to post the `arg1=Foo` data to the web server.
3. As a result of the post request, the web server on ultraqa8 executes the myScripts.cfm script using the `arg1` data.

MMHttp.postTextCallback()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Performs an HTTP post of the text to the specified URL and passes the reply from the server to the specified function. Typically, the data associated with a post operation is form-encoded text, but it could be any type of data that the server expects to receive.

Arguments

callbackFunc, *URL*, *dataToPost*, *{contentType}, {serverScriptsFolder}*

- The *callbackFunc* argument is the name of the JavaScript function to call when the HTTP request is complete.
- The *URL* argument is an absolute URL on a web server; if http:// is omitted from the URL, Dreamweaver assumes HTTP protocol.

- The *`dataToPost`* argument is the data to be posted. If the third argument is `"application/x-www-form-urlencoded"` or omitted, *`data`* must be form-encoded according to section 8.2.1 of the RFC 1866 specification (available at www.faqs.org/rfcs/rfc1866.html).
- The *`contentType`* argument, which is optional, is the content type of the data to be posted. If omitted, this argument defaults to `"application/x-www-form-urlencoded"`.
- The *`serverScriptsFolder`* argument is an optional string. It names a particular folder, relative to the Configuration folder on the application server—to which you want to post the data. To post the data, Dreamweaver uses the appropriate transfer protocol (such as FTP, WebDAV, or Remote File System). Dreamweaver retrieves these data and passes them to the function identified by *`callbackFunc`*.

If an error occurs, Dreamweaver reports it in the `statusCode` property of the returned object.

CHAPTER 4
The Design Notes API

Macromedia Dreamweaver MX 2004, Macromedia Fireworks MX 2004, and Macromedia Flash MX 2004 give web designers and developers a way to store and retrieve extra information about documents—information such as review comments, change notes, or the source file for a GIF or JPEG—in files that are called Design Notes.

MMNotes is a C shared library that lets extensions authors read and write Design Notes files. As with the DWfile shared library, MMNotes has a JavaScript API that lets you call the functions in the library from objects, commands, behaviors, floating panels, Property inspectors, and data translators.

MMNotes also has a C API that lets other applications read and write Design Notes files. The MMNotes shared library can be used independently, even if Dreamweaver is not installed.

For more information about using the Design Notes feature from within Dreamweaver, see *Using Dreamweaver*.

How Design Notes work

Each Design Notes file stores information for a single document. If one or more documents in a folder has an associated Design Notes file, Dreamweaver creates a _notes subfolder where Design Notes files can be stored. The _notes folder and the Design Notes files that it contains are not visible in the Site panel, but they appear in the Finder (Macintosh) or Windows Explorer. A Design Notes filename comprises the main filename plus the .mno extension. For example, the Design Notes file that is associated with avocado8.gif is avocado8.gif.mno.

Design Notes files are XML files that store information in a series of key/value pairs. The key describes the type of information that is being stored, and the value represents the information. Keys are limited to 64 characters.

The following example shows the Design Notes file for foghorn.gif.mno:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<info>
  <infoitem key="FW_source" value="file:///C|sites/¬
  dreamcentral/images/sourceFiles/foghorn.png" />
  <infoitem key="Author" value="Heidi B." />
  <infoitem key="Status" value="Final draft, approved ¬
  by Jay L." />
</info>
```

The Design Notes JavaScript API

All functions in the Design Notes JavaScript API are methods of the `MMNotes` object.

MMNotes.close()

Description

This function closes the specified Design Notes file and saves any changes. If all the key/value pairs are removed, Dreamweaver deletes the Design Notes file. If it is the last Design Notes file in the _notes folder, Dreamweaver also deletes the folder.

Note: Always call the `MMNotes.close()` function when you finish with Design Notes so Dreamweaver writes to the file.

Arguments

`fileHandle`

- The *`fileHandle`* argument is the file handle that the `MMNotes.open()` function returns.

Returns

Nothing.

Example

See "MMNotes.set()" on page 56.

MMNotes.filePathToLocalURL()

Description

This function converts the specified local drive path to a file:// URL.

Arguments

`drivePath`

- The *`drivePath`* argument is a string that contains the full drive path.

Returns

A string that contains the file:// URL for the specified file.

Example

A call to `MMNotes.filePathToLocalURL('C:\sites\webdev\index.htm')` returns `"file:///c|/sites/webdev/index.htm"`.

MMNotes.get()

Description

This function gets the value of the specified key in the specified Design Notes file.

Arguments

`fileHandle`, *`keyName`*

- The *`fileHandle`* argument is the file handle that `MMNotes.open()` returns.
- The *`keyName`* argument is a string that contains the name of the key.

Returns

A string that contains the value of the key.

Example

See "MMNotes.getKeys()" on page 53.

MMNotes.getKeyCount()

Description

This function gets the number of key/value pairs in the specified Design Notes file.

Arguments

`fileHandle`

- The *`fileHandle`* argument is the file handle that the `MMNotes.open()` function returns.

Returns

An integer that represents the number of key/value pairs in the Design Notes file.

MMNotes.getKeys()

Description

This function gets a list of all the keys in a Design Notes file.

Arguments

`fileHandle`

- The *`fileHandle`* argument is the file handle that the `MMNotes.open()` function returns.

Returns

An array of strings where each string contains the name of a key.

Example

The following code might be used in a custom floating panel to display the Design Notes information for the active document:

```
var noteHandle = MMNotes.open(dw.getDocumentDOM().URL);
var theKeys = MMNotes.getKeys(noteHandle);
var noteString = "";
var theValue = "";
for (var i=0; i < theKeys.length; i++){
  theValue = MMNotes.get(noteHandle,theKeys[i]);
  noteString +=  theKeys[i] + " = " theValue + "\n";
}
document.theForm.bigTextField.value = noteString;
// always close noteHandle
MMNotes.close(noteHandle);
```

MMNotes.getSiteRootForFile()

Description

This function determines the site root for the specified Design Notes file.

Arguments

`fileURL`

- The *`fileURL`* argument, which is expressed as a file:// URL, is the path to a local file.

Returns

A string that contains the path of the Local Root folder for the site, which is expressed as a file:// URL, or an empty string if Dreamweaver is not installed or the Design Notes file is outside any site that is defined with Dreamweaver. This function searches for all the sites that are defined in Dreamweaver.

MMNotes.getVersionName()

Description

This function gets the version name of the MMNotes shared library, which indicates the application that implemented it.

Arguments

None.

Returns

A string that contains the name of the application that implemented the MMNotes shared library.

Example

Calling the `MMNotes.getVersionName()` function from a Dreamweaver command, object, behavior, Property inspector, floating panel, or data translator returns `"Dreamweaver"`. Calling the `MMNotes.getVersionName()` function from Fireworks also returns `"Dreamweaver"` because Fireworks uses the same version of the library, which was created by the Dreamweaver engineering team.

MMNotes.getVersionNum()

Description

This function gets the version number of the MMNotes shared library.

Arguments

None.

Returns

A string that contains the version number.

MMNotes.localURLToFilePath()

Description

This function converts the specified file:// URL to a local drive path.

Arguments

`fileURL`

- The *`fileURL`* argument, which is expressed as a file:// URL, is the path to a local file.

Returns

A string that contains the local drive path for the specified file.

Example

A call to `MMNotes.localURLToFilePath('file:///MacintoshHD/images/moon.gif')` returns `"MacintoshHD:images:moon.gif"`.

MMNotes.open()

Description

This function opens the Design Notes file that is associated with the specified file or creates one if none exists.

Arguments

`filePath`, *`{bForceCreate}`*

- The *`filePath`* argument, which is expressed as a file:// URL, is the path to the main file with which the Design Notes file is associated.
- The *`bForceCreate`* argument is a Boolean value that indicates whether to create the note even if Design Notes is turned off for the site or if the *`filePath`* argument is not associated with any site.

Returns

The file handle for the Design Notes file or 0 if the file was not opened or created.

Example

See "MMNotes.set()" on page 56.

MMNotes.remove()

Description

The function removes the specified key (and its value) from the specified Design Notes file.

Arguments

`fileHandle`, *`keyName`*

- The *`fileHandle`* argument is the file handle that the `MMNotes.open()` function returns.
- The *`keyName`* argument is a string that contains the name of the key to remove.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise.

MMNotes.set()

Description

This function creates or updates one key/value pair in a Design Notes file.

Arguments

`fileHandle`, *`keyName`*, *`valueString`*

- The *`fileHandle`* argument is the file handle that the `MMNotes.open()` function returns.
- The *`keyName`* argument is a string that contains the name of the key.
- The *`valueString`* argument is a string that contains the value.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise.

Example

The following example opens the Design Notes file that is associated with a file in the dreamcentral site called peakhike99/index.html, adds a new key/value pair, changes the value of an existing key, and then closes the Design Notes file:

```
var noteHandle = MMNotes.open('file:///c|/sites/dreamcentral/
peakhike99/index.html',true);
if(noteHandle > 0){
  MMNotes.set(noteHandle,"Author","M. G. Miller");
  MMNotes.set(noteHandle,"Last Changed","August 28, 1999");
  MMNotes.close(noteHandle);
}
```

The Design Notes C API

In addition to the JavaScript API, the MMNotes shared library also exposes a C API that lets other applications create Design Notes files. It is not necessary to call these C functions directly if you use the MMNotes shared library in Dreamweaver because the JavaScript versions of the functions call them.

This section contains descriptions of the functions, their arguments, and their return values. You can find definitions for the functions and data types in the MMInfo.h file in the Extending/c_files folder inside the Dreamweaver application folder.

void CloseNotesFile()

Description

This function closes the specified Design Notes file and saves any changes. If all key/value pairs are removed from the Design Note file, Dreamweaver deletes it. Dreamweaver deletes the _notes folder when the last Design Notes file is deleted.

Arguments

`FileHandle` *`noteHandle`*

- The *`noteHandle`* argument is the file handle that the `OpenNotesFile()` function returns.

Returns

Nothing.

BOOL FilePathToLocalURL()

Description

This function converts the specified local drive path to a file:// URL.

Arguments

`const char*` *`drivePath`*, `char*` *`localURLBuf`*, `int` *`localURLMaxLen`*

- The *`drivePath`* argument is a string that contains the full drive path.
- The *`localURLBuf`* argument is the buffer where the file:// URL is stored.
- The *`localURLMaxLen`* argument is the maximum size of *`localURLBuf`*.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. The *`localURLBuf`* argument receives the file:// URL value.

BOOL GetNote()

Description

This function gets the value of the specified key in the specified Design Notes file.

Arguments

`FileHandle` *`noteHandle`*, `const char` *`keyName[64]`*, `char*` *`valueBuf`*, `int` *`valueBufLength`*

- The *`noteHandle`* argument is the file handle that the `OpenNotesFile()` function returns.
- The *`keyName[64]`* argument is a string that contains the name of the key.
- The *`valueBuf`* argument is the buffer where the value is stored.
- The *`valueBufLength`* argument is the integer that `GetNoteLength(`*`noteHandle`*, *`keyName`*`)` returns, which indicates the maximum length of the value buffer.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. The *`valueBuf`* argument receives the value of the key.

Example

The following code gets the value of the `comments` key in the Design Notes file that is associated with the welcome.html file:

```
FileHandle noteHandle = OpenNotesFile("file:///c|/sites/avocado8/¬
iwjs/welcome.html");
if(noteHandle > 0){
  int valueLength = GetNoteLength( noteHandle, "comments");
  char* valueBuffer = new char[valueLength + 1];
  GetNote(noteHandle, "comments", valueBuffer, valueLength + 1);
  printf("Comments: %s",valueBuffer);
  CloseNotesFile(noteHandle);
}
```

int GetNoteLength()

Description

This function gets the length of the value that is associated with the specified key.

Arguments

`FileHandle` *`noteHandle`*, `const char` *`keyName[64]`*

- The *`noteHandle`* argument is the file handle that the `OpenNotesFile()` function returns.
- The *`keyName[64]`* argument is a string that contains the name of the key.

Returns

An integer that represents the length of the value.

Example

See “BOOL GetNote()” on page 58.

int GetNotesKeyCount()

Description

This function gets the number of key/value pairs in the specified Design Notes file.

Arguments

`FileHandle` *`noteHandle`*

- The *`noteHandle`* argument is the file handle that the `OpenNotesFile()` function returns.

Returns

An integer that represents the number of key/value pairs in the Design Notes file.

BOOL GetNotesKeys()

Description

This function gets a list of all the keys in a Design Notes file.

Arguments

`FileHandle` *`noteHandle`*, `char*` *`keyBufArray[64]`*, `int` *`keyArrayMaxLen`*

- The *`noteHandle`* argument is the file handle that `OpenNotesFile()` returns.
- The *`keyBufArray[64]`* argument is the buffer array where the keys are stored.
- The *`keyArrayMaxLen`* argument is the integer that `GetNotesKeyCount(`*`noteHandle`*`)` returns, indicating the maximum number of items in the key buffer array.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. The *`keyBufArray`* argument receives the key names.

Example

The following code prints the key names and values of all the keys in the Design Notes file that are associated with the welcome.html file:

```
typedef char[64] InfoKey;
FileHandle noteHandle = OpenNotesFile("file:///c|/sites/avocado8/¬
iwjs/welcome.html");
if (noteHandle > 0){
  int keyCount = GetNotesKeyCount(noteHandle);
  if (keyCount <= 0)
    return;
  InfoKey* keys = new InfoKey[keyCount];
  BOOL succeeded = GetNotesKeys(noteHandle, keys, keyCount);

  if (succeeded){
    for (int i=0; i < keyCount; i++){
      printf("Key is: %s\n", keys[i]);
      printf("Value is: %s\n\n", GetNote(noteHandle, keys[i]);
    }
  }
  delete []keys;
}
CloseNotesFile(noteHandle);
```

BOOL GetSiteRootForFile()

Description

This function determines the site root for the specified Design Notes file.

Arguments

`const char*` *`filePath`*, `char*` *`siteRootBuf`*, `int` *`siteRootBufMaxLen`*, `{InfoPrefs*` *`infoPrefs`*`}`

- The *`filePath`* argument is the file://URL of the file for which you want the site root.
- The *`siteRootBuf`* argument is the buffer where the site root is stored.
- The *`siteRootBufMaxLen`* argument is the maximum size of the buffer that *`siteRootBuf`* references.
- The *`infoPrefs`* argument, which is optional, is a reference to a `struct` in which the preferences for the site are stored.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. The *`siteRootBuf`* argument receives the address of the buffer that stores the site root. If you specify the *`infoPrefs`* argument, the function also returns the Design Notes preferences for the site. The `InfoPrefs` struct has two variables: `bUseDesignNotes` and `bUploadDesignNotes`, both of type `BOOL`.

BOOL GetVersionName()

Description

This function gets the version name of the MMNotes shared library, which indicates the application that implemented it.

Arguments

`char*` *`versionNameBuf`*, `int` *`versionNameBufMaxLen`*

- The *`versionNameBuf`* argument is the buffer where the version name is stored.
- The *`versionNameBufMaxLen`* argument is the maximum size of the buffer that the *`versionNameBuf`* argument references.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. Dreamweaver stores `"Dreamweaver"` in *`versionNameBuf`* argument.

BOOL GetVersionNum()

Description

This function gets the version number of the MMNotes shared library, which lets you determine whether certain functions are available.

Arguments

`char*` *`versionNumBuf`*, `int` *`versionNumBufMaxLen`*

- The *`versionNumBuf`* argument is the buffer where the version number is stored.
- The *`versionNumBufMaxLen`* argument is the maximum size of the buffer that *`versionNumBuf`* references.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. The *`versionNumBuf`* argument stores the version number.

BOOL LocalURLToFilePath()

Description

This function converts the specified file:// URL to a local drive path.

Arguments

`const char*` *`localURL`*, `char*` *`drivePathBuf`*, `int` *`drivePathMaxLen`*

- The *`localURL`* argument, which is expressed as a file:// URL, is the path to a local file.
- The *`drivePathBuf`* argument is the buffer where the local drive path is stored.
- The *`drivePathMaxLen`* argument is the maximum size of the buffer that the *`drivePathBuf`* argument references.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise. The *`drivePathBuf`* argument receives the local drive path.

FileHandle OpenNotesFile()

Description

This function opens the Design Notes file that is associated with the specified file or creates one if none exists.

Arguments

`const char*` *`localFileURL`*, `{BOOL` *`bForceCreate`*`}`

- The *`localFileURL`* argument, which is expressed as a file:// URL, is a string that contains the path to the main file with which the Design Notes file is associated.
- The *`bForceCreate`* argument is a Boolean value that indicates whether to create the Design Notes file even if Design Notes is turned off for the site or if the path specified by the *`localFileURL`* argument is not associated with any site.

FileHandle OpenNotesFilewithOpenFlags()

Description

This function opens the Design Notes file that is associated with the specified file or creates one if none exists. You can open the file in read-only mode.

Arguments

`const char*` *`localFileURL`*, `{BOOL` *`bForceCreate`*`}`, `{BOOL` *`bReadOnly`*`}`

- The *`localFileURL`* argument, which is expressed as a file:// URL, is a string that contains the path to the main file with which the Design Notes file is associated.
- The *`bForceCreate`* argument is a Boolean value that indicates whether to create the Design Notes file even if Design Notes are turned off for the site or the path is not associated with any site. The default value is `false`. This argument is optional, but you need to specify it if you specify the third argument.
- The *`bReadOnly`* argument, which is optional, is a Boolean value that indicates whether to open the file in read-only mode. The default value is `false`. You can specify the *`bReadOnly`* argument starting in version 2 of the MMNotes.dll file.

BOOL RemoveNote()

Description

This function removes the specified key (and its value) from the specified Design Notes file.

Arguments

`FileHandle` *`noteHandle`*, `const char` *`keyName[64]`*

- The *`noteHandle`* argument is the file handle that the `OpenNotesFile()` function returns.
- The *`keyName[64]`* argument is a string that contains the name of the key to remove.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise.

BOOL SetNote()

Description

This function creates or updates one key/value pair in a Design Notes file.

Arguments

`FileHandle` *`noteHandle`*, `const char` *`keyName[64]`*, `const char*` *`value`*

- The *`noteHandle`* argument is the file handle that the `OpenNotesFile()` function returns.
- The *`keyName[64]`* argument is a string that contains the name of the key.
- The *`value`* argument is a string that contains the value.

Returns

A Boolean value: `true` indicates the operation is successful; `false` otherwise.

CHAPTER 5
Fireworks Integration

FWLaunch is a C shared library that gives authors of objects, commands, behaviors, and Property inspectors the ability to communicate with Macromedia Fireworks MX 2004. Using FWLaunch, you write JavaScript to open the Fireworks user interface (UI) and provide commands to Fireworks through its own JavaScript API documented in the *Extending Fireworks* manual. For general information on how C libraries interact with the JavaScript interpreter in Macromedia Dreamweaver MX 2004, see *Extending Dreamweaver* for details on C-level extensibility.

The FWLaunch API

The FWLaunch object lets extensions open Fireworks, perform Fireworks operations using the Fireworks JavaScript API, and then return values back to Dreamweaver. This chapter describes the FWLaunch Communication API and how to use it.

FWLaunch.bringDWToFront()

Availability

Dreamweaver 3, Fireworks 3.

Description

This function brings Dreamweaver to the front.

Arguments

None.

Returns

Nothing.

FWLaunch.bringFWToFront()

Availability

Dreamweaver 3, Fireworks 3.

Description

This function brings Fireworks to the front if it is running.

Arguments

None.

Returns

Nothing.

FWLaunch.execJsInFireworks()

Availability

Dreamweaver 3, Fireworks 3.

Description

This function passes the specified JavaScript, or a reference to a JavaScript file, to Fireworks to execute.

Arguments

`javascriptOrFileURL`

- The `javascriptOrFileURL` argument, which is expressed as a file:// URL, is either a string of literal JavaScript or the path to a JavaScript file.

Returns

A cookie object if the JavaScript passes successfully or a nonzero error code that indicates one of the following errors occurred:

- Invalid usage, which indicates that the `javascriptOrFileURL` argument is specified as a `null` value or as an empty string, or the path to the JS or JSF file is invalid.
- File I/O error, which indicates that Fireworks cannot create a Response file because the disk is full.
- Error notifying Dreamweaver that the user is not running a valid version of Dreamweaver (version 3 or later).
- Error starting Fireworks process, which indicates that the function does not open a valid version of Fireworks (version 3 or later).
- User cancelled the operation.

FWLaunch.getJsResponse()

Availability

Dreamweaver 3, Fireworks 3.

Description

This function determines whether Fireworks is still executing the JavaScript passed to it by the `FWLaunch.execJsInFireworks()` function, whether the script completed successfully, or whether an error occurred.

Arguments

`progressTrackerCookie`

- The `progressTrackerCookie` argument is the cookie object that the `FWLaunch.execJsInFireworks()` function returns.

Returns

A string that contains the result of the script passed to the `FWLaunch.execJsInFireworks()` function if the operation completes successfully, a `null` value if Fireworks is still executing the JavaScript, or a nonzero error code that indicates one of the following errors occurred:

- Invalid usage, which indicates that a JavaScript error occurred while Fireworks executed the script.
- File I/O error, which indicates that Fireworks cannot create a Response file because the disk is full.
- Error notifying Dreamweaver that the user is not running a valid version of Dreamweaver (version 3 or later).
- Error starting Fireworks process, which indicates that the function does not open a valid version of Fireworks (version 3 or later).
- User cancelled the operation.

Returns

The following code passes the string `"prompt('Please enter your name:')"` to `FWLaunch.execJsInFireworks()` and checks for the result:

```
var progressCookie = FWLaunch.execJsInFireworks("prompt('Please enter your
  name:')");
var doneFlag = false;
while (!doneFlag){
  // check for completion every 1/2 second
  setTimeout('checkForCompletion()',500);
}

function checkForCompletion(){
  if (progressCookie != null) {
    var response = FWLaunch.getJsResponse(progressCookie);
    if (response != null) {
      if (typeof(response) == "number") {
        // error or user-cancel, time to close the window
        // and let the user know we got an error
        window.close();
        alert("An error occurred.");
      }else{
            // got a valid response!
            alert("Nice to meet you, " + response);
            window.close();
         }
            doneFlag = true;
      }
   }
}
```

FWLaunch.mayLaunchFireworks()

Availability

Dreamweaver 2, Fireworks 2.

Description

This function determines whether it is possible to open a Fireworks optimization session.

Arguments

None.

Returns

A Boolean value that indicates whether the platform is Windows or the Macintosh; if it is the Macintosh, the value indicates if another Fireworks optimization session is already running.

FWLaunch.optimizeInFireworks()

Availability

Dreamweaver 2, Fireworks 2.

Description

This function opens a Fireworks optimization session for the specified image.

Arguments

`docURL, imageURL, {targetWidth}, {targetHeight}`

- The *`docURL`* argument is the path to the active document, which is expressed as a file:// URL.
- The *`imageURL`* argument is the path to the selected image. If the path is relative, it is relative to the path that you specify in the *`docURL`* argument.
- The *`targetWidth`* argument, which is optional, defines the width to which the image should be resized.
- The *`targetHeight`* argument, which is optional, defines the height to which the image should be resized.

Returns

Zero, if a Fireworks optimization session successfully opens for the specified image; otherwise, a nonzero error code that indicates one of the following errors occurred:

- Invalid usage, which indicates that the *`docURL`* argument, the *`imageURL`* argument, or both, are specified as a `null` value or an empty string.
- File I/O error, which indicates that Fireworks cannot create a response file because the disk is full.
- Error notifying Dreamweaver that the user is not running a valid version of Dreamweaver (version 2 or later).
- Error starting Fireworks process, which indicates that the function does not open a valid version of Fireworks (version 2 or later).
- User cancelled the operation.

FWLaunch.validateFireworks()

Availability

Dreamweaver 2, Fireworks 2.

Description

This function looks for the specified version of Fireworks on the user's hard disk.

Arguments

`{versionNumber}`

- The *`versionNumber`* argument is an optional floating-point number that is greater than or equal to 2; it represents the required version of Fireworks. If this argument is omitted, the default is 2.

Returns

A Boolean value that indicates whether the specified version of Fireworks was found.

Example

The following code checks whether Fireworks is installed:

```
if (FWLaunch.validateFireworks(6.0)){
  alert("Fireworks 6.0 or later is installed.");
}else{
  alert("Fireworks 6.0 is not installed.");
}
```

A simple FWLaunch communication example

The following command asks Fireworks to prompt the user for their name and returns the name to Dreamweaver:

```
<html>
<head>
<title>Prompt in Fireworks</title>
<meta http-equiv="Content-Type" content="text/html; ¬
charset=iso-8859-1">
<script>

function commandButtons(){
  return new Array("Prompt", "promptInFireworks()", "Cancel", ¬
  "readyToCancel()", "Close","window.close()");
}

var gCancelClicked = false;
var gProgressTrackerCookie = null;

function readyToCancel() {
  gCancelClicked = true;
}

function promptInFireworks() {
  var isFireworks3 = FWLaunch.validateFireworks(3.0);
  if (!isFireworks3) {
    alert("You must have Fireworks 3.0 or later to use this ¬
    command");
  return;
}

  // Tell Fireworks to execute the prompt() method.
  gProgressTrackerCookie = FWLaunch.execJsInFireworks¬
  ("prompt('Please enter your name:')");

  // null means it wasn't launched, a number means an error code
  if (gProgressTrackerCookie == null || ¬
  typeof(gProgressTrackerCookie) == "number") {
    window.close();
    alert("an error occurred");
    gProgressTrackerCookie = null;
  } else {
    // bring Fireworks to the front
    FWLaunch.bringFWToFront();
    // start the checking to see if Fireworks is done yet
    checkOneMoreTime();
  }
}

function checkOneMoreTime() {
  // Call checkJsResponse() every 1/2 second to see if Fireworks
   // is done yet
  window.setTimeout("checkJsResponse();", 500);
}

function checkJsResponse() {
  var response = null;

  // The user clicked the cancel button, close the window
  if (gCancelClicked) {
      window.close();
      alert("cancel clicked");
    } else {
      // We're still going, ask Fireworks how it's doing
      if (gProgressTrackerCookie != null)
        response = ¬
        FWLaunch.getJsResponse(gProgressTrackerCookie);

      if (response == null) {
        // still waiting for a response, call us again in 1/2 a
        // second
        checkOneMoreTime();
      } else if (typeof(response) == "number") {
        // if the response was a number, it means an error
        // occurred
        // the user cancelled in Fireworks
        window.close();
        alert("an error occurred.");

      } else {
        // got a valid response!  This return value might not
        // always be a useful one, since not all functions in
        // Fireworks return a string, but we know this one does,
        // so we can show the user what we got.
        window.close();
        FWLaunch.bringDWToFront();  // bring Dreamweaver to the ¬
        front
        alert("Nice to meet you, " + response + "!");
      }
    }
  }

</script>
</head>
<body>
<form>
<table width="313" nowrap>
<tr>
<td>This command asks Fireworks to execute the prompt() ¬
function. When you click Prompt, Fireworks comes forward and ¬
asks you to enter a value into a dialog box. That value is then ¬
returned to Dreamweaver and displayed in an alert.</td>
</tr>
</table>
</form>
</body>
</html>
```

CHAPTER 6
Flash Integration

Macromedia Dreamweaver MX 2004 now provides support for Macromedia Flash XM 2004 elements as well as continuing support for the Flash Object API, which leverages the Flash Generator Template file to create new Flash objects. This chapter describes ways of working with Flash elements (SWC files), and also provides details for the creation of Flash objects (SWF files) from Flash Generator templates (SWT files).

For information about simply adding Flash content to Dreamweaver objects or commands, see *Extending Dreamweaver*.

How Flash elements work

Flash elements are packaged as SWC files. A SWC file is a compiled component movie clip that is generated by Flash for use by Macromedia and third-party products. Dreamweaver can make these components available to users through the Insert bar, Insert menu, or a toolbar. You create Flash elements using the Flash authoring tool, but Dreamweaver can parse properties of a Flash element and express them through the `param` tag (a child of the `object` tag). Users can then edit the `param` tag attributes to change the properties of the element as it is published (for more information about working with component properties in Dreamweaver, see *Using Dreamweaver*).

Inserting Flash elements

Flash elements are installed through the Extension Manager. Dreamweaver adds Flash elements to documents in the same manner as the objects that are available on the Insert bar or the Insert menu (for details about working with Dreamweaver objects, see “Insert Bar Objects” in *Extending Dreamweaver*). By clicking on objects on the Insert bar or selecting menu options from the Insert menu, users can add strings of code to documents. Flash elements are available to users through the Insert bar or the Insert menu (meaning you can add a valid Flash element file that is already installed in the Configuration/Objects/FlashElements folder or one of its subfolders to the Insert bar or Insert menu). Extension developers can use the JavaScript function dom.insertFlashElement() in the object definition file to add available Flash elements to a document. When the user selects the Flash element object, Dreamweaver unpacks the SWC file, which contains Flash content (SWF file) and a file that details the parameters the user can edit. Dreamweaver then inserts the SWF file into the user’s document.

Adding a Flash element to the Insert Bar

As with other objects, you add a Flash element to the Insert Bar using the `button` tag. However, a `button` tag for a Flash element must have both `file` and `command` attributes to add the element successfully to the document (see the button tag details in “Insert Bar Objects” in *Extending Dreamweaver*). Use the `file` attribute to tell Dreamweaver where the source file for the element resides relative to the Objects folder. Then, use the `command` attribute to tell Dreamweaver to use the `dom.insertFlashElement()` function when the user clicks the button on the Insert bar.

The following example shows the code to place in the inserbar.xml file (either as a child of the appropriate `category` or `menubutton` tag, depending on where you want the Flash element button to appear):

```
<button id="FlashElement_Nav"
name="Navigation"
file="FlashElements\nav.swc"
command="dw.getDocumentDOM().insertFlashElement('nav.swc')" />
```

Note: The image on the Insert bar for the Flash element is determined within the SWC file. Also, the *`button`* tag for a Flash element object must have a file attribute defined.

Adding a Flash Element to a menu

A Flash element can also reside on the Insert menu, or on other menus, in Dreamweaver. Use the JavaScript function dom.insertFlashElement() with the menus.xml file format (see “Menus and Menu Commands” in *Extending Dreamweaver*) to specify the Flash element menu item location. The following example shows how code within the menus.xml file makes a Navigation Flash element available on the Insert > Flash Element menu:

```
<menuitem name="Navigation"
key=""command="dw.getDocumentDOM().insertFlashElement('nav.swc')"
enabled="(dw.getFocus() != 'browser') && (dw.getDocumentDOM() != null && ¬
   dw.getDocumentDOM().getParseMode() == 'html')"
id="DWMenu_Insert_FlashElement_Nav" />
```

The Flash Objects API

The Flash Objects API lets extension developers build objects that create simple Flash content through Flash Generator. This API provides a way to set parameters in a Flash Generator template and output a SWF or image file. The API lets you create new Flash objects as well as read and manipulate existing Flash objects. The Flash button and Flash text features are built using this API.

The SWT file is a Flash Generator Template file, which contains all the information you need to construct a Flash Object file. These API functions let you create a new SWF file (or image file) from a SWT file by replacing the parameters of the SWT file with real values. For more information on Flash, see the Flash documentation. The following functions are methods of the `SWFFile` object.

SWFFile.createFile()

Description

This function generates a new Flash Object file with the specified template and array of parameters. It also creates a GIF, PNG, JPEG, and MOV version of the title if filenames for those formats are specified.

If you want to specify an optional parameter that follows optional parameters that you do not want to include, you need to specify empty strings for the unused parameters. For example, if you want to specify a PNG file, but not a GIF file, you need to specify an empty string before specifying the PNG filename.

Arguments

`templateFile`, *`templateParams`*, *`swfFileName`*, *`{gifFileName}`*, *`{pngFileName}`*, *`{jpgFileName}`*, *`{movFileName}`*, *`{generatorParams}`*

- The *`templateFile`* argument is a path to a template file, which is expressed as a file:// URL. This file can be a SWT file.
- The *`templateParams`* argument is an array of name/value pairs where the names are the parameters in the SWT file, and the values are what you want to specify for those parameters. For Dreamweaver to recognize a SWF file as a Flash object, the first parameter must be `"dwType"`. Its value should be a string that represents the name of the object type, such as `"Flash Text"`.
- The *`swfFileName`* argument, which is expressed as a file:// URL, is the output filename of an SWF file or an empty string to ignore.
- The *`gifFileName`* argument, which is expressed as a file:// URL, is the output filename of a GIF file. This argument is optional.
- The *`pngFileName`* argument, which is expressed as a file:// URL, is the output filename of a PNG file. This argument is optional.
- The *`jpgFileName`* argument, which is expressed as a file:// URL, is the output filename of a JPEG file. This argument is optional.
- The *`movFileName`* argument, which is expressed as a file:// URL, is the output filename of a QuickTime file. This argument is optional.
- The *`generatorParams`* argument is an array of strings that represents optional Generator command line flags. This argument is optional. Each flag's data items must follow it in the array. Some commonly used flags are listed in the following table:

Option Flag	Data	Description	Example
`-defaultsize`	Width, height	Sets the output image size to the specified width and height	`"-defaultsize", "640", "480"`
`-exactFit`	None	Stretches the contents in the output image to fit exactly into the specified output size	`"-exactFit"`

Returns

A string that contains one of the following values:

- `"noError"` means the call completed successfully.
- `"invalidTemplateFile"` means the specified template file is invalid or not found.
- `"invalidOutputFile"` means at least one of the specified output filenames is invalid.
- `"invalidData"` means that one or more of the *`templateParams`* name/value pairs is invalid.
- `"initGeneratorFailed"` means the Generator cannot be initialized.
- `"outOfMemory"` means there is insufficient memory to complete the operation.
- `"unknownError"` means an unknown error occurred.

Example

The following JavaScript creates a Flash object file of type `"myType"`, which replaces any occurrences of the string `"text"` inside the Template file with the string, `"Hello World"`. It creates a GIF file as well as a SWF file.

```
var params = new Array;
params[0] = "dwType";
params[1] = "myType";
params[2] = "text";
params[3] = "Hello World";
errorString = SWFFile.createFile( "file:///MyMac/test.swt", ¬
params, "file:///MyMac/test.swf", "file:///MyMac/test.gif");
```

SWFFile.getNaturalSize()

Description

This function returns the natural size of any Flash content.

Arguments

`fileName`

- The *`fileName`* argument, which is expressed as a file:// URL, is a path to the Flash content.

Returns

An array that contains two elements that represent the width and the height of the Flash content or a `null` value if the file is not a Flash file.

SWFFile.getObjectType()

Description

This function returns the Flash object type; the value that passed in the `dwType` parameter when the `SWFFile.createFile()` function created the file.

Arguments

`fileName`

- The *`fileName`* argument, which is expressed as a file:// URL, is a path to a Flash Object file. This file is usually a SWF file.

Returns

A string that represents the object type, or `null` if the file is not a Flash Object file or if the file cannot be found.

Example

The following code checks to see if the test.swf file is a Flash object of type `myType`:

```
if ( SWFFile.getObjectType("file:///MyMac/test.swf") == ¬
"myType" ){
  alert ("This is a myType object.");
}else{
  alert ("This is not a myType object.");
}
```

SWFFile.readFile()

Description

This function reads a Flash Object file.

Arguments

`fileName`

- The *`fileName`* argument, which is expressed as a file:// URL, is a path to a Flash Object file.

Returns

An array of strings where the first array element is the full path to the template SWT file. The following strings represent the parameters (name/value pairs) for the object. Each name is followed in the array by its value. The first name/value pair is `"dwType"`, followed by its value. The function returns a `null` value if the file cannot be found or if it is not a Flash Object file.

Example

Calling `var params = SWFFile.readFile("file:///MyMac/test.swf")` returns the following values in the parameters array:

```
"file:///MyMac/test.swt"  // template file used to create this .swf file
"dwType"                  // first parameter
"myType"                  // first parameter value
"text"                    // second parameter
"Hello World"             // second parameter value
```

CHAPTER 7
The Database API

Functions in the Database API let you manage database connections and access information that is stored in databases. The Database API is divided by two distinct purposes: managing and accessing database connections.

In managing database connections, you can get the user name and password needed to make a connection to a database, open up a database connection dialog box, and so on.

In accessing database information, you can, for example, retrieve metadata that describes the schema or structure of a database. This metadata includes information such as the names of tables, columns, stored procedures, and views. You can also show the results of executing a database query or stored procedure. When accessing a database through this API, you use structured query language (SQL) statements.

Database API functions are used at design time when users are building web applications, not at runtime when the web application is deployed.

You can use these functions in any extension. In fact, the Macromedia Dreamweaver MX 2004 Server Behavior, Data Format, and Data Sources API functions all use these database functions.

How Database API functions work

The following example shows how the server behavior function, `getDynamicBindings()`, is defined for Recordset.js. This example uses the `MMDB.getColumnAndTypeList()` function:

```
function getDynamicBindings(ss)
{
  var serverModel = dw.getDocumentDOM().serverModel.getServerName();
  var bindingsAndTypeArray = new Array();
  var connName  =  ss.connectionName;
  var statement = ss.source;
  var rsName    = ss.rsName;

  // remove SQL comments
  statement = statement.replace(/\/\*[\S\s]*?\*\//g, " ");
  var bIsSimple = ParseSimpleSQL(statement);
  statement = stripCFIFSimple(statement);

  if (bIsSimple) {
  statement = RemoveWhereClause(statement,false);
  } else {
  var pa = new Array();
```

```
    if (ss.ParamArray != null) {
    for (var i = 0; i < ss.ParamArray.length; i++) {
      pa[i] = new Array();
      pa[i][0] = ss.ParamArray[i].name;
      pa[i][1] = ss.ParamArray[i].value;
    }
    }

  var statement = replaceParamsWithVals(statement, pa, serverModel);
}
  bindingsAndTypeArray = MMDB.getColumnAndTypeList(connName, statement);
  return bindingsAndTypeArray;
}
```

Database connection functions

Database connection functions let you make and manage any connection, including the Dreamweaver-provided ADO, ColdFusion, and JDBC connections. These functions interface with the Connection Manager only; they do not access a database. For functions that access a database, see “Database access functions” on page 91.

MMDB.deleteConnection()

Availability

Dreamweaver MX.

Description

This function deletes the named database connection.

Arguments

connName

- The *connName* argument is the name of the database connection as it is specified in the Connection Manager. This argument identifies, by name, the database connection to delete.

Returns

Nothing.

Example

The following example deletes a database connection:

```
function clickedDelete()
{
  var selectedObj = dw.serverComponents.getSelectedNode();
  if (selectedObj && selectedObj.objectType=="Connection")
  {
    var connRec = MMDB.getConnection(selectedObj.name);
    if (connRec)
    {
      MMDB.deleteConnection(selectedObj.name);
      dw.serverComponents.refresh();
    }
  }
}
```

MMDB.getColdFusionDsnList()

Availability

Dreamweaver UltraDev 4.

Description

This function gets the ColdFusion data source names (DSNs) from the site server, using the `getRDSUserName()` and `getRDSPassword()` functions.

Arguments

None.

Returns

An array that contains the ColdFusion DSNs that are defined on the server for the current site.

MMDB.getConnection()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

This function gets a named connection object.

Arguments

`name`

- The *`name`* argument is a string variable that specifies the name of the connection that you want to reference.

Returns

A reference to a named connection object. Connection objects contain the following properties:

Property	Description
`name`	Connection name
`type`	Indicates, if `useHTTP` is a value of `false`, which DLL to use for connecting to a database at runtime
`string`	Runtime ADO connection string or JDBC URL
`dsn`	ColdFusion DSN
`driver`	Runtime JDBC driver
`username`	Runtime user name
`password`	Runtime password
`useHTTP`	String that contains either a `true` or `false` value , specifying whether to use a remote driver (HTTP connection) at design time; otherwise, use a local driver (DLL)

Property	Description
`includePattern`	Regular expression used to find the file include statement on the page during Live Data and Preview In Browser
`variables`	Array of page variable names and their corresponding values used during Live Data and Preview In Browser
`catalog`	Used to restrict the metadata that appears (for more information, see "MMDB.getProcedures()" on page 94)
`schema`	Used to restrict the metadata that appears (for more information, see "MMDB.getProcedures()" on page 94)
`filename`	Filename of dialog box that was used to create the connection

Note: These properties are the standard ones that Dreamweaver implements. Developers can define their connection types and add new properties to this standard set or provide a different set of properties.

MMDB.getConnectionList()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of all the connection strings that are defined in the Connection Manager.

Arguments

None.

Returns

An array of strings where each string is the name of a connection as it appears in the Connection Manager.

Example

A call to `MMDB.getConnectionList()` can return the strings `["EmpDB", "Test", TestEmp"]`.

MMDB.getConnectionName()

Availability

Dreamweaver UltraDev 1.

Description

This function gets the connection name that corresponds to the specified connection string. This function is useful when you need to reselect a connection name in the user interface (UI) from data on the page.

If you have a connection string that references two drivers, you can specify the connection string and the driver that corresponds to the connection name that you want to return. For example, you can have two connections.

- Connection 1 has the following properties:
  ```
  ConnectionString="jdbc:inetdae:velcro-qa-5:1433?database=pubs"
  DriverName="com.inet.tds.TdsDriver"
  ```
- Connection 2 has the following properties:
  ```
  ConnectionString="jdbc:inetdae:velcro-qa-5:1433?database=pubs"
  DriverName="com.inet.tds.TdsDriver2"
  ```

The connection strings for Connection 1 and Connection 2 are the same. Connection 2 connects to a more recent version of the `TdsDriver` driver. You should pass the driver name to this function to fully qualify the connection name you want to return.

Arguments

`connString, {driverName}`

- The *`connString`* argument is the connection string that gets the connection name.
- The *`driverName`* argument, which is optional, further qualifies the *`connString`* argument.

Returns

A connection name string that corresponds to the connection string.

Example

The following code returns the string `"EmpDB"`:

```
var connectionName = MMDB.getConnectionName ¬
("dsn=EmpDB;uid=;pwd=");
```

MMDB.getConnectionString()

Availability

Dreamweaver UltraDev 1.

Description

This function gets the connection string that is associated with the named connection.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

A connection string that corresponds to the named connection.

Example

The code `var connectionString = MMDB.getConnectionString ("EmpDB")` returns different strings for an ADO or JDBC connection.

- For an ADO connection, the following string can return:
  ```
  "dsn=EmpDB;uid=;pwd=";
  ```
- For a JDBC connection, the following string can return:
  ```
  "jdbc:inetdae:192.168.64.49:1433?database=pubs&user=JoeUser&¬
  password=joesSecret"
  ```

MMDB.getDriverName()

Availability

Dreamweaver UltraDev 1.

Description

This function gets the driver name that is associated with the specified connection. Only a JDBC connection has a driver name.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

A string that contains the driver name.

Example

The statement `MMDB.getDriverName ("EmpDB");` might return the following string:

```
"jdbc/oracle/driver/JdbcOracle"
```

MMDB.getDriverUrlTemplateList() (deprecated)

Availability

Dreamweaver UltraDev 4, deprecated in Dreamweaver MX.

Note: For Dreamweaver UltraDev 4, the list of JDBC drivers are stored in the connections.xml file, which is located in the Configuration/Connections folder. Each driver has an associated URL template. This function returns the list of JDBC drivers.

For Dreamweaver MX (or later), these drivers and URL templates are hard-coded in the JDBC dialog boxes. In addition, this function is an empty function definition to eliminate undefined-function errors. The following example shows how a JDBC driver and URL template are hard-coded:

```
var DEFAULT_DRIVER = "COM.ibm.db2.jdbc.app.DB2Driver";
var DEFAULT_TEMPLATE = "jdbc:db2:[database name]";
```

Dreamweaver has a dialog box for each driver/URL template pair.

In summary, Dreamweaver UltraDev 4 developers need to add a new entry to the XML, and Dreamweaver MX (or later), developers need to implement a new dialog box.

Description

This function gets JDBC Drivers and respective URL templates.

Arguments

None.

Returns

An array that contains JDBC drivers that have been detected on the user's system and their respective URL templates, if they are specified. The array has an even number of elements that contain: `Driver1`, `UrlTemplate1`, `Driver2`, `UrlTemplate2`, and so on.

MMDB.getLocalDsnList()

Availability

Dreamweaver UltraDev 4.

Description

This function gets ODBC DSNs that are defined on the user's system.

Arguments

None.

Returns

An array that contains the ODBC DSNs that are defined on the user's system.

MMDB.getPassword()

Availability

Dreamweaver UltraDev 1.

Description

This function gets the password that is used for the specified connection.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

A password string that is associated with the connection name.

Example

The statement `MMDB.getPassword ("EmpDB");` might return `"joessecret"`.

MMDB.getRDSPassword()

Availability

Dreamweaver UltraDev 4.

Description

This function gets the Remote Development Services (RDS) password (for use with ColdFusion connections).

Arguments

None.

Returns

A string that contains the RDS password.

MMDB.getRDSUserName()

Availability

Dreamweaver UltraDev 4.

Description

This function gets the RDS user name (for use with ColdFusion connections).

Arguments

None.

Returns

A string that contains the RDS user name.

MMDB.getRemoteDsnList()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

This function gets the ODBC DSNs from the site server. The `getRDSUserName()` and `getRDSPassword()` functions are used when the server model of the current site is ColdFusion. This function provides an option for a developer to specify a URL parameter string to be appended to the Remote Connectivity URL that `MMDB.getRemoteDsnList()` generates. If the developer provides a parameter string, this function passes it to the HTTP connectivity scripts.

Arguments

`{urlParams}`

- The `urlParams` argument, which is optional, is a string that contains a list of `name=value` expressions, which are separated by ampersand (&) characters. You must not enclose values with quotes. Some characters, such as the space in the value `Hello World`, need to be encoded. The following example shows a valid sample argument that you can pass to `MMDB.getRemoteDsnList()`:

  ```
  a=1&b=Hello%20World
  ```

Returns

Returns an array that contains the ODBC DSNs that are defined on the server for the current site.

MMDB.getRuntimeConnectionType()

Availability

Dreamweaver UltraDev 1.

Description

This function returns the runtime connection type of the specified connection name.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

A string that corresponds to the connection type. This function can return one of the following values: `"ADO"`, `"ADODSN"`, `"JDBC"`, or `"CFDSN"`.

Example

The following code returns the string `"ADO"` for an ADO connection:

```
var connectionType = MMDB.getRuntimeConnectionType ("EmpDB")
```

MMDB.getUserName()

Availability

Dreamweaver UltraDev 1.

Description

This function returns a user name for the specified connection.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

A user name string that is associated with the connection name.

Example

The statement `MMDB.getUserName ("EmpDB");` might return `"amit"`.

MMDB.hasConnectionWithName()

Availability

Dreamweaver UltraDev 4.

Description

This function determines whether a connection of a given name exists.

Arguments

`name`

- The *`name`* argument is the connection name.

Returns

Returns a Boolean value: `true` indicates that a connection with the specified name exists; `false` otherwise.

MMDB.needToPromptForRdsInfo()

Availability

Dreamweaver MX.

Description

This function determines whether Dreamweaver should open the RDS Login Information dialog box.

Arguments

`bForce`

- The *`bForce`* argument is a Boolean value; `true` indicates that the user who has previously cancelled out of the RDS login dialog box still needs to be prompted for RDS login information.

Returns

A Boolean value: `true` indicates that the user needs to be prompted for RDS login information; `false` otherwise.

MMDB.needToRefreshColdFusionDsnList()

Availability

Dreamweaver MX.

Description

This function tells the Connection Manager to empty the cache and get the ColdFusion data source list from the application server the next time a user requests the list.

Arguments

None.

Returns

Nothing.

MMDB.popupConnection()

Availability

Dreamweaver MX.

Description

This function invokes a connection dialog box. This function has the following three signatures:

- If the argument list consists only of *`dialogFileName`* (a string), the `popupConnection()` function makes Dreamweaver open the Connection dialog box so you can define a new connection.
- If the argument list consists only of *`connRec`* (a connection reference), the `popupConnection()` function makes Dreamweaver launch the Connection dialog box in edit mode for editing the named connection. In this mode, the name text field is dimmed.
- If the argument list consists of *`connRec`* and the Boolean value *`bDuplicate`*, the `popupConnection()` function makes Dreamweaver open the Connection dialog box in duplicate mode. In this mode, the name text field is blanked out, and the remaining properties are copied to define a duplicate connection.

Arguments

`dialogFileName`

or

`connRec`

or

`connrec, bDuplicate`

- The *`dialogFileName`* argument is a string that contains the name of an HTML file that resides in the Configuration/Connections/*server-model* folder. This HTML file defines the dialog box that creates a connection. This file must implement three JavaScript API functions: `findConnection()`, `inspectConnection()`, and `applyConnection()`. Typically, you create a JavaScript file that implements these functions and then include that file in the HTML file. (For more information on creating a connection, see "The Database Connectivity API" on page 103.)

- The `connRec` argument is a reference to an existing Connection object.
- The `bDuplicate` argument is a Boolean value.

Returns

Nothing. The defined connection dialog box appears.

MMDB.setRDSPassword()

Availability

Dreamweaver UltraDev 4.

Description

This function sets the RDS password.

Arguments

`password`

- The `password` argument is a string that contains the RDS password.

Returns

Nothing.

MMDB.setRDSUserName()

Availability

Dreamweaver UltraDev 4.

Description

This function sets the RDS user name.

Arguments

`username`

- The `username` argument is a valid RDS user name.

Returns

Nothing.

MMDB.showColdFusionAdmin()

Availability

Dreamweaver MX.

Description

This function displays the ColdFusion Administrator dialog box.

Arguments

None.

Returns

Nothing. The ColdFusion Administrator dialog box appears.

MMDB.showConnectionMgrDialog()

Availability

Dreamweaver UltraDev 1.

Description

This function displays the Connection Manager dialog box.

Arguments

Nothing.

Returns

Nothing. The Connection Manager dialog box appears.

MMDB.showOdbcDialog()

Availability

Dreamweaver UltraDev 4 (Windows only).

Description

This function displays the System ODBC Administration dialog box or the ODBC Data Source Administrator dialog box.

Arguments

None.

Returns

Nothing. The System ODBC Administration dialog box or the ODBC Data Source Administrator dialog box appears.

MMDB.showRdsUserDialog()

Availability

Dreamweaver UltraDev 4.

Description

This function displays the RDS user name and password dialog box.

Arguments

`username, password`

- The *`username`* argument is the initial user name.
- The *`password`* argument is the initial password.

Returns

An object that contains the new values in the `username` and `password` properties. If either property is not defined, it indicates that the user cancelled the dialog box.

MMDB.showRestrictDialog()

Availability

Dreamweaver UltraDev 4.

Description

This function displays the Restrict dialog box.

Arguments

`catalog, schema`

- The *`catalog`* argument is the initial catalog value.
- The *`schema`* argument is the initial schema value.

Returns

An object that contains the new values in the `catalog` and `schema` properties. If either property is not defined, it indicates that the user cancelled the dialog box.

MMDB.testConnection()

Availability

Dreamweaver UltraDev 4.

Description

This function tests connection settings. It displays a modal dialog box that describes the results.

Arguments

`serverPropertiesArray`

This function expects a single argument, an array object that contains values from the following list, which are appropriate for the current server model. For properties that do not apply to the connection being tested, set them to empty ("").

- The *`type`* argument indicates, when *`useHTTP`* is a `false` value, which DLL to use for connecting to a database at design time to test connection settings.
- The *`string`* argument is the ADO connection string or JDBC URL.
- The *`dsn`* argument is the data source name.
- The *`driver`* argument is the JDBC driver.
- The *`username`* argument is the user name.
- The *`password`* argument is the password.
- The *`useHTTP`* argument is a Boolean value. A value of `true` specifies that Dreamweaver should use an HTTP connection at design time; otherwise, Dreamweaver uses a DLL.

Returns

A Boolean value: `true` if the connection test is successful; `false` otherwise.

Database access functions

Database access functions let you query a database. For the collection of functions that manage a database connection, see "Database connection functions" on page 78.

The following list describes some of the arguments that are common to the functions that are available:

- Most database access functions use a connection name as an argument. You can see a list of valid connection names in the Connection Manager, or you can use the `MMDB.getConnectionList()` function to get a list of all the connection names programmatically.
- Stored procedures often require parameters. There are two ways of specifying parameter values for database access functions. First, you can provide an array of parameter values (*`paramValuesArray`*). If you specify only parameter values, the values need to be in the sequence in which the stored procedure requires the parameters. Second, you specify parameter values to provide an array of parameter names (*`paramNameArray`*). You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure. If you provide parameter names, the values that you specify in *`paramValuesArray`* must be in the sequence of the parameter names that you specify in *`paramNameArray`*.

MMDB.getColumnAndTypeList()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of columns and their types from an executed SQL `SELECT` statement.

Arguments

`connName, statement`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`statement`* argument is the SQL `SELECT` statement to execute.

Returns

An array of strings that represents a list of columns (and their types) that match the `SELECT` statement, or an error if the SQL statement is invalid or the connection cannot be made.

Example

The code `var columnArray = MMDB.getColumnAndTypeList("EmpDB","Select * from Employees")` returns the following array of strings:

```
columnArray[0] = "EmpName"
columnArray[1] = "varchar"
columnArray[2] = "EmpFirstName"
columnArray[3] = "varchar"
columnArray[4] = "Age"
columnArray[5] = "integer"
```

MMDB.getColumnList()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of columns from an executed SQL `SELECT` statement.

Arguments

`connName, statement`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`statement`* argument is the SQL `SELECT` statement to execute.

Returns

An array of strings that represents a list of columns that match the `SELECT` statement, or an error if the SQL statement is invalid or the connection cannot be made.

Example

The code `var columnArray = MMDB.getColumnList("EmpDB","Select * from Employees")` returns the following array of strings:

```
columnArray[0] = "EmpName"
columnArray[1] = "EmpFirstName"
columnArray[2] = "Age"
```

MMDB.getColumns()

Availability

Dreamweaver MX, arguments updated in Dreamweaver MX 2004.

Description

This function returns an array of objects that describe the columns in the specified table.

Arguments

`connName, tableName`

- The *`connName`* argument is the connection name. This value identifies the connection containing the string that Dreamweaver should use to make a database connection to a live data source.
- The *`tableName`* argument is the table to query.

Returns

An array of objects, one object for each column. Each object defines the following three properties for the column with which it is associated.

Property Name	Description
`name`	Name of the column (for example, `price`)
`datatype`	Data type of the column (for example, `small money`)
`definedsize`	Defined size of the column (for example, `8`)
nullable	Indicates whether the column can contain `null` values

Example

The following example uses `MMDB.getColumns()` to set the tooltip text value:

```
var columnNameObjs = MMDB.getColumns(connName,tableName);
var databaseType   = MMDB.getDatabaseType(connName);

  for (i = 0; i < columnNameObjs.length; i++)
  {
    var columnObj = columnNameObjs[i];
    var columnName = columnObj.name;
    var typename = columnObj.datatype;
    if (dwscripts.isNumber(typename))
    {
      // it already is a num
      typename = dwscripts.getDBColumnTypeAsString(typename, databaseType);
    }

    var tooltiptext = typename;
  }
```

MMDB.getColumnsOfTable()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of all the columns in the specified table.

Arguments

`connName, tableName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`tableName`* argument is the name of a table in the database that is specified by the *`connName`* argument.

Returns

An array of strings where each string is the name of a column in the table.

Example

The statement `MMDB.getColumnsOfTable ("EmpDB","Employees");` returns the following strings:

```
["EmpID", "FirstName", "LastName"]
```

MMDB.getPrimaryKeys()

Availability

Dreamweaver MX.

Description

This function returns the column names that combine to form the primary key of the named table. A primary key serves as the unique identifier for a database row and consists of at least one column.

Arguments

`connName`, *`tableName`*

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`tableName`* argument is the name of the table for which you want to retrieve the set of columns that comprises the primary key of that table.

Returns

An array of strings. The array contains one string for each column that comprises the primary key.

Example

The following example returns the primary key for the specified table.

```
var connName    = componentRec.parent.parent.parent.name;
var tableName   = componentRec.name;
var primaryKeys = MMDB.getPrimaryKeys(connName,tableName);
```

MMDB.getProcedures()

Availability

Dreamweaver MX.

Description

This function returns an array of procedure objects that are associated with a named connection.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

An array of procedure objects where each procedure object has the following set of three properties:

Property Name	Description
`schema*`	Name of the schema that is associated with the object. This property identifies the user that is associated with the stored procedure in the SQL database that the `getProcedures()` function accesses. The database that this function accesses depends on the type of connection. • For ODBC connections, the ODBC data source defines the database. The DSN is specified by the `dsn` property in the connection object (*`connName`*) that you pass to the `getProcedures()` function. • For OLE DB connections, the connection string names the database.
`catalog`	Name of the catalog that is associated with the object (owner qualifier). The value of the `catalog` property is defined by an attribute of the OLE DB driver. This driver attribute defines a default `user.database` property to use when the OLE DB connection string does not specify a database.
`procedure`	Name of the procedure.

* Dreamweaver connects to and gets all the tables in the database whenever you modify a recordset. If the database has many tables, Dreamweaver might take a long time to retrieve them on certain systems. If your database contains a schema or catalog, you can use the schema or catalog to restrict the number of database items Dreamweaver gets at design time. You must first create a schema or catalog in your database application before you can apply it in Dreamweaver. Consult your database documentation or your system administrator.

Example

The following code gets a list of procedures:

```
var procObjects   = MMDB.getProcedures(connectionName);
for (i = 0; i < procObjects.length; i++)
{
  var thisProcedure = procObjects[i]
  thisSchema =  Trim(thisProcedure.schema)
  if (thisSchema.length == 0)
  {
  thisSchema = Trim(thisProcedure.catalog)
  }
  if (thisSchema.length > 0)
  {
  thisSchema += "."
  }

  var procName = String(thisSchema + thisProcedure.procedure);
 }
```

MMDB.getSPColumnList()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of result set columns that are generated by a call to the specified stored procedure.

Arguments

connName, statement, paramValuesArray

- The *connName* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *statement* argument is the name of the stored procedure that returns the result set when it executes.
- The *paramValuesArray* argument is an array that contains a list of design-time parameter test values. Specify the parameter values in the order in which the stored procedure expects them. You can use the `MMDB.getSPParamsAsString()` function to get the parameters for the stored procedure.

Returns

An array of strings that represents the list of columns. This function returns an error if the SQL statement or the connection string is invalid.

Example

The following code can return a list of result set columns that are generated from the executed stored procedure, `getNewEmployeesMakingAtLeast`:

```
var paramValueArray = new Array("2/1/2000", "50000")
var columnArray = MMDB.getSPColumnList("EmpDB", ¬
"getNewEmployeesMakingAtLeast", paramValueArray)
```

The following values return:

```
columnArray[0] = "EmpID", columnArray[1] = "LastName", ¬
columnArray[2] ="startDate", columnArray[3] = "salary"
```

MMDB.getSPColumnListNamedParams()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of result set columns that are generated by a call to the specified stored procedure.

Arguments

connName, statement, paramNameArray, paramValuesArray

- The *connName* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *statement* argument is the name of the stored procedure that returns the result set when it executes.
- The *paramNameArray* argument is an array that contains a list of parameter names. You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.
- The *paramValuesArray* argument is an array that contains a list of design-time parameter test values. You can specify if the procedure requires parameters when it executes. If you have provided parameter names in *paramNameArray*, specify the parameter values in the same order that their corresponding parameter names appear in *paramNameArray*. If you did not provide *paramNameArray*, specify the values in the order in which the stored procedure expects them.

Returns

An array of strings that represents the list of columns. This function returns an error if the SQL statement or the connection string is invalid.

Example

The following code can return a list of result set columns that are generated from the executed stored procedure, `getNewEmployeesMakingAtLeast`:

```
var paramNameArray = new Array("startDate", "salary")
var paramValueArray = new Array("2/1/2000", "50000")
var columnArray = MMDB.getSPColumnListNamedParams("EmpDB", ¬
"getNewEmployeesMakingAtLeast", paramNameArray, paramValueArray)
```

The following values return:

```
columnArray[0] = "EmpID", columnArray[1] = "LastName",¬
columnArray[2] ="startDate", columnArray[3] = "salary"
```

MMDB.getSPParameters()

Availability

Dreamweaver MX.

Description

This function returns an array of parameter objects for a named procedure.

Arguments

connName, *procName*

- The *connName* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *procName* argument is the name of the procedure.

Returns

An array of parameter objects, each specifying the following set of properties:

Property name	Description
`name`	Name of the parameter (for example, `@@lolimit`)
`datatype`	Datatype of the parameter (for example, `smallmoney`)
`direction`	Direction of the parameter: `1`-The parameter is used for input only. `2`-The parameter is used for output only. In this case, you pass the parameter by reference and the method places a value in it. You can use the value after the method returns. `3`- The parameter is used for both input and output. `4`- The parameter holds a return value.

Example

The following example retrieves the parameter objects for the specified procedure and creates a tooltip for each object using its properties.

```
var paramNameObjs = MMDB.getSPParameters(connName,procName);
for (i = 0; i < paramNameObjs.length; i++)
{
  var paramObj = paramNameObjs[i];
  var tooltiptext = paramObj.datatype;
  tooltiptext+=" ";
  tooltiptext+=GetDirString(paramObj.directiontype);
}
```

MMDB.getSPParamsAsString()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a comma-delimited string that contains the list of parameters that the stored procedure takes.

Arguments

`connName, procName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`procName`* argument is the name of the stored procedure.

Returns

A comma-delimited string that contains the list of parameters that the stored procedure requires. The parameters' names, direction, and data type are included, separated by semicolons (;).

Example

The code `MMDB.getSPParamsAsString ("EmpDB","getNewEmployeesMakingAtLeast")` can return a string of form name `startDate;direction:in;datatype:date, salary;direction:in;datatype:integer`.

In this example, the stored procedure, `getNewEmployeesMakingAtLeast`, has two parameters: `startDate` and `Salary`. For `startDate`, the direction is `in` and the data type is `date`. For `salary`, the direction is `in` and the data type is `date`.

MMDB.getTables()

Availability

Dreamweaver UltraDev 1.

Description

This function gets a list of all the tables that are defined for the specified database. Each table object has three properties: `table`, `schema`, and `catalog`.

Arguments

`connName`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

An array of objects where each object has three properties: `table`, `schema`, and `catalog`. `Table` is the name of the table. `Schema` is the name of the schema that contains the table. `Catalog` is the catalog that contains the table.

Example

The statement `MMDB.getTables ("EmpDB");` might produce an array of two objects. The first object's properties might be similar to the following example:

```
object1[table:"Employees", schema:"personnel", catalog:"syscat"]
```

The second object's properties might be similar to the following example:

```
object2[table:"Departments", schema:"demo", catalog:"syscat2"]
```

MMDB.getViews()

Availability

Dreamweaver UltraDev 4.

Description

This function gets a list of all the views that are defined for the specified database. Each view object has `catalog`, `schema`, and `view` properties.

Arguments

connName

- The *connName* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.

Returns

An array of view objects where each object has three properties: `catalog`, `schema`, and `view`. Use `catalog` or `schema` to restrict or filter the number of views that pertain to an individual schema name or catalog name that is defined as part of the connection information.

Example

The following example returns the views for a given connection value, `CONN_LIST.getValue()`:

```
var viewObjects = MMDB.getViews(CONN_LIST.getValue())
for (i = 0; i < viewObjects.length; i++)
{
  thisView = viewObjects[i]
  thisSchema = Trim(thisView.schema)
  if (thisSchema.length == 0)
  {
    thisSchema = Trim(thisView.catalog)
  }
  if (thisSchema.length > 0)
  {
    thisSchema += "."
  }
  views.push(String(thisSchema + thisView.view))
}
```

MMDB.showResultset()

Availability

Dreamweaver UltraDev 1.

Description

This function displays a dialog box that contains the results of executing the specified SQL statement.The dialog box displays a tabular grid in which the header provides column information that describes the result set. If the connection string or the SQL statement is invalid, an error appears. This function validates the SQL statement.

Arguments

connName, SQLstatement

- The *connName* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *SQLstatement* argument is the SQL `SELECT` statement.

Returns

Nothing. This function returns an error if the SQL statement or the connection string is invalid.

Example

The following code displays the results of the executed SQL statement:

```
MMDB.showResultset("EmpDB","Select EmpName,EmpFirstName,Age ¬
from Employees")
```

MMDB.showSPResultset()

Availability

Dreamweaver UltraDev 1.

Description

This function displays a dialog box that contains the results of executing the specified stored procedure. The dialog box displays a tabular grid in which the header provides column information that describes the result set. If the connection string or the stored procedure is invalid, an error appears. This function validates the stored procedure.

Arguments

`connName, procName, paramValuesArray`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`procName`* argument is the name of the stored procedure to execute.
- The *`paramValuesArrayargument`* is an array that contains a list of design-time parameter test values. Specify the parameter values in the order in which the stored procedure expects them. You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.

Returns

This function returns an error if the SQL statement or the connection string is invalid; otherwise, it returns nothing.

Example

The following code displays the results of the executed stored procedure:

```
var paramValueArray = new Array("2/1/2000", "50000")
MMDB.showSPResultset("EmpDB", "getNewEmployeesMakingAtLeast", ¬
paramValueArray)
```

MMDB.showSPResultsetNamedParams()

Availability

Dreamweaver UltraDev 1.

Description

This function displays a dialog box that contains the result set of the specified stored procedure. The dialog box displays a tabular grid in which the header provides column information that describes the result set. If the connection string or the stored procedure is invalid, an error appears. This function validates the stored procedure. This function differs from the `MMDB.showSPResultset()` function because you can specify the parameter values by name instead of the order in which the stored procedure expects them.

Arguments

`connName, procName, paramNameArray, paramValuesArray`

- The *`connName`* argument is a connection name that is specified in the Connection Manager. It identifies the connection string that Dreamweaver should use to make a database connection to a live data source.
- The *`procName`* argument is the name of the stored procedure that returns the result set when it executes.
- The *`paramNameArray`* argument is an array that contains a list of parameter names. You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.
- The *`paramValuesArray`* argument is an array that contains a list of design-time parameter test values.

Returns

This function returns an error if the SQL statement or the connection string is invalid; otherwise, it returns nothing.

Example

The following code displays the results of the executed stored procedure:

```
var paramNameArray = new Array("startDate", "salary")
var paramValueArray = new Array("2/1/2000", "50000")
MMDB.showSPResultsetNamedParams("EmpDB","getNewEmployees¬
MakingAtLeast", paramNameArray, paramValueArray)
```

CHAPTER 8
The Database Connectivity API

As a developer, you can create new connection types and corresponding dialog boxes for new or existing server models for Macromedia Dreamweaver MX 2004. Then, when a user sets up a site to start building pages, he or she creates a new connection object after selecting the particular type of connection that you created.

The user can select your new connection type in the following ways:

- On the Application panel, the user can click the Plus (+) button and select Recordset. In the Recordset dialog box, the user can expand the Connection list box.
- On the Database tab of the Databases panel, the user can click the Plus (+) button and select Data Source Name.

How to develop a new connection type

The following steps outline the process for creating a new connection type:

1 Create the layout for the connection dialog box.

   Create an HTML file that lays out the user interface (UI) for your connection dialog box. Name this file using the name of the connection (for example, myConnection.htm). For information about creating a dialog box, see *Getting Started with Dreamweaver*.

   Make sure this HTML file includes a reference to the JavaScript implementation file that you define in Step 2, "Create a JavaScript file that implements at least the following elements:" on page 104, as shown in the following example:

```
<head>
  <script SRC="../myConnectionImpl.js"></script>
</head>
```

   Store this HTML file, which defines your connection dialog box, in the Configuration/Connections/*server-model*/*platform* folder (where the *platform* is either Windows or Macintosh).

   For example, the default ADO connection dialog box for an ASP JavaScript document on a Windows platform is stored in the ASP_Js/Win folder and is named Connection_ado_conn_string.htm.

   Note: At runtime, Macromedia Dreamweaver dynamically builds the list of connection types that are available to the user from the collection of dialog boxes that are in the ASP_Js/Win folder.

The Configuration/ServerModels folder has HTML files that define each server model. Inside each HTML file is the `getServerModelFolderName()` function, which returns the name of the folder that is associated with the server model. The following example shows the function for the ASP JavaScript document type:

```
function getServerModelFolderName()
{
  return "ASP_JS";
}
```

You can also look at the MMDocumentTypes.xml file, which is located in the Configuration/DocumentTypes folder, to determine the mapping between server models and document types.

2 Create a JavaScript file that implements at least the following elements:

Element	Description	Examples
A set of variables	Each variable defines a specific connection property	Type of connection, data source name, and so on
A set of buttons	Each button appears in the connection dialog box	Test, Help, and so on (OK and Cancel are automatically included)
Connectivity functions	Together, these functions define the Connectivity API	`findConnection()` `applyConnection()` `inspectConnection()`

You can select any name for this implementation file, but it must have a .js extension (for example, myConnectionImpl.js). You can store this implementation file on either your local or a remote computer. You might want to store your implementation file in the appropriate subfolder within the Configuration/Connections folder.

Note: The HTML file that you defined in Step 1, "Create the layout for the connection dialog box." on page 103 must include this connection type implementation file.

Unless you need to define connection parameters other than the ones provided in the standard connection_includefile.edml file, these two steps are the minimum to create a new connection dialog box.

Note: The title of the dialog box that the user sees is in the `title` tag, which is specified in the HTML document.

The functions listed in the next section let you create a connection dialog box. Along with implementing the calls for generating include files for the user, you can register your connectivity type within the server model section of the connection XML file.

For information about the Database Connectivity API that is associated with creating a new connection, see "Database connection functions" on page 78.

The Connection API

To create a new type of connection, including the dialog box with which users interact, you must implement the following three functions: `findConnection()`, `inspectConnection()`, and `applyConnection()`. You write these three functions and include them in the JavaScript implementation file that is associated with your new connection type (see Step 2 "Create a JavaScript file that implements at least the following elements:" on page 104).

The `applyConnection()` function returns an HTML source within an include file. You can see examples of the HTML source in "The generated include file" on page 108. The `findConnection()` function takes the HTML source and extracts its properties. You can implement `findConnection()` to use the search patterns in XML files to extract the information that returns from `applyConnection()`. For an example of such an implementation, see the following two JavaScript files:

- `connection_ado_conn_string.js` is located in Configuration/Connections/ASP_Js folder.
- `connection_common.js` is located in Configuration/Connections/Shared folder.

When the user opens a site, Dreamweaver goes through each file in the Connections folder, opens it, and passes the contents to `findConnection()`. If the contents of a file match the criteria for a valid connection, `findConnection()` returns a connection object. Dreamweaver then lists all the connection objects in the Database Explorer panel.

When the user opens a connection dialog box and selects to create a new connection or duplicate or edit an existing connection, Dreamweaver calls the `inspectConnection()` function and passes back the same connection object that `findConnection()` created. This process lets Dreamweaver populate the dialog box with the connection information.

When the user clicks OK in a connection dialog box, Dreamweaver calls the `applyConnection()` function to build the HTML, which is placed in the connection include file that is located in the Configuration/Connections folder. The `applyConnection()` function returns an empty string that indicates there is an error in one of the fields and the dialog box should not be closed. The include file has the default file extension type for the current server model.

When the user adds to the page a server behavior that uses the connection, such as a recordset or a stored procedure, Dreamweaver adds a statement to the page that includes the connection include file.

findConnection()

Availability

Dreamweaver UltraDev 4.

Description

Dreamweaver calls this function to detect a connection in the specified HTML source and to parse the connection parameters. If the contents of this source file match the criteria for a valid connection, `findConnection()` returns a connection object; otherwise, this function returns a `null` value.

Argument

`htmlSource`

The *`htmlSource`* argument is the HTML source for a connection.

Returns

A connection object that provides values for a particular combination of the properties that are listed in the following table. The properties for which this function returns a value depend on the document type.

Property	Description
`name`	Name of the connection
`type`	If `useHTTP` is `false`, indicates which DLL to use for connecting to database at runtime
`string`	Runtime connection string. For ADO, it is a string of connection parameters; for JDBC, it is a connection URL
`dsn`	Data source name used for ODBC or Cold Fusion runtime connections
`driver`	Name of a JDBC driver used at runtime
`username`	Name of the user for the runtime connection
`password`	Password used for the runtime connection
`designtimeString`	Design-time connection string (see `string`)
`designtimeDsn`	Design-time data source name (see `dsn`)
`designtimeDriver`	Name of a JDBC driver used at design time
`designtimeUsername`	Name of the user used for the design-time connection
`designtimePassword`	Password used for the design-time connection
`designtimeType`	Design-time connection type
`usesDesigntimeInfo`	When `false`, Dreamweaver uses runtime properties at design time; otherwise, Dreamweaver uses design-time properties
`useHTTP`	String containing either `true` or `false: true` specifies to use HTTP connection at design time; `false` specifies to use DLL
`includePattern`	Regular expression used to find the file include statement on the page during Live Data and Preview In Browser
`variables`	Object with a property for each page variable that is set to its corresponding value. This object is used during Live Data and Preview In Browser
`catalog`	String containing a database identifier that restricts the amount of metadata that appears
`schema`	String containing a database identifier that restricts the amount of metadata that appears
`filename`	Name of the dialog box used to create the connection

If a connection is not found in htmlSource, a `null` value returns.

Note: Developers can add custom properties (for example, metadata) to the HTML source, which `applyConnection()` returns along with the standard properties.

inspectConnection()

Availability

Dreamweaver UltraDev 4.

Description

Dreamweaver calls this function to initialize the dialog box data for defining a connection when the user edits an existing connection. This process lets Dreamweaver populate the dialog box with the appropriate connection information.

Argument

`parameters`

The *`parameters`* argument is the same object that the `findConnection()` function returns.

Returns

Nothing.

applyConnection()

Availability

Dreamweaver UltraDev 4.

Description

Dreamweaver calls this function when the user clicks OK in the connection dialog box. The `applyConnection()` function generates the HTML source for a connection. Dreamweaver writes the HTML to the Configuration/Connections/*connection-name.ext* include file, where *connection-name* is the name of your connection (see “Create the layout for the connection dialog box.” on page 103), and .ext is the default extension that is associated with the server model.

Arguments

None.

Returns

The HTML source for a connection. Dreamweaver also closes the connection dialog box. If a field validation error occurs, `applyConnection()` displays an error message and returns an empty string to indicate that the dialog box should remain open.

The generated include file

The include file that `applyConnection()` generates declares all the properties of a connection.The filename for the include file is the connection name and has the file extension that is defined for the server model associated with the current site.

Note: Connections are shared, so set the `allowMultiple` value to `false`. This ensures that the connection file is included in the document only once and that the server script remains in the page if any other server behaviors use it.

The following sections illustrate some sample include files that `applyConnection()` generates for various default server models.

Note: To create a new connection include file format, you need to define a new EDML mapping file, which should be similar to connection_includefile.edml, as shown in "The definition file for your connection type" on page 109.

ASP JavaScript

The ASP and JavaScript include file should be named MyConnection1.asp, where MyConnection1 is the name of the connection. The following sample is an include file for an ADO connection string:

```
<%
  // Filename="Connection_ado_conn_string.htm"
  // Type="ADO"
  // HTTP="true"
  // Catalog=""
  // Schema=""
  var MM_MyConnection1_STRING = "dsn=pubs";
%>
```

The server behavior file includes this connection by using the relative file include statement, as shown in the following example:

```
<!--#include file="../Connections/MyConnection1.asp"-->
```

ColdFusion

When you use UltraDev 4 ColdFusion, Dreamweaver relies on a ColdFusion include file to get a list of data sources.

Note: For regular Dreamweaver ColdFusion, Dreamweaver ignores any include files and, instead, makes use of RDS to retrieve the list of data sources from ColdFusion.

The UltraDev 4 ColdFusion include file should be named MyConnection1.cfm, where MyConnection1 is the name of your connection. The following example shows the include file for a ColdFusion connection to a product table:

```
<!-- FileName="Connection_cf_dsn.htm" "dsn=products" -->
<!-- Type="ADO" -->
<!-- Catalog="" -->
<!-- Schema="" -->
<!-- HTTP="false" -->
<CFSET MM_MyConnection1_DSN       = "products">
<CFSET MM_MyConnection1_USERNAME  = "">
<CFSET MM_Product_USERNAME        = "">
<CFSET MM_MyConnection1_PASSWORD  = "">
```

The server behavior file includes this connection by using the `cfinclude` statement, as shown in the following example:

```
<cfinclude template="Connections/MyConnection1.cfm">
```

JSP

The JSP include file should be named MyConnection1.jsp, where MyConnection1 is the name of your connection. The following example is the include file for a JDBC connection to a database:

```
<%
  // Filename="Connection_jdbc_conn1.htm"
  // Type="JDBC"
  // HTTP="false"
  // Catalog=""
  // Schema=""
  String MM_MyConnection1_DRIVER    = "com.inet.tds.TdsDriver";
  String MM_MyConnection1_USERNAME  = "testadmin";
  String MM_MyConnection1_PASSWORD  = "velcro";
  String MM_MyConnection1_URL       = "jdbc:server:test-3:1433?database=pubs";
%>
```

The server behavior file includes this connection by using the relative file include statement, as shown in the following example:

```
<%@ include file="Connections/MyConnection1.jsp" %>
```

The definition file for your connection type

For each server model, there is a connection_includefile.edml file that defines the connection type and maps the properties that are defined in the include file to elements in the Dreamweaver interface.

Dreamweaver provides seven default definition files, one for each of the predefined server models, as listed in the following table.

Server model	Subfolder within the Configuration/Connections folder
ASP JavaScript	ASP_Js
ASP.NET CSharp	ASP.NET_Csharp
ASP.NET VBScript	ASP.NET_VB
ASP VBScript	ASP_Vbs
ColdFusion	ColdFusion
JavaServer Page	JSP
PHP MySql	PHP_MySql

Dreamweaver uses the `quickSearch` and `searchPattern` parameters to recognize connection blocks and the `insertText` parameter to create connection blocks. For more information on EDML tags and attributes, and regular expression search patterns, see “Server Behaviors” in *Extending Dreamweaver.*

Note: If you change the format of your include file or define an include file for a new server model, you need to map the connection parameters with the Dreamweaver UI, Live Data, and Preview In Browser. The following sample EDML file, which is associated with the default ASP JS server model, maps all connection page variables with their respective live values before sending the page to the server. For more information on EDML and regular expression search patterns, see "Server Behaviors" in *Extending Dreamweaver*.

```
<participant name="connection_includefile" version="5.0">
  <quickSearch>
    <![CDATA[// HTTP=]]></quickSearch>
  <insertText location="">
<![CDATA[<%
// FileName="@@filename@@"
// Type="@@type@@" @@designtimeString@@
// DesigntimeType="@@designtimeType@@"
// HTTP="@@http@@"
// Catalog="@@catalog@@"
// Schema="@@schema@@"
var MM_@@cname@@_STRING = @@string@@
%>
]]>
  </insertText>
  <searchPatterns whereToSearch="directive">
    <searchPattern paramNames="filename">
      <![CDATA[/\/\/\s*FileName="([^"]*)"/]]></searchPattern>
    <searchPattern paramNames="type,designtimeString">
      <![CDATA[/\/\/\s+Type="(\w*)"([^\r\n]*)/]]></searchPattern>
    <searchPattern paramNames="designtimeType" isOptional="true">
      <![CDATA[/\/\/\s*DesigntimeType="(\w*)"/]]></searchPattern>
    <searchPattern paramNames="http">
      <![CDATA[/\/\/\s*HTTP="(\w+)"/]]></searchPattern>
    <searchPattern paramNames="catalog">
      <![CDATA[/\/\/\s*Catalog="(\w*)"/]]></searchPattern>
    <searchPattern paramNames="schema">
      <![CDATA[/\/\/\s*Schema="(\w*)"/]]></searchPattern>
    <searchPattern paramNames="cname,string">
      <![CDATA[/var\s+MM_(\w*)_STRING\s*=\s*([^\r\n]+)/]]></searchPattern>
  </searchPatterns>
</participant>
```

Tokens in an EDML file—such as `@@filename@@` in this example—map values in the include file to properties of a connection object. You set the properties of connection objects in the JavaScript implementation file.

All the default connection dialog boxes that come with Dreamweaver use the connection_includefile.edml mapping file. To let Dreamweaver find this file, its name is set in the JavaScript implementation file, as shown in the following example:

```
var PARTICIPANT_FILE = "connection_includefile";
```

When you create a custom connection type, you can use any mapping file in your custom dialog boxes. If you create a mapping file, you can use a name other than connection_includefile for your EDML file. If you use a different name, you need to use this name in your JavaScript implementation file when you specify the value that is assigned to the `PARTICIPANT_FILE` variable, as shown in the following example:

```
var PARTICIPANT_FILE = "myConnection_mappingfile";
```

CHAPTER 9
The JavaBeans API

This chapter explains the APIs for JavaBeans; the `MMJB*()` functions are JavaScript hooks that invoke Java introspection calls for JavaBeans support. These functions get class names, methods, properties, and events from the JavaBeans, which can appear in the Dreamweaver user interface (UI). To use these JavaScript functions and let Macromedia Dreamweaver MX 2004 access your JavaBeans, the JavaBeans must reside in the Configuration/Classes folder.

Note: The function arguments described in this chapter sometimes contain an argument called `packageName.className`, which is intended to represent a single value.

The JavaBeans API

The following functions are methods of the `MMJB` object.

MMJB.getClasses()

Availability

Dreamweaver UltraDev 4.

Description

This function reads all the JavaBeans class names from the Configuration/Classes folder.

Arguments

None.

Returns

A string array of class names that are located in Configuration/Classes folder; an error returns an empty array.

MMJB.getClassesFromPackage()

Availability

Dreamweaver UltraDev 4.

Description

This function reads all the JavaBeans classes from the package.

Arguments

`packageName.pathName`

- The *`packageName.pathName`* argument is the path to the package. It must be a Java JAR or ZIP Java archive (for example, `C:/jdbcdrivers/Una2000_Enterprise.zip`).

Returns

A string array of class names inside the particular JAR or ZIP Java archive; an error returns an empty array.

MMJB.getErrorMessage()

Availability

Dreamweaver UltraDev 4.

Description

This function gets the last error message from Dreamweaver that occurred while using the MMJB interface.

Arguments

None.

Returns

A string of the Dreamweaver message from the last error.

MMJB.getEvents()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Introspects the JavaBeans class and returns its events.

Arguments

`packageName.className`, *`{packagePath}`*

- The *`packageName.className`* argument is the name of the class. The class must reside in a JAR or ZIP Java archive. If *`packagePath`* is omitted, the archive must reside in your system `classpath` or be a class file that is installed in the Configuration/Classes folder.
- The *`packagePath`* argument is an optional string that points to the location of the JAR or ZIP Java archive that contains *`className`*.

Returns

A string array of the events associated with *`className`*; an error returns an empty array.

MMJB.getIndexedProperties()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Introspects the JavaBeans class and returns its indexed properties, which are design patterns that behave the same way as collections.

Arguments

`packageName.className`, {*`packagePath`*}

- The *`packageName.className`* argument is the name of the class. The class must reside in a JAR or ZIP Java archive. If *`packagePath`* is omitted, the archive must reside in your system `classpath` or be a class file that is installed in the Configuration/Classes folder.
- The *`packagePath`* argument, which is optional, is a string that points to the location of the JAR or ZIP Java archive that contains *`className`*.

Returns

A string array of the indexed properties associated with *`className`*; an error returns an empty array.

MMJB.getMethods()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Introspects the JavaBeans class and returns its methods.

Arguments

`packageName.className`, {*`packagePath`*}

- The *`packageName.className`* argument is the name of the class. The class must reside in a JAR or ZIP Java archive. If *`packagePath`* is omitted, the archive must reside in your system `classpath` or be a class file that is installed in the Configuration/Classes folder.
- The *`packagePath`* argument is an optional string that points to the location of the JAR or ZIP Java archive that contains *`className`*.

Returns

A string array of the methods associated with *`className`*; an error returns an empty array.

MMJB.getProperties()

Availability

Dreamweaver UltraDev 4, enhanced in Dreamweaver MX.

Description

Introspects the JavaBeans class and returns its properties.

Arguments

`packageName.className`, {*`packagePath`*}

- The *`packageName.className`* argument is the name of the class. The class must reside in a JAR or ZIP Java archive. If *`packagePath`* is omitted, the archive must reside in your system `classpath` or be a class file that is installed in the Configuration/Classes folder.
- The *`packagePath`* argument is an optional string that points to the location of the JAR or ZIP Java archive that contains *`className`*.

Returns

A string array of the properties associated with *`className`*; an error returns an empty array.

MMJB.getReadProperties()

Availability

Dreamweaver MX.

Description

Gets read-only properties for JavaBeans that support get accessor calls.

Arguments

`packageName.className`, {*`packagePath`*}

- The *`packageName.className`* argument is the name of the class. The class must reside in a JAR or ZIP Java archive. If *`packagePath`* is omitted, the archive must reside in your system `classpath` or be a class file that is installed in the Configuration/Classes folder.
- The *`packagePath`* argument, which is optional, is a string that points to the location of the JAR or ZIP Java archive that contains *`className`*.

Returns

A string array of read-only properties associated with *`className`*; an error returns an empty array.

MMJB.getWriteProperties()

Availability

Dreamweaver MX.

Description

Gets write-only properties for JavaBeans that support set method calls.

Arguments

`packageName.className,` {*`packagePath`*}

- The *`packageName.className`* argument is the name of the class. The class must reside in a JAR or ZIP Java archive. If *`packagePath`* is omitted, the archive must reside in your system `classpath` or be a class file that is installed in the Configuration/Classes folder.
- The *`packagePath`* argument, which is optional, is a string that points to the location of the JAR or ZIP Java archive that contains *`className`*.

Returns

A string array of write-only properties associated with *`className`*; an error returns an empty array.

CHAPTER 10
The Source Control Integration API

The Source Control Integration API lets you write shared libraries to extend the Macromedia Dreamweaver MX 2004 Check In/Check Out feature using source control systems (such as Sourcesafe or CVS).

Your libraries must support a minimum set of API functions for Dreamweaver to integrate with a source control system. And, your libraries must reside in the Program Files/Common Files/Macromedia/2004/Source Control folder.

When Dreamweaver starts, it loads each library. Dreamweaver determines which features the library supports by calling `GetProcAddress()` for each API function. If an address does not exist, Dreamweaver assumes the library does not support the API. If the address exists, Dreamweaver uses the library's version of the function to support the functionality. When a Dreamweaver user defines or edits a site and then selects the Web Server SCS tab, the choices that correspond to the DLLs that loaded from the Program Files/Common Files/Macromedia/2004/Source Control folder appear (in addition to the standard items) on the tab.

To create a Site > Source Control menu to which you can add custom items, add the following code in the Site menu in the menus.xml file:

```
<menu name="Source Control" id="DWMenu_MainSite_Site_Source¬
Control"><menuitem dynamic name="None"file="Menus/MM/¬
File_SCSItems.htm" id="DWMenu_MainSite_Site_NewFeatures_¬
Default" />
</menu>
```

How source control integration with Dreamweaver works

When a Dreamweaver user selects server connection, file transfer, or Design Notes features, Dreamweaver calls the DLL's version of the corresponding API function (`Connect()`, `Disconnect()`, `Get()`, `Put()`, `Checkin()`, `Checkout()`, `Undocheckout()`, and `Synchronize()`). The DLL handles the request, including displaying dialog boxes that gather information or letting the user interact with the DLL. The DLL also displays information or error messages.

The source control system can optionally support Design Notes and Check In/Check Out. The Dreamweaver user enables Design Notes in source control systems by selecting the Design Notes tab in the Edit Sites dialog box and checking the box that enables the feature; this process is same to enable Design Notes with FTP and LAN. If the source control system does not support Design Notes and the user wants to use this feature, Dreamweaver transports Design Note (MNO) files to maintain the Design Notes (as it does with FTP and LAN).

Check In/Check Out is treated differently than the Design Notes feature; if the source control system supports it, the user cannot override its use from the Design Notes dialog box. If the user tries to override the source control system, an error message appears.

Adding source control system functionality

You can add source control system functionality to Dreamweaver by writing a `GetNewFeatures` handler that returns a set of menu items and corresponding C functions. For example, if you write a Sourcesafe library and want to let Dreamweaver users see the history of a file, you can write a `GetNewFeatures` handler that returns the History menu item and the C function name of `history`. Then, in Windows, when the user right-clicks a file, the History menu item is one of the items on the menu. If a user selects the History menu item, Dreamweaver calls the corresponding function, passing the selected file(s) to the DLL. The DLL displays the History dialog box so the user can interact with it in the same way as Sourcesafe.

The Source Control Integration API required functions

The Source Control Integration API has required and optional functions. The functions listed in this section are required.

bool SCS_GetAgentInfo()

Description

This function asks the DLL to return its name and description, which appear in the Edit Sites dialog box. The name appears in the Server Access pop-up menu (for example, sourcesafe, webdav, perforce) and the description below the pop-up menu.

Arguments

`char name[32]`, *`char version[32]`*, *`char description[256]`*, *`const char *dwAppVersion`*

- The *`name`* argument is the name of the source control system. The name appears in the combo box for selecting a source control system on the Source Control tab in the Edit Sites dialog box. The name can be a maximum of 32 characters.
- The *`version`* argument is a string that indicates the version of the DLL. The version appears on the Source Control tab in the Edit Sites dialog box. The version can be a maximum of 32 characters.
- The *`description`* argument is a string that indicates the description of the source control system. The description appears on the Source Control tab in the Edit Sites dialog box. The description can be a maximum of 256 characters.
- The *`dwAppVersion`* argument is a string that indicates the version of Dreamweaver that is calling the DLL. The DLL can use this string to determine the version and language of Dreamweaver.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Connect()

Description

This function connects the user to the source control system. If the DLL does not have log-in information, the DLL must display a dialog box to prompt the user for the information and must store the data for later use.

Arguments

`void **connectionData`, *`const char siteName[64]`*

- The *`connectionData`* argument is a handle to the data that the agent wants Dreamweaver to pass to it when calling other API functions.
- The *`siteName`* argument is a string that points to the name of the site. The site name can be a maximum of 64 characters.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Disconnect()

Description

This function disconnects the user from the source control system.

Arguments

`void *connectionData`

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_IsConnected()

Description

This function determines the state of the connection.

Arguments

`void *connectionData`

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

A Boolean value: `true` if successful; `false` otherwise.

int SCS_GetRootFolderLength()

Description

This function returns the length of the name of the root folder.

Arguments

`void *connectionData`

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

An integer that indicates the length of the name of the root folder. If the function returns `< 0`, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

bool SCS_GetRootFolder()

Description

This function returns the name of the root folder.

Arguments

`void *connectionData`, *`char remotePath[]`*, *`const int folderLen`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* is a buffer where the full remote path of the root folder is stored.
- The *`folderLen`* argument is an integer that indicates the length of *`remotePath`*. This is the value that *`GetRootFolderLength`* returns.

Returns

A Boolean value: `true` if successful; `false` otherwise.

int SCS_GetFolderListLength()

Description

This function returns the number of items in the passed-in folder.

Arguments

`void *connectionData`, *`const char *remotePath`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is the full path and name of the remote folder that the DLL checks for the number of items.

Returns

An integer that indicates the number of items in the current folder. If the function returns `< 0`, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

bool SCS_GetFolderList()

Description

This function returns a list of files and folders in the passed-in folder, including pertinent information such as modified date, size, and whether the item is a folder or file.

Arguments

`void *connectionData`, *`const char *remotePath`*, *`itemInfo itemList`*[ ], *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is the path of the remote folder that the DLL checks for the number of items.
- The *`itemList`* argument is a preallocated list of `itemInfo` structures:

name	**char[256]**	**Name of file or folder**
`isFolder`	bool	`true` if folder; `false` if file
`month`	int	Month component of modification date 1-12
`day`	int	Day component of modification date 1-31
`year`	int	Year component of modification date 1900+
`hour`	int	Hour component of modification date 0-23
`minutes`	int	Minute component of modification date 0-59
`seconds`	int	Second component of modification date 0-59
`type`	char[256]	Type of file (if not set by DLL, Dreamweaver uses file extensions to determine type, as it does now)
`size`	int	In bytes

- The *`numItems`* argument is the number of items that are allocated for the `itemList` (returned from `GetFolderListLength`).

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Get()

Description

This function gets a list of files or folders and stores them locally.

Arguments

*void *connectionData*, *const char *remotePathList[]*, *const char *localPathList[]*, *const int numItems*

- The *connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *remotePathList* argument is a list of the remote files or folders to retrieve, which is specified as complete paths and names.
- The *localPathList* argument is a mirrored list of local filenames or folder paths.
- The *numItems* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Put()

Description

This function puts a list of local files or folders into the source control system.

Arguments

*void *connectionData*, *const char *localPathList[]*, *const char *remotePathList[]*, *const int numItems*

- The *connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *localPathList* argument is the list of local filenames or folder paths to put into the source control system.
- The *remotePathList* argument is a mirrored list of remote filenames or folder paths.
- The *numItems* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_NewFolder()

Description

This function creates a new folder.

Arguments

*void *connectionData*, *const char *remotePath*

- The *connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

- The *remotePath* argument is the full path of the remote folder that the DLL creates.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Delete()

Description

This function deletes a list of files or folders from the source control system.

Arguments

*void *connectionData*, *const char *remotePathList[]*, *const int numItems*

- The *connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *remotePathList* argument is a list of remote filenames or folder paths to delete.
- The *numItems* argument is the number of items in *remotePathList*.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Rename()

Description

This function renames or moves a file or folder, depending on the values that are specified for *oldRemotePath* and *newRemotePath*. For example, if *oldRemotePath* equals `"$/folder1/file1"` and *newRemotePath* equals `"$/folder1/renamefile1"`, file1 is renamed renamefile1 and is located in folder1.

If *oldRemotePath* equals `"$/folder1/file1"` and *newRemotePath* equals `"$/folder1/subfolder1/file1"`, file1 is moved to the subfolder1 folder.

To find out if an invocation of this function is a move or a rename, check the parent paths of the two input values; if they are the same, the operation is a rename.

Arguments

*void *connectionData*, *const char *oldRemotePath*, *const char *newRemotePath*

- The *connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *oldRemotePath* argument is a remote file or folder path to rename.
- The *newRemotePath* argument is the remote path of the new name for the file or folder.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_ItemExists()

Description

This function determines whether a file or folder exists on the server.

Arguments

`void *connectionData`, *`const char *remotePath`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is a remote file or folder path.

Returns

A Boolean value: `true` if successful; `false` otherwise.

The Source Control Integration API optional functions

The Source Control Integration API has required and optional functions. The functions in this section are optional.

bool SCS_GetConnectionInfo()

Description

This function displays a dialog box to let the user change or set the connection information for this site. It does not make the connection. This function is called when the user clicks the Settings button in the Remote Info section of the Edit Sites dialog box.

Arguments

`void **connectionData`, *`const char siteName[64]`*

- The *`connectionData`* argument is a handle to data that the agent wants Dreamweaver to pass it when calling other API functions.
- The *`siteName`* argument is a string that points to the name of the site. The name cannot exceed 64 characters.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_SiteDeleted()

Description

This function notifies the DLL that the site has been deleted or that the site is no longer tied to this source control system. It indicates that the source control system can delete its persistent information for this site.

Arguments

`const char siteName[64]`

- The *`siteName`* argument is a string that points to the name of the site. The name cannot exceed 64 characters.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_SiteRenamed()

Description

This function notifies the DLL when the user has renamed the site so that it can update its persistent information about the site.

Arguments

`const char oldSiteName[64]`, *`const char newSiteName[64]`*

- The *`oldSiteName`* argument is a string that points to the original name of the site before it was renamed. The name cannot exceed 64 characters.
- The *`newSiteName`* argument is a string that points to the new name of the site after it was renamed. The name cannot exceed 64 characters.

Returns

A Boolean value: `true` if successful; `false` otherwise.

int SCS_GetNumNewFeatures()

Description

This function returns the number of new features to add to Dreamweaver (for example, File History, Differences, and so on).

Arguments

None.

Returns

An integer that indicates the number of new features to add to Dreamweaver. If the function returns `< 0`, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

bool SCS_GetNewFeatures()

Description

This function returns a list of menu items to add to the Dreamweaver main and context menus. For example, the Sourcesafe DLL can add History and File Differences to the main menu.

Arguments

char menuItemList[][32], *scFunction functionList[], scFunction enablerList[]*, *const int numNewFeatures*

- The *menuItemList* argument is a string list that is populated by the DLL; it specifies the menu items to add to the main and context menus. Each string can contain a maximum of 32 characters.
- The *functionList* argument is populated by the DLL; it specifies the routines in the DLL to call when the user selects the corresponding menu item.
- The *enablerList* argument is populated by the DLL; it specifies the routines in the DLL to call when Dreamweaver needs to determine whether the corresponding menu item is enabled.
- The *numNewFeatures* argument is the number of items being added by the DLL; this value is retrieved from the `GetNumNewFeatures()` call.

The following function signature defines the functions and enablers that passed to the `SCS_GetNewFeatures()` call in the `functionlist` and `enablerList` arguments.

```
bool (*scFunction)(void *connectionData, const char *remotePathList[],
  const char *localPathList[], const int numItems)
```

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_GetCheckoutName()

Description

This function returns the check-out name of the current user. If it is unsupported by the source control system and this feature is enabled by the user, this function uses the Dreamweaver internal Check In/Check Out functionality, which transports LCK files to and from the source control system.

Arguments

*void *connectionData*, *char checkOutName[64]*, *char emailAddress[64]*

- The *connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *checkOutName* argument is the name of the current user.
- The *emailAddress* argument is the e-mail address of the current user.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Checkin()

Description

This function checks a list of local files or folders into the source control system. The DLL is responsible for making the file read-only. If it is unsupported by the source control system and this feature is enabled by the user, this function uses the Dreamweaver internal Check In/Check Out functionality, which transports LCK files to and from the source control system.

Arguments

`void *connectionData`, *`const char *localPathList[]`*, *`const char *remotePathList[]`*, *`bool successList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`localPathList`* argument is a list of local filenames or folder paths to check in.
- The *`remotePathList`* argument is a mirrored list of remote filenames or folder paths.
- The *`successList`* argument is a list of Boolean values that are populated by the DLL to let Dreamweaver know which of the corresponding files are checked in successfully.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_Checkout()

Description

This function checks out a list of local files or folders from the source control system. The DLL is responsible for granting the privileges that let the file be writable. If it is unsupported by the source control system and this feature is enabled by the user, this function uses the Dreamweaver internal Check In/Check Out functionality, which transports LCK files to and from the source control system.

Arguments

`void *connectionData`, *`const char *remotePathList[]`*, *`const char *localPathList[]`*, *`bool successList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is a list of remote filenames or folder paths to check out.
- The *`localPathList`* argument is a mirrored list of local filenames or folder paths.
- The *`successList`* argument is a list of Boolean values that are populated by the DLL to let Dreamweaver know which of the corresponding files are checked out successfully.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_UndoCheckout()

Description

This function undoes the check-out status of a list of files or folders. The DLL is responsible for making the file read-only. If it is unsupported by the source control system and this feature is enabled by the user, this function uses the Dreamweaver internal Check In/Check Out functionality, which transports LCK files to and from the source control system.

Arguments

`void *connectionData`, *`const char *remotePathList[]`*, *`const char*localPathList[]`*, *`bool successList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is a list of remote filenames or folder paths on which to undo the check out.
- The *`localPathList`* argument is a mirrored list of local filenames or folder paths.
- The *`successList`* argument is a list of Boolean values that are populated by the DLL to let Dreamweaver know which corresponding files' check outs are undone successfully.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

int SCS_GetNumCheckedOut()

Description

This function returns the number of users who have a file checked out.

Arguments

`void *connectionData`, *`const char *remotePath`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is the remote file or folder path to check to see how many users have it checked out.

Returns

An integer that indicates the number of people who have the file checked out. If the function returns `< 0`, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

bool SCS_GetFileCheckoutList()

Description

This function returns a list of users who have a file checked out. If the list is empty, no one has the file checked out.

Arguments

`void *connectionData`, *`const char *remotePath`*, *`char checkOutList[][64]`*, *`char emailAddressList[][64]`*, *`const int numCheckedOut`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is the remote file or folder path to check how many users have it checked out.
- The *`checkOutList`* argument is a list of strings that corresponds to the users who have the file checked out. Each user string cannot exceed a maximum length of 64 characters.
- The *`emailAddressList`* argument is a list of strings that corresponds to the users' e-mail addresses. Each e-mail address string cannot exceed a maximum length of 64 characters.
- The *`numCheckedOut`* argument is the number of people who have the file checked out. This is returned from `GetNumCheckedOut()`.

Returns

A Boolean value: `true` if successful; `false` otherwise.

int SCS_GetErrorMessageLength()

Description

This function returns the length of the DLL's current internal error message. This allocates the buffer that passes into the `GetErrorMessage()` function. This function should be called only if an API function returns `false` or `<0`, which indicates a failure of that API function.

Arguments

`void *connectionData`

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

An integer that represents the length of the error message.

bool SCS_GetErrorMessage()

Description

This function returns the last error message. If you implement `getErrorMessage()`, Dreamweaver calls it each time one of your API functions returns the value `false`.

If a routine returns `-1` or `false`, it indicates an error message should be available.

Arguments

`void *connectionData`, *`char errorMsg[]`*, *`const int *msgLength`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`errorMsg`* argument is a preallocated string for the DLL to fill in with the error message.
- The *`msgLength`* argument is the length of the buffer represented by the *`errorMsg[]`* argument.

Returns

A Boolean value: `true` if successful; `false` otherwise.

int SCS_GetNoteCount()

Description

This function returns the number of Design Note keys for the specified remote file or folder path. If unsupported by the source control system, Dreamweaver gets this information from the companion MNO file.

Arguments

`void *connectionData`, *`const char *remotePath`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is the remote file or folder path that the DLL checks for the number of attached Design Notes.

Returns

An integer that indicates the number of Design Notes that are associated with this file. If the function returns `< 0`, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

int SCS_GetMaxNoteLength()

Description

This function returns the length of the largest Design Note for the specified file or folder. If it is unsupported by the source control system, Dreamweaver gets this information from the companionMNO file.

Arguments

*void *connectionData, const char *remotePath*

- The `connectionData` argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The `remotePath` argument is the remote file or folder path that the DLL checks for the maximum Design Note length.

Returns

An integer that indicates the size of the longest Design Note that is associated with this file. If the function returns `< 0`, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

bool SCS_GetDesignNotes()

Description

This function retrieves key-value pairs from the meta information for the specified file or folder. If it is unsupported by the source control system, Dreamweaver retrieves the information from the companionMNO file.

Arguments

*void *connectionData, const char *remotePath, char keyList[][64],*
*char *valueList[], bool showColumnList[], const int noteCount,*
const int noteLength

- The `connectionData` argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The `remotePath` argument is the remote file or folder path that the DLL checks for the number of items.
- The `keyList` argument is a list of Design Note keys, such as `"Status"`.
- The `valueList` argument is a list of Design Note values that correspond to the Design Note keys, such as `"Awaiting Signoff"`.
- The `showColumnList` argument is a list of Boolean values that correspond to the Design Note keys, which indicate whether Dreamweaver can display the key as a column in the Site panel.
- The `noteCount` argument is the number of Design Notes that are attached to a file or folder; the `GetNoteCount()` call returns this value.
- The `noteLength` argument is the maximum length of a Design Note; this is the value that the `GetMaxNoteLength()` call returns.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_SetDesignNotes()

Description

This function stores the key-value pairs in the meta information for the specified file or folder. This replaces the set of meta information for the file. If it is unsupported by the source control system, Dreamweaver stores Design Notes in MNO files.

Arguments

```
void *connectionData, const char *remotePath, const char keyList[][64],
  const char *valueList[], bool showColumnList[], const int noteCount,
  const int noteLength
```

- The `connectionData` argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The `remotePath` argument is the remote file or folder path that the DLL checks for the number of items.
- The `keyList` argument is a list of Design Note keys, such as `"Status"`.
- The `valueList` argument is a list of Design Note values that corresponds to the Design Note keys, such as `"Awaiting Signoff"`.
- The `showColumnList` argument is a list of Boolean values that correspond to the Design Note keys, which indicate whether Dreamweaver can display the key as a column in the Site panel.
- The `noteCount` argument is the number of Design Notes that are attached to a file or folder; this number lets the DLL know the size of the specified lists. If `noteCount` is 0, all the Design Notes are removed from the file.
- The `noteLength` argument is the length of the largest Design note for the specified file or folder.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_IsRemoteNewer()

Description

This function checks each specified remote path to see if the remote copy is newer. If it is unsupported by the source control system, Dreamweaver uses its internal `isRemoteNewer` algorithm.

Arguments

```
void *connectionData, const char *remotePathList[], const char *localPathList[],
int remoteIsNewerList[], const int numItems
```

- The `connectionData` argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The `remotePathList` argument is a list of remote filenames or folder paths to compare for newer status.
- The `localPathList` argument is a mirrored list of local filenames or folder paths.
- The `remoteIsNewerList` argument is a list of integers that are populated by the DLL to let Dreamweaver know which of the corresponding files is newer on the remote side. The following values are valid: 1 indicates the remote version is newer; -1 indicates the local version is newer; 0 indicates the versions are the same.
- The `numItems` argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

Enablers

If the optional enablers are not supported by the source control system or the application is not connected to the server, Dreamweaver determines when the menu items are enabled, based on the information it has about the remote files.

bool SCS_canConnect()

Description

This function returns whether the Connect menu item should be enabled.

Arguments

None.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canGet()

Description

This function returns whether the Get menu item should be enabled.

Arguments

`void *connectionData, const char *remotePathList[], const char *localPathList[], const int numItems`

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is a list of remote filenames or folder paths to get.
- The *`localPathList`* argument is a mirrored list of local filenames or folder paths.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canCheckout()

Description

This function returns whether the Checkout menu item should be enabled.

Arguments

`void *connectionData`, *`const char *remotePathList[]`*, *`const char *localPathList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is a list of remote filenames or folder paths to check out.
- The *`localPathList`* argument is a mirrored list of local filenames or folder paths.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canPut()

Description

This function returns whether the Put menu item should be enabled.

Arguments

`void *connectionData`, *`const char *localPathList[]`*, *`const char *remotePathList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`localPathList`* argument is a list of local filenames or folder paths to put into the source control system.
- The *`remotePathList`* argument is a mirrored list of remote filenames or folder paths to put into the source control system.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canCheckin()

Description

This function returns whether the Checkin menu item should be enabled.

Arguments

`void *connectionData`, *`const char *localPathList[]`*, *`const char *remotePathList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`localPathList`* argument is a list of local filenames or folder paths to check in.
- The *`remotePathList`* argument is a mirrored list of remote filenames or folder paths.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_CanUndoCheckout()

Description

This function returns whether the Undo Checkout menu item should be enabled.

Arguments

`void *connectionData`, *`const char *remotePathList[]`*, *`const char*localPathList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is a list of remote filenames or folder paths to check out.
- The *`localPathList`* argument is a list of the local filenames or folder paths to put to the source control system.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canNewFolder()

Description

This function returns whether the New Folder menu item should be enabled.

Arguments

`void *connectionData`, *`const char *remotePath`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePath`* argument is a list of remote filenames or folder paths that the user selected to indicate where the new folder will be created.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canDelete()

Description

This function returns whether the Delete menu item should be enabled.

Arguments

`void *connectionData`, *`const char *remotePathList[]`*, *`const int numItems`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is a list of remote filenames or folder paths to delete.
- The *`numItems`* argument is the number of items in each list.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_canRename()

Description

This function returns whether the Rename menu item should be enabled.

Arguments

`void *connectionData`, *`const char *remotePath`*

- The *`connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.
- The *`remotePathList`* argument is the remote filenames or folder paths that can be renamed.

Returns

A Boolean value: `true` if successful; `false` otherwise.

bool SCS_BeforeGet()

Description

Dreamweaver calls this function before getting or checking out one or more files. This function lets your DLL perform one operation, such as adding a check-out comment, to a group of files.

Arguments

`*connectionData`

- The *`*connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

A Boolean value: `true` if successful; `false` otherwise.

Example

To get a group of files, Dreamweaver makes calls to the DLL in the following order:

```
SCS_BeforeGet(connectionData);
SCS_Get(connectionData,remotePathList1,localPathList1,¬
successList1);
SCS_Get(connectionData,remotePathList2,localPathList2,¬
successList2);
SCS_Get(connectionData,remotePathList3,localPathList3,¬
successList3);
SCS_AfterGet(connectionData);
```

bool SCS_BeforePut()

Description

Dreamweaver calls this function before putting or checking in one or more files. This function lets your DLL perform one operation, such as adding a check-in comment, to a group of files.

Arguments

**connectionData*

- The **connectionData* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

A Boolean value: `true` if successful; `false` otherwise.

Example

To get a group of files, Dreamweaver makes calls to the DLL in the following order:

```
SCS_BeforePut(connectionData);
SCS_Put(connectionData,localPathList1,remotePathList1,¬
successList1);
SCS_Put(connectionData,localPathList2,remotePathList2,¬
successList2);
SCS_Put(connectionData,localPathList3,remotePathList3,¬
successList3);
SCS_AfterPut(connectionData);
```

bool SCS_AfterGet()

Description

Dreamweaver calls this function after getting or checking out one or more files. This function lets your DLL perform any operation after a batch get or check out, such as creating a summary dialog box.

Arguments

**connectionData*

- The **connectionData* argument is a pointer the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

A Boolean value: `true` if successful; `false` otherwise.

Example

See "bool SCS_BeforeGet()" on page 136.

bool SCS_AfterPut()

Description

Dreamweaver calls this function after putting or checking in one or more files. This function lets the DLL perform any operation after a batch put or check in, such as creating a summary dialog box.

Arguments

`*connectionData`

- The *`*connectionData`* argument is a pointer to the agent's data that passed into Dreamweaver during the `Connect()` call.

Returns

A Boolean value: `true` if successful; `false` otherwise.

Example

See "bool SCS_BeforePut()" on page 137.

PART II
JavaScript API

Use any of the more than 600 core JavaScript functions available in Macromedia Dreamweaver MX 2004, which encapsulate the kinds of tasks users perform when creating or editing a document. You can use these functions to perform any task that the user can accomplish using menus, floating panels, property inspectors, the Site panel, or the Document window.

CHAPTER 11
Application

The application functions perform operations related to Macromedia Dreamweaver MX 2004's interaction with other applications or Dreamweaver operations independent of individual documents (setting preferences, exiting Dreamweaver, and other functions).

External application functions

External application functions handle operations that are related to the Macromedia Flash MX 2004 application and to the browsers and external editors that are defined in the Preview in Browser and External Editors preferences. These functions let you get information about these external applications and open files with them.

dreamweaver.browseDocument()

Availability

Dreamweaver 2; enhanced in 3 and 4.

Description

Opens the specified URL in the specified browser.

Arguments

`fileName`, `{browser}`

- The `fileName` argument is the name of the file to open, which is expressed as an absolute URL.

  Note: Some browsers cannot locate the file if the URL contains an anchor, such as "Configuration/ExtensionHelp/browseHelp.htm#helpyou".

- The `browser` argument, which was added in Dreamweaver 3, specifies a browser. This argument can be the name of a browser, as defined in the Preview in Browser preferences or either `'primary'` or `'secondary'`. If the argument is omitted, the URL opens in the user's primary browser.

Returns

Nothing.

Example

The following function uses the `dreamweaver.browseDocument()` function to open the Hotwired home page in a browser:

```
function goToHotwired(){
  dreamweaver.browseDocument('http://www.hotwired.com/');
}
```

In Dreamweaver 4, you can expand this operation to open the document in Microsoft Internet Explorer using the following code:

```
function goToHotwired(){
  var prevBrowsers = dw.getBrowserList();
  var theBrowser = "";
  for (var i=1; i < prevBrowsers.length; i+2){
    if (prevBrowsers[i].indexOf('Iexplore.exe') != -1){
      theBrowser = prevBrowsers[i];
      break;
    }
  }
  dw.browseDocument('http://www.hotwired.com/',theBrowser);
}
```

For more information on the `dreamweaver.getBrowserList()` function, see "dreamweaver.getBrowserList()" on page 142.

dreamweaver.getBrowserList()

Availability

Dreamweaver 3.

Description

Gets a list of all the browsers in the File > Preview in Browser submenu.

Arguments

None.

Returns

An array that contains a pair of strings for each browser in the list. The first string in each pair is the name of the browser, and the second string is its location on the user's computer, which is expressed as a file:// URL. If no browsers appear in the submenu, the function returns nothing.

dreamweaver.getExtensionEditorList()

Availability

Dreamweaver 3

Description

Gets a list of editors for the specified file from the External Editors preferences.

Arguments

`fileURL`

- The `fileURL` argument can be a complete file:// URL, a filename, or a file extension (including the period).

Returns

An array that contains a pair of strings for each editor in the list. The first string in each pair is the name of the editor, and the second string is its location on the user's computer, which is expressed as a file:// URL. If no editors appear in Preferences, the function returns an array that contains one empty string.

Example

A call to the `dreamweaver.getExtensionEditorList(".gif")` function might return an array that contains the following strings:

- `"Fireworks 3"`
- `"file:///C|/Program Files/Macromedia/Fireworks 3/Fireworks 3.exe"`

dreamweaver.getExternalTextEditor()

Availability

Dreamweaver 4.

Description

Gets the name of the currently configured external text editor.

Arguments

None.

Returns

A string that contains the name of the text editor that is suitable for presentation in the user interface (UI), not the full path.

dreamweaver.getFlashPath()

Availability

Dreamweaver MX.

Description

Gets the full path to the Flash MX application in the form of a file URL.

Arguments

None.

Returns

An array that contains two elements. Element [0] is a string that contains the name of the Flash MX editor. Element [1] is a string that contains the path to the Flash application on the local computer, which is expressed as a file:// URL. If Flash is not installed, it returns nothing.

Example

The following example calls the `dw.getFlashPath()` function to obtain the path to the Flash application and then passes the path in the form of a file://URL to the `dw.openWithApp()` function to open the document with Flash:

```
var myDoc = dreamweaver.getDocumentDOM();

if (dreamweaver.validateFlash()) {
  var flashArray = dreamweaver.getFlashPath();
  dreamweaver.openWithApp(myDoc.myForm.swfFilePath, flashArray[1]);
}
```

dreamweaver.getPrimaryBrowser()

Availability

Dreamweaver 3.

Description

Gets the path to the primary browser.

Arguments

None.

Returns

A string that contains the path on the user's computer to the primary browser, which is expressed as a file:// URL. If no primary browser is defined, it returns nothing.

dreamweaver.getPrimaryExtensionEditor()

Availability

Dreamweaver 3.

Description

Gets the primary editor for the specified file.

Arguments

`fileURL`

- The *`fileURL`* argument is the path to the file to open, which is expressed as a file:// URL.

Returns

An array that contains a pair of strings. The first string in the pair is the name of the editor, and the second string is its location on the user's computer, which is expressed as a file:// URL. If no primary editor is defined, the function returns an array that contains one empty string.

dreamweaver.getSecondaryBrowser()

Availability

Dreamweaver 3.

Description

Gets the path to the secondary browser.

Arguments

None.

Returns

A string that contains the path on the user's computer to the secondary browser, which is expressed as a file:// URL. If no secondary browser is defined, it returns nothing.

dreamweaver.openHelpURL()

Availability

Dreamweaver MX.

Description

Opens the specified Help file in the operating system Help viewer.

Dreamweaver displays help content in the standard operating system help viewer instead of a browser. Help content is in HTML, but it is packaged for Windows HTML Help or Help Viewer for Mac OS X.

The following four types of files comprise the full help content. For more information on Help files, see your operating system documentation.

- Help book

  A Help book consists of the HTML Help files, images, and indexes. In Windows, the Help book is a file that has a name with a .chm extension. On the Macintosh, the Help book is a folder.

  The Help book files reside in the Dreamweaver Help folder.

- The help.xml file

  The help.xml file maps book IDs to help book names. For example, the following XML code maps the book ID for Dreamweaver Help to the filenames that contains help on both the Windows and Macintosh operating systems:

  ```
  <?xml version = "1.0" ?>
  <help-books>
  <book-id id="DW_Using" win-mapping="UsingDreamweaver.chm" mac-
  mapping="Dreamweaver Help"/>
  </help-books>
  ```

Each `book-id` entry has the following attributes:

  - The `id` attribute is the book ID that is used in the help.map and HelpDoc.js files.
  - The `win-mapping` attribute is the Windows book name, which is `"UsingDreamweaver.chm"` in this example.
  - The `mac-mapping` attribute is the Macintosh book name, which is `"Dreamweaver Help"` in this example.

- The help.map file

  The help.map file maps a help content ID to a specific help book. Dreamweaver uses the help.map file to locate specific help content when it calls help internally.

- The helpDoc.js file

  The helpDoc.js file lets you map variable names that you can use in place of the actual book ID and page string. The helpDoc.js file maps a help content ID to an HTML page in a specific help book. Dreamweaver uses the helpDoc.js file when it calls help from JavaScript.

Arguments

`bookID`

- The `bookID` argument, which is required, has the following format:

  `ID:page`

  The `ID` portion is the `bookID` of the entry in the help.xml file that names the file that contains the help content to display. The `page` portion of the entry identifies the specific page to display. The pages are referenced in the help.map file.

Returns

A value of `true` if successful; `false` if Dreamweaver cannot open the specified file in the help viewer.

Example

```
openHelpURL("DW_Using:index.htm");
```

dreamweaver.openWithApp()

Availability

Dreamweaver 3.

Description

Opens the specified file with the specified application.

Arguments

`fileURL`, *`appURL`*

- The *`fileURL`* argument is the path to the file to open, which is expressed as a file:// URL.
- The *`appURL`* argument is the path to the application that is to open the file, which is expressed as a file:// URL.

Returns

Nothing.

dreamweaver.openWithBrowseDialog()

Availability

Dreamweaver 3.

Description

Opens the Select External Editor dialog box to let the user select the application with which to open the specified file.

Arguments

`fileURL`

- The *`fileURL`* argument is the path to the file to open, which is expressed as a file:// URL.

Returns

Nothing.

dreamweaver.openWithExternalTextEditor()

Availability

Dreamweaver 3.

Description

Opens the current document in the external text editor that is specified in the External Editors entry in the Preferences dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.openWithImageEditor()

Availability

Dreamweaver 3.

Description

Opens the named file with the specified image editor.

Note: This function invokes a special Macromedia Fireworks MX 2004 integration mechanism that returns information to the active document if Fireworks is specified as the image editor. To prevent errors if no document is active, this function should never be called from the Site panel.

Arguments

`fileURL`, *`appURL`*

- The *`fileURL`* argument is the path to the file to open, which is expressed as a file:// URL.
- The *`appURL`* argument is the path to the application with which to open the file, which is expressed as a file:// URL.

Returns

Nothing.

dreamweaver.validateFlash()

Availability

Dreamweaver MX.

Description

Determines whether Flash MX (or a later version) is installed on the local computer.

Arguments

None.

Returns

A Boolean value: `true` if Flash MX (or a later version) is installed on the local computer; `false` otherwise.

Global application functions

Global application functions act on the entire application. They handle tasks such as quitting and accessing Preferences.

dreamweaver.beep()

Availability

Dreamweaver MX.

Description

Creates a system beep.

Arguments

None.

Returns

Nothing.

Example

The following example calls `dw.beep()` to call the user's attention to a message that the `alert()` function displays:

```
beep(){
  if(confirm("Is your order complete?")
  {
    dreamweaver.beep();
    alert("Click OK to submit your order");
  }
}
```

dreamweaver.getShowDialogsOnInsert()

Availability

Dreamweaver 3.

Description

Checks whether the Show Dialog When Inserting Objects option is turned on in the General category of Preferences.

Arguments

None.

Returns

A Boolean value that indicates whether the option is on.

dreamweaver.quitApplication()

Availability

Dreamweaver 3.

Description

Quits Dreamweaver after the script that calls this function finishes executing.

Arguments

None.

Returns

Nothing.

dreamweaver.showAboutBox()

Availability

Dreamweaver 3.

Description

Opens the About dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.showDynamicDataDialog()

Availability

Dreamweaver UltraDev 1.

Description

Displays the Dynamic Data or the Dynamic Text dialog box, and waits for the user to dismiss the dialog box. If the user clicks OK, the `showDynamicDataDialog()` function returns a string to insert into the user's document. (This string returns from the Data Sources API function, `generateDynamicDataRef()`, and passes to the Data Format API function, `formatDynamicDataRef()`; the return value from `formatDynamicDataRef()` is the one that the `showDynamicDataDialog()` function returns.)

Arguments

`source`, *`{title}`*

- The *`source`* argument is a string that contains source code, which represents the dynamic data object. It is the same string that a previous call to this function returned. The function uses the contents of the *`source`* argument to initialize all the dialog box controls, so they appear exactly as when the user clicked OK to create this string.

  Dreamweaver passes this string to the `inspectDynamicDataRef()` function to determine if the string matches any of the nodes in the tree. If the string matches a node, that node is selected when the dialog box appears. You can also pass an empty string, which does not initialize the dialog box. For example, a dialog box is not initialized when used to create a new item.
- The *`title`* argument, which is optional, is a string that contains the text to display in the title bar of the dialog box. If this argument is not supplied, Dreamweaver displays Dynamic Data in the title bar.

Returns

A string that represents the dynamic data object, if the user clicks OK.

dreamweaver.showPreferencesDialog()

Availability

Dreamweaver 3.

Description

Opens the Preferences dialog box.

Arguments

{strCategory}

- The *strCategory* argument, which is optional, must be one of the following strings to open the correlating category of the Preferences dialog box: `"general"`, `"accessibility"`, `"html colors"` (for the Code Coloring category), `"html format"` (for the Code Format category), `"code hints"`, `"html rewriting"` (for the Code Rewriting category), `"css styles"`, `"external editors"` (for the File Types/Editors category), `"fonts"`, `"highlighting"`, `"invisible elements"`, `"layers"`, `"layout mode"`, `"new document"`, `"office"`, `"floaters"` (for the Panels category), `"browsers"` (for the Preview in Browser category), `"site ftp"`, `"status bar"`, and `"validator"`. If Dreamweaver does not recognize the argument as a valid pane name, or if the argument is omitted, the dialog box opens to the last active pane.

Returns

Nothing.

dreamweaver.showTagChooser()

Availability

Dreamweaver MX.

Description

Toggles the visibility of the Tag Chooser dialog box for users to insert tags into the Code view. The function shows the Tag Chooser dialog box on top of all other Dreamweaver windows. If the dialog box is not visible, the function opens it, brings it to the front, and sets focus to it. If the Tag Chooser is visible, the function hides the dialog box.

Arguments

None.

Returns

Nothing.

CHAPTER 12
Workspace

Workspace API functions create or operate on an element of the Macromedia Dreamweaver MX 2004 workspace. They perform tasks such as redoing steps that appear in the History panel, placing an object on the Insert bar, navigating with Keyboard functions, reloading menus, manipulating standalone or built-in results windows, setting options, positioning a toolbar, and getting or setting focus.

History functions

History functions handle undoing, redoing, recording, and playing steps that appear in the History panel. A step is any repeatable change to the document or to a selection in the document. Methods of the `dreamweaver.historyPalette` object either control or act on the selection in the History panel, not in the current document.

dom.redo()

Availability

Dreamweaver 3.

Description

Redoes the step that was most recently undone in the document.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canRedo()" on page 438.

dom.undo()

Availability

Dreamweaver 3.

Description

Undoes the previous step in the document.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canUndo()" on page 441 .

dreamweaver.getRedoText()

Availability

Dreamweaver 3.

Description

Gets the text that is associated with the editing operation that will be redone if the user selects Edit > Redo or presses Control+Y (Windows) or Command+Y (Macintosh).

Arguments

None.

Returns

A string that contains the text that is associated with the editing operation that will be redone.

Example

If the user's last action applied bold to selected text, a call to the `dreamweaver.getRedoText()` function returns `"Repeat Apply Bold"`.

dreamweaver.getUndoText()

Availability

Dreamweaver 3.

Description

Gets the text that is associated with the editing operation that will be undone if the user selects Edit > Undo or presses Control+Z (Windows) or Command+Z (Macintosh).

Arguments

None.

Returns

A string that contains the text that is associated with the editing operation that will be undone.

Example

If the user's last action applied a cascading style sheet (CSS) style to a selected range of text, a call to the `dreamweaver.getUndoText()` function returns `"Undo Apply <span>"`.

dreamweaver.playRecordedCommand()

Availability

Dreamweaver 3.

Description

Plays the recorded command in the active document.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canPlayRecordedCommand()" on page 445.

dreamweaver.redo()

Availability

Dreamweaver 3.

Description

Redoes the step that was most recently undone in the active Document window, dialog box, floating panel, or Site panel.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canRedo()" on page 446.

dreamweaver.startRecording()

Availability

Dreamweaver 3.

Description

Starts recording steps in the active document; the previously recorded command is immediately discarded.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.isRecording()" on page 451 (must return a value of `false`).

dreamweaver.stopRecording()

Availability

Dreamweaver 3.

Description

Stops recording without prompting the user.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.isRecording()" on page 451 (must return a `true` value).

dreamweaver.undo()

Availability

Dreamweaver 3.

Description

Undoes the previous step in the Document window, dialog box, floating panel, or Site panel that has focus.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canUndo()" on page 441 .

dreamweaver.historyPalette.clearSteps()

Availability

Dreamweaver 3.

Description

Clears all steps from the History panel and disables the Undo and Redo menu items.

Arguments

None.

Returns

Nothing.

dreamweaver.historyPalette.copySteps()

Availability

Dreamweaver 3.

Description

Copies the specified history steps to the Clipboard. Dreamweaver warns the user about possible unintended consequences if the specified steps include an unrepeatable action.

Arguments

`arrayOfIndices`

- The *`arrayOfIndices`* argument is an array of position indices in the History panel.

Returns

A string that contains the JavaScript that corresponds to the specified history steps.

Example

The following code copies the first four steps in the History panel:

```
dreamweaver.historyPalette.copySteps([0,1,2,3]);
```

dreamweaver.historyPalette.getSelectedSteps()

Availability

Dreamweaver 3.

Description

Determines which portion of the History panel is selected.

Arguments

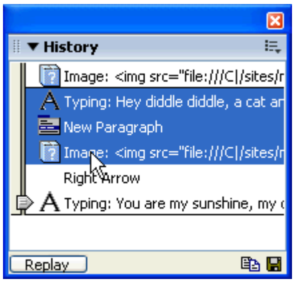
None.

Returns

An array that contains the position indices of all the selected steps. The first position is position 0 (zero).

Example

If the second, third, and fourth steps are selected in the History panel, as shown in the following figure, a call to the `dreamweaver.historyPalette.getSelectedSteps()` function returns `[1,2,3]`:


History
Image: <img src="file:///C|/sites/r
Typing: Hey diddle diddle, a cat ar
New Paragraph
Image: <img src="file:///C|/sites/r
Right Arrow
Typing: You are my sunshine, my
Replay

dreamweaver.historyPalette.getStepCount()

Availability

Dreamweaver 3.

Description

Gets the number of steps in the History panel.

Arguments

None.

Returns

An integer that represents the number of steps that are currently listed in the History panel.

dreamweaver.historyPalette.getStepsAsJavaScript()

Availability

Dreamweaver 3.

Description

Gets the JavaScript equivalent of the specified history steps.

Arguments

`arrayOfIndices`

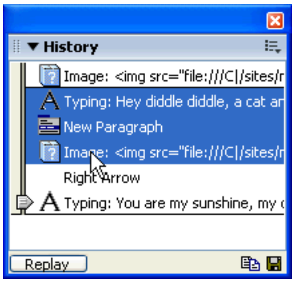
- The *`arrayOfIndices`* argument is an array of position indices in the History panel.

Returns

A string that contains the JavaScript that corresponds to the specified history steps.

Example

If the three steps shown in the following example are selected in the History panel, a call to the `dreamweaver.historyPalette.getStepsAsJavaScript(dw.historyPalette.getSelectedSteps())` function returns `"dw.getDocumentDOM().insertText('Hey diddle diddle, a cat and a fiddle, the cow jumped over the moon.');\ndw.getDocumentDOM().newBlock();\n dw.getDocumentDOM().insertHTML('<img src=\"../wdw99/50browsers/images/sun.gif\">', true);\n":`


History
Image: <img src="file:///C|/sites/r
Typing: Hey diddle diddle, a cat ar
New Paragraph
Image: <img src="file:///C|/sites/r
Right Arrow
Typing: You are my sunshine, my
Replay

dreamweaver.historyPalette.getUndoState()

Availability

Dreamweaver 3.

Description

Gets the current undo state.

Arguments

None.

Returns

The position of the Undo marker in the History panel.

dreamweaver.historyPalette.replaySteps()

Availability

Dreamweaver 3.

Description

Replays the specified history steps in the active document. Dreamweaver warns the user of possible unintended consequences if the specified steps include an unrepeatable action.

Arguments

`arrayOfIndices`

- The *`arrayOfIndices`* argument is an array of position indices in the History panel.

Returns

A string that contains the JavaScript that corresponds to the specified history steps.

Example

A call to `dreamweaver.historyPalette.replaySteps([0,2,3])` function plays the first, third, and fourth steps in the History panel.

dreamweaver.historyPalette.saveAsCommand()

Availability

Dreamweaver 3.

Description

Opens the Save As Command dialog box, which lets the user save the specified steps as a command. Dreamweaver warns the user of possible unintended consequences if the steps include an unrepeatable action.

Arguments

`arrayOfIndices`

- The *`arrayOfIndices`* argument is an array of position indexes in the History panel.

Returns

A string that contains the JavaScript that corresponds to the specified history steps.

Example

The following code saves the fourth, sixth, and eighth steps in the History panel as a command:

```
dreamweaver.historyPalette.saveAsCommand([3,5,7]);
```

dreamweaver.historyPalette.setSelectedSteps()

Availability

Dreamweaver 3.

Description

Selects the specified steps in the History panel.

Arguments

`arrayOfIndices`

- The *`arrayOfIndices`* function is an array of position indices in the History panel. If no argument is supplied, all the steps are unselected.

Returns

Nothing.

Example

The following code selects the first, second, and third steps in the History panel:

```
dreamweaver.historyPalette.setSelectedSteps([0,1,2]);
```

dreamweaver.historyPalette.setUndoState()

Availability

Dreamweaver 3.

Description

Performs the correct number of undo or redo operations to arrive at the specified undo state.

Arguments

`undoState`

- The *`undoState`* argument is the object that the `dreamweaver.historyPalette.getUndoState()` function returns.

Returns

Nothing.

Insert object functions

Insert object functions handle operations related to the objects on the Insert bar or listed on the Insert menu.

dom.insertFlashElement()

Availability

Dreamweaver MX 2004.

Description

Inserts a specified Flash element (SWC file) into the current document. This function assumes that the Flash element has been added to the Insert bar, and the component file resides in the Configuration/Objects/FlashElements folder or subfolder.

Arguments

`swcFilename`

- The *`swcFilename`* string is the path and name of the desired flash component relative to the Configuration/Objects/FlashElements folder.

Returns

Nothing.

Example

The following example inserts the navigation bar Flash component, which resides in the Components/Objects/FlashElements/Navigation folder, into the current document:

```
dom.insertFlashElement("\Navigation\navBar.swc");
```

dreamweaver.objectPalette.getMenuDefault()

Availability

Dreamweaver MX 2004.

Description

Retrieves the ID string of the default item for the associated menu.

Arguments

`menuId`

- The *`menuId`* argument is the string that defines the menu in the insertbar.xml file.

Returns

A string value defining the ID of the default item.

Example

The following example assigns the current default object for the Media menu to the `defID` variable:

```
var defId = dw.objectPalette.getMenuDefault("DW_Media");
```

dreamweaver.objectPalette.setMenuDefault()

Availability

Dreamweaver MX 2004.

Description

Sets the default object for a pop-up menu. The default object's icon represents the specified pop-up menu on the Insert bar. The user can click on the default object to insert it, or click on the arrow beside the default object to open the pop-up menu and see the other objects in that menu. Dreamweaver sets the new menu default the next time the user opens Dreamweaver or uses the Reload Extensions command.

Arguments

`menuId, defaultId`

- The *`menuId`* argument is the string that defines the menu in the insertbar.xml file.
- The *`defaultId`* argument is the string that defines the new default object in the insertbar.xml field.

Returns

A Boolean value: `true` if the new default is successfully set; `false` otherwise.

Example

The following example sets the Flash object as the default object for the Media menu:

```
dw.objectPalette.setMenuDefault("DW_Media", "DW_Flash");
```

dreamweaver.reloadObjects()

Availability

Dreamweaver MX 2004.

Description

Reloads all the objects on the Insert bar. This function is the equivalent of Control+left-clicking the mouse on the Categories menu on the Insert bar and selecting the Reload Extensions menu option.

Arguments

None.

Returns

A Boolean value: `true` if the objects were successfully loaded; `false` otherwise.

Keyboard functions

Keyboard functions mimic document navigation tasks that are accomplished by pressing the arrow, Backspace, Delete, Page Up, and Page Down keys. In addition to such general arrow and key functions as `arrowLeft()` and `backspaceKey()`, Dreamweaver also provides methods for moving to the next or previous word or paragraph as well as moving to the start of the line or document or the end of the line or document.

dom.arrowDown()

Availability

Dreamweaver 3.

Description

Moves the insertion point down the specified number of times.

Arguments

`{nTimes}`, *`{bShiftIsDown}`*

- The *`nTimes`* argument is the number of times that the insertion point must move down. If this argument is omitted, the default is 1.
- The *`bShiftIsDown`* argument is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.arrowLeft()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the left the specified number of times.

Arguments

`{nTimes}`, `{bShiftIsDown}`

- The `nTimes` argument, which is optional, is the number of times that the insertion point must move left. If this argument is omitted, the default is 1.
- The `bShiftIsDown` argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.arrowRight()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the right the specified number of times.

Arguments

`{nTimes}`, `{bShiftIsDown}`

- The `nTimes` argument, which is optional, is the number of times that the insertion point must move right. If this argument is omitted, the default is 1.
- The `bShiftIsDown` argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.arrowUp()

Availability

Dreamweaver 3.

Description

Moves the insertion point up the specified number of times.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of times that the insertion point must move up. If this argument is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.backspaceKey()

Availability

Dreamweaver 3.

Description

This function is equivalent to pressing the Backspace key a specified number of times. The exact behavior depends on whether there is a current selection or only an insertion point.

Arguments

{nTimes}

- The *nTimes* argument, which is optional, is the number of times that a Backspace operation must occur. If the argument is omitted, the default is 1.

Returns

Nothing.

dom.deleteKey()

Availability

Dreamweaver 3.

Description

This function is equivalent to pressing the Delete key the specified number of times. The exact behavior depends on whether there is a current selection or only an insertion point.

Arguments

{nTimes}

- The *nTimes* argument, which is optional, is the number of times that a Delete operation must occur. If the argument is omitted, the default is 1.

Returns

Nothing.

dom.endOfDocument()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the end of the document (that is, after the last visible content in the Document window or after the closing `HTML` tag in the Code inspector, depending on which window has focus).

Arguments

`{bShiftIsDown}`

- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If the argument is omitted, the default is `false`.

Returns

Nothing.

dom.endOfLine()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the end of the line.

Arguments

`{bShiftIsDown}`

- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If the argument is omitted, the default is `false`.

Returns

Nothing.

dom.nextParagraph()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the beginning of the next paragraph or skips multiple paragraphs if *`nTimes`* is greater than 1.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of paragraphs that the insertion point must move ahead. If this argument is omitted, the default is 1.
- The *bShiftIsDown* argument is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.nextWord()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the beginning of the next word or skips multiple words if *nTimes* is greater than 1.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of words that the insertion point must move ahead. If this argument is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.pageDown()

Availability

Dreamweaver 3.

Description

Moves the insertion point down one page (equivalent to pressing the Page Down key).

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of pages that the insertion point must move down. If this argument is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.pageUp()

Availability

Dreamweaver 3.

Description

Moves the insertion point up one page (equivalent to pressing the Page Up key).

Arguments

`{nTimes}`, *`{bShiftIsDown}`*

- The *`nTimes`* argument, which is optional, is the number of pages that the insertion point must move up. If this argument is omitted, the default is 1.
- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.previousParagraph()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the beginning of the previous paragraph or skips multiple paragraphs if *`nTimes`* is greater than 1.

Arguments

`{nTimes}`, *`{bShiftIsDown}`*

- The *`nTimes`* argument, which is optional, is the number of paragraphs that the insertion point must move back. If this argument is omitted, the default is 1.
- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.previousWord()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the beginning of the previous word or skips multiple words if *`nTimes`* is greater than 1.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of words that the insertion point must move back. If this argument is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

dom.startOfDocument()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the beginning of the document (that is, before the first visible content in the Document window, or before the opening HTML tag in the Code inspector, depending on which window has focus).

Arguments

{bShiftIsDown}

- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If the argument is omitted, the default is `false`.

Returns

Nothing.

dom.startOfLine()

Availability

Dreamweaver 3.

Description

Moves the insertion point to the beginning of the line.

Arguments

{bShiftIsDown}

- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether to extend the selection. If the argument is omitted, the default is `false`.

Returns

Nothing.

dreamweaver.mapKeyCodeToChar()

Availability

Dreamweaver 4.

Description

Takes a key code as retrieved from the event object's `keyCode` field and translates it to a character. You should check whether the key code is a special key, such as HOME, PGUP, and so on. If the key code is not a special key, this method can be used to translate it to a character code that is suitable for display to the user.

Arguments

`keyCode`

- The *`keyCode`* argument is the key code to translate to a character.

Returns

Nothing.

Menu functions

Menu functions handle optimizing and reloading the menus in Dreamweaver. The `dreamweaver.getMenuNeedsUpdating()` function and the `dreamweaver.notifyMenuUpdated()` function are designed specifically to prevent unnecessary update routines from running on the dynamic menus that are built into Dreamweaver. See dreamweaver.getMenuNeedsUpdating() and dreamweaver.notifyMenuUpdated() for more information.

dreamweaver.getMenuNeedsUpdating()

Availability

Dreamweaver 3.

Description

Checks whether the specified menu needs to be updated.

Arguments

`menuId`

- The *`menuId`* argument is a string that contains the value of the `id` attribute for the menu item, as specified in the menus.xml file.

Returns

A Boolean value that indicates whether the menu needs to be updated. This function returns `false` only if `dreamweaver.notifyMenuUpdated()` has been called with this *`menuId`*, and the return value of *`menuListFunction`* has not changed. For more information, see "dreamweaver.notifyMenuUpdated()" on page 171.

dreamweaver.notifyMenuUpdated()

Availability

Dreamweaver 3.

Description

Notifies Dreamweaver when the specified menu needs to be updated.

Arguments

`menuId`, *`menuListFunction`*

- The *`menuId`* argument is a string that contains the value of the `id` attribute for the menu item, as specified in the menus.xml file.
- The *`menuListFunction`* argument must be one of the following strings: `"dw.cssStylePalette.getStyles()"`, `"dw.getDocumentDOM().getFrameNames()"`, `"dw.getDocumentDOM().getEditableRegionList"`, `"dw.getBrowserList()"`, `"dw.getRecentFileList()"`, `"dw.getTranslatorList()"`, `"dw.getFontList()"`, `"dw.getDocumentList()"`, `"dw.htmlStylePalette.getStyles()"`, or `"site.getSites()"`.

Returns

Nothing.

dreamweaver.reloadMenus()

Availability

Dreamweaver 3.

Description

Reloads the entire menu structure from the menus.xml file in the Configuration folder.

Arguments

None.

Returns

Nothing.

Results window functions

Results window functions let you create a stand-alone window that displays columns of formatted data, or you can interact with the built-in windows of the Results panel group.

Creating a Stand-alone Results window

These functions create custom windows that are similar to the output from the JavaScript Debugger window.

dreamweaver.createResultsWindow()

Availability

Dreamweaver 4.

Description

Creates a new Results window and returns a JavaScript object reference to the window.

Arguments

`strName`, *`arrColumns`*

- The *`strName`* argument is the string to use for the window's title.
- The *`arrColumns`* argument is an array of column names to use in the list control.

Returns

An object reference to the created window.

dreamweaver.showResults()

Availability

Dreamweaver MX 2004.

Description

Opens the specified results floating panel and selects the item.

Note: This function is supported only in `the` Validation, Target Browser Check, and Site Reports windows of the Results floating panel.

Arguments

`floaterName, floaterIndex`

- The *`floaterName`* argument is a string that specifies the results floating panel to open. Valid values are `'validation'`, `'btc'`, or `'reports'`.
- The *`floaterIndex`* argument is a number or string. Use a number to specify the index of an item to select in the Results panel. Use a string to specify the URL of a document. If you specify a URL, the function selects the first visible item for that document.

Returns

Nothing.

Example

The following example checks for errors at the offset of the current selection in the document and, if there are errors, displays them in the specified window (`floaterName`) of the Results panel. Otherwise, it opens the Target Browser Check window of the Results panel and displays the first visible item for the document.

```
var offset = dw.getDocumentDOM().source.getSelection()[0];
var errors = dw.getDocumentDOM().source.getValidationErrorsForOffset(offset);
if ( errors && errors.length > 0 )
  dw.showResults( errors[0].floaterName, errors[0].floaterIndex );
else
  dw.showResults('btc', dw.getDocumentDOM().URL);
```

resWin.addItem()

Availability

Dreamweaver 4.

Description

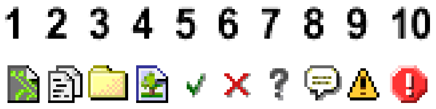
Adds a new item to the Results window.

Note: Use only on stand-alone results windows created with dreamweaver.createResultsWindow(). resWin.addItem() cannot be used with the built-in results windows, including Validation, Browser Target Check, or Site Reports.

Arguments

`resultWindowObj, strIcon`, *`strDesc`*, *`itemData`*, *`iStartSel`*, *`iEndSel`*, *`colNdata`*

- The *`resultWindowObj`* argument is the object that the `createResultsWindow()` function returns.
- The *`strIcon`* argument is a string that specifies the path to the icon to use. To display a built-in icon, use a value "1" through "10" instead of the fully qualified path of the icon. Specify **"0"** (zero) for no icon. The following table shows the icons that correspond to the values of "1" through "10":


1
2
3
4
5
6
7
8
9
10

- The *`strDesc`* argument is a detailed description of the item. Specify **"0"** if there is no description.
- The *`itemData`* argument is a string you can use to store specific data about the item being added such as a document line number.
- The *`iStartSel`* argument is the start of selection offset in the file. Specify the value `null` if you are not specifying an offset.
- The *`iEndSel`* argument is the end of selection offset in the file. Specify the value `null` if you are not specifying an offset.
- The *`colNdata`* argument is an array of strings that provide the data for each column (that is, if there are 3 columns, an array of 3 strings).

Returns

A Boolean value: `true` if the item was added successfully; `false` otherwise.

Example

The following example creates a Results window called `resWin` that has the column headings: Frodo, Sam, and Gollum. The call to the `resWin.addItem()` function adds a folder icon and then the three strings, `msg1`, `msg2`, and `msg3` into the three columns defined for the window.

```
var resWin = dw.createResultsWindow("Test Window", ["Frodo", "Sam",
  "Gollum"]);
resWin.addItem(resWin, "3", "Description", null, null, null, ["msg1", "msg2",
  "msg3"]);
```

resWin.addResultItem()

Availability

Dreamweaver 4.

Description

Adds a new results entry to the current Results window, based on the information in the file that the `processfile()` function processes.

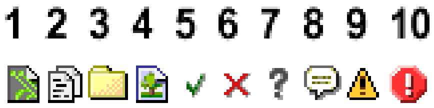
Note: Use only on the built-in results window for Site Reports (dreamweaver.resultsPalette.siteReports). resWin.addResultItem() cannot be used with other built-in results windows or stand-alone results windows created with dreamweaver.createResultsWindow().

This function is only available in the `processFile()` callback of a site report. See "Reports" in *Extending Dreamweaver* for details on site reports.

Arguments

`strFilePath, strIcon, strDisplay, strDesc, {iLineNo}, {iStartSel}, {iEndSel}`

- The *`strFilePath`* argument is a fully qualified URL path of the file to process.
- The *`strIcon`* argument is the path to the icon to use. To display a built-in icon, use a value "1" through "10" instead of the fully qualified path for the icon (use "0" for no icon). The following table shows the icons that correspond to the values of "1" through "10":


1
2
3
4
5
6
7
8
9
10

- The *`strDisplay`* argument is the string to display to the user in first column of the Results window (usually, the filename).
- The *`strDesc`* argument is the description that goes with the entry.
- The *`iLineNo`* argument is the number of lines in the file (optional).
- The *`iStartSel`* argument is the start of offset into the file (optional, but if it is used, the *`iEndSel`* argument must also be used.).
- The *`iEndSel`* argument is the end of offset into the file (required if *`iStartSel`* is used).

Returns

Nothing.

resWin.setCallbackCommands()

Availability

Dreamweaver 4.

Description

Tells the Results window on which commands to call the `processFile()` method. If this function is not called, the command that created the Results window is called.

Arguments

`arrCmdNames`

- The *`arrCmdNames`* argument is an array of command names on which to call the `processFile()` function.

Returns

Nothing.

resWin.setColumnWidths()

Availability

Dreamweaver 4.

Description

Sets the width of each column.

Arguments

`arrWidth`

- The *`arrWidth`* argument is an array of integers that represents the widths to use for each column in the control.

Returns

Nothing.

resWin.setFileList()

Availability

Dreamweaver 4.

Description

Gives the Results window a list of files, folders, or both to call a set of commands to process.

Arguments

`arrFilePaths`, *`bRecursive`*

- The *`arrFilePaths`* argument is an array of file or folder paths to iterate through.
- The *`bRecursive`* argument is a Boolean value that indicates whether the iteration should be recursive (`true`) or not (`false`).

Returns

Nothing.

resWin.setTitle()

Availability

Dreamweaver 4.

Description

Sets the title of the window.

Arguments

`strTitle`

- The *`strTitle`* argument is the new name of the floating panel.

Returns

Nothing.

resWin.startProcessing()

Availability

Dreamweaver 4.

Description

Starts processing the file.

Arguments

None.

Returns

Nothing.

resWin.stopProcessing()

Availability

Dreamweaver 4.

Description

Stops processing the file.

Arguments

None.

Returns

Nothing.

Working with the built-in Results panel group

These functions produce output in the Results panel group. The Results panel group displays tabbed reports on searches, source validation, sitewide reports, browser targets, console reports, FTP logging, and link-checking.

Working with specific child panels

The following child panels are built-in Results windows that always exist in the Dreamweaver interface and can be accessed directly. Because these panels are Results windows, you can use the following methods that are defined for stand-alone Results windows:

- `dreamweaver.resultsPalette.siteReports`
- `dreamweaver.resultsPalette.validator`
- `dreamweaver.resultsPalette.btc` (Target Browser Check panel)

For more information about using the `resWin` methods, see "Creating a Stand-alone Results window" on page 172.

Working with the active child panel

The following general API functions apply to whichever child panel is active. Some child panels might ignore some of these functions. If the active child panel does not support the function, calling it has no effect.

dreamweaver.resultsPalette.clear()

Availability

Dreamweaver MX.

Description

Clears the contents of the panel in focus.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canClear()" on page 452.

dreamweaver.resultsPalette.Copy()

Availability

Dreamweaver MX.

Description

Sends a copied message to the window that is in focus (often used for the FTP logging window).

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canCopy()" on page 452.

dreamweaver.resultsPalette.cut()

Availability

Dreamweaver MX.

Description

Sends a cut message to the window in focus (often used for the FTP logging window).

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canCut()" on page 452.

dreamweaver.resultsPalette.Paste()

Availability

Dreamweaver MX.

Description

Sends a pasted message to the window in focus (often used for the FTP logging window).

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canPaste()" on page 453.

dreamweaver.resultsPalette.openInBrowser

Availability

Dreamweaver MX.

Description

Sends a report (Site Reports, Browser Target Check, Validation, and Link Checker) to the default browser.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canOpenInBrowser()" on page 453.

dreamweaver.resultsPalette.openInEditor()

Availability

Dreamweaver MX.

Description

Jumps to the selected line for specific reports (Site Reports, Browser Target Check, Validation, and Link Checker), and opens the document in the editor.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canOpenInEditor()" on page 453.

dreamweaver.resultsPalette.save()

Availability

Dreamweaver MX.

Description

Opens the Save dialog box for a window that supports the Save function (Site Reports, Browser Target Check, Validation, and Link Checker).

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canSave()" on page 454.

dreamweaver.resultsPalette.selectAll()

Availability

Dreamweaver MX.

Description

Sends a Select All command to the window in focus.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.resultsPalette.canSelectAll()" on page 454.

Server debugging

Dreamweaver can request files from ColdFusion and display the response in its embedded browser. When the response returns from the server, Dreamweaver searches the response for a packet of XML that has a known signature. If Dreamweaver finds XML with that signature, it processes the XML and displays the contained information in a tree control. This tree displays information about the following items:

- All templates, custom tags, and include files that are used to generate the rendered CFM page
- Exceptions
- SQL queries
- Object queries
- Variables
- Trace trail

Additionally, the Server Debug panel can display debug data from other server models. To set up Dreamweaver to debug other server models, use the `dreamweaver.resultsPalette.debugWindow.addDebugContextData()` function.

dreamweaver.resultsPalette.debugWindow.addDebugContextData()

Availability

Dreamweaver MX.

Description

Interprets a customized XML file that returns from the server that is specified in the Site Definition dialog box. The contents of the XML file display as tree data in the Server Debug panel, so you can use the Server Debug panel to evaluate server-generated content from various server models.

Arguments

`treedata`

- The `treedata` argument is the XML string that the server returns. The XML string should use the following formatting:

`server debug node`	Root node for the debug XML data
`debugnode`	Corresponds to every node
`context`	Name of item that appears in the context list
`icon`	Icon to use for tree node
`name`	Name to display
`value`	Value to display
`timestamp`	Only applicable to context node
The following strings are optional:	
`jumptoline`	Link to a specific line number
`template`	Name of the template file part of the URL
`path`	Path of the file from server point of view
`line number`	Line number within the file
`start position`	Opening character offset within the line
`end position`	Ending character offset within the line

For example:

```
<serverdebuginfo>
  <context>
    <template><![CDATA[/ooo/master.cfm]]></template>
    <path><![CDATA[C:\server\wwwroot\ooo\master.cfm]]></path>
    <timestamp><![CDATA[0:0:0.0]]></timestamp>
  </context>
  <debugnode>
    <name><![CDATA[CGI]]></name>
    <icon><![CDATA[ServerDebugOutput/ColdFusion/CGIVariables.gif]]></icon>
    <debugnode>
      <name><![CDATA[Pubs.name.sourceURL]]></name>
      <icon><![CDATA[ServerDebugOutput/ColdFusion/Variable.gif]]></icon>
```

```
        <value><![CDATA[jdbc:macromedia:sqlserver://
    name.macromedia.com:1111;databaseName=Pubs]]></value>
      </debugnode>
    </debugnode>
    <debugnode>
      <name><![CDATA[Element Snippet is undefined in class
    coldfusion.compiler.TagInfoNotFoundException]]></name>
      <icon><![CDATA[ServerDebugOutput/ColdFusion/Exception.gif]]></icon>
      <jumptoline linenumber="3" startposition="2" endposition="20">
        <template><![CDATA[/ooo/master.cfm]]></template>
        <path><![CDATA[C:\Neo\wwwroot\ooo\master.cfm]]></path>
      </jumptoline>
    </debugnode>
  </serverdebuginfo>
```

Returns

Nothing.

Toggle functions

Toggle functions get and set various options either on or off.

dom.getEditNoFramesContent()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Modify > Frameset > Edit NoFrames Content option.

Arguments

None.

Returns

A Boolean value: `true` indicates the `NOFRAMES` content is the active view; `false` otherwise.

dom.getHideAllVisualAids()

Availability

Dreamweaver 4.

Description

This function determines whether visual aids are set as hidden.

Arguments

None.

Returns

A Boolean value: `true` sets Hide All Visual Aids to hidden; `false` otherwise.

dom.getPreventLayerOverlaps()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Prevent Layer Overlaps option.

Arguments

None.

Returns

A Boolean value: `true` turns on the Prevent Layer Overlaps option; `false` otherwise.

dom.getShowAutoIndent()

Availability

Dreamweaver 4.

Description

This function determines whether auto-indenting is on in the Code view of the document window.

Arguments

None.

Returns

A Boolean value: `true` if auto-indenting is on; `false` otherwise.

dom.getShowFrameBorders()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Frame Borders option.

Arguments

None.

Returns

A Boolean value: `true` indicates frame borders are visible; `false` otherwise.

dom.getShowGrid()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Grid > Show option.

Arguments

None.

Returns

A Boolean value: `true` indicates the grid is visible; `false` otherwise.

dom.getShowHeadView()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Head Content option.

Arguments

None.

Returns

A Boolean value: `true` indicates the head content is visible; `false` otherwise.

dom.getShowInvalidHTML()

Availability

Dreamweaver 4.

Description

This function determines whether invalid HTML code is currently highlighted in the Code view of the document window.

Arguments

None.

Returns

A Boolean value: `true` if invalid HTML code is highlighted; `false` otherwise.

dom.getShowImageMaps()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Image Maps option.

Arguments

None.

Returns

A Boolean value: `true` indicates the image maps are visible; `false` otherwise.

dom.getShowInvisibleElements()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Invisible Elements option.

Arguments

None.

Returns

A Boolean value: `true` indicates the invisible element markers are visible; `false` otherwise.

dom.getShowLayerBorders()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Layer Borders option.

Arguments

None.

Returns

A Boolean value: `true` indicates the layer borders are visible; `false` otherwise.

dom.getShowLineNumbers()

Availability

Dreamweaver 4.

Description

This function determines whether line numbers are shown in the Code view.

Arguments

None.

Returns

A Boolean value: `true` indicates the line numbers are shown; `false` otherwise.

dom.getShowRulers()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Rulers > Show option.

Arguments

None.

Returns

A Boolean value: `true` indicates the rulers are visible; `false` otherwise.

dom.getShowSyntaxColoring()

Availability

Dreamweaver 4.

Description

This function determines whether syntax coloring is on in the Code view of the document window.

Arguments

None.

Returns

A Boolean value: `true` if syntax coloring is on; `false` otherwise.

dom.getShowTableBorders()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Table Borders option.

Arguments

None.

Returns

A Boolean value: `true` indicates the table borders are visible; `false` otherwise.

dom.getShowToolbar()

Availability

Dreamweaver 4.

Description

This function determines whether the toolbar appears.

Arguments

None.

Returns

A Boolean value: `true` if the toolbar appears; `false` otherwise.

dom.getShowTracingImage()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Tracing Image > Show option.

Arguments

None.

Returns

A Boolean value: `true` indicates the option is on; `false` otherwise.

dom.getShowWordWrap()

Availability

Dreamweaver 4.

Description

This function determines whether word wrap is on in the Code view of the document window.

Arguments

None.

Returns

A Boolean value: `true` if word wrap is on; `false` otherwise.

dom.getSnapToGrid()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Grid > Snap To option.

Arguments

None.

Returns

A Boolean value: `true` indicates that the snap-to-grid option is on; `false` otherwise.

dom.setEditNoFramesContent()

Availability

Dreamweaver 3.

Description

This function toggles the Modify > Frameset > Edit NoFrames Content option on and off.

Arguments

`bEditNoFrames`

- The *`bEditNoFrames`* argument is a Boolean value: `true` turns on the Edit NoFrames Content option; `false` turns it off.

Returns

Nothing.

Enabler

"dom.canEditNoFramesContent()" on page 435.

dom.setHideAllVisualAids()

Availability

Dreamweaver 4.

Description

This function turns off the display of all borders, image maps, and invisible elements, regardless of their individual settings in the View menu.

Arguments

`bSet`

- The *`bSet`* argument is a Boolean value: `true` hides visual aids; `false` otherwise.

Returns

Nothing.

dom.setPreventLayerOverlaps()

Availability

Dreamweaver 3.

Description

This function toggles the Prevent Layer Overlaps option on and off.

Arguments

`bPreventLayerOverlaps`

- The *`bPreventLayerOverlaps`* argument is a Boolean value: `true` turns on the Prevent Layer Overlaps option; `false` turns it off.

Returns

Nothing.

dom.setShowFrameBorders()

Availability

Dreamweaver 3.

Description

This function toggles the View > Frame Borders option on and off.

Arguments

`bShowFrameBorders`

- The *`bShowFrameBorders`* argument is a Boolean value: `true` turns the Frame Borders on; `false` otherwise.

Returns

Nothing.

dom.setShowGrid()

Availability

Dreamweaver 3.

Description

This function toggles the View > Grid > Show option on and off.

Arguments

bShowGrid

- The *bShowGrid* argument is a Boolean value: `true` turns on the View > Grid > Show option; `false` turns it off.

Returns

Nothing.

dom.setShowHeadView()

Availability

Dreamweaver 3.

Description

This function toggles the View > Head Content option on and off.

Arguments

bShowHead

- The *bShowHead* argument is a Boolean value: `true` turns on the Head Content option; `false` turns it off.

Returns

Nothing.

dom.setShowInvalidHTML()

Availability

Dreamweaver 4.

Description

This function turns highlighting of invalid HTML code on or off in the Code view of the document window.

This function determines whether invalid HTML code is currently highlighted.

Arguments

bShow

- The *bShow* argument is a Boolean value: `true` indicates that highlighting invalid HTML code is visible; `false` otherwise.

Returns

Nothing.

dom.setShowImageMaps()

Availability

Dreamweaver 3.

Description

This function toggles the View > Image Maps option on and off.

Arguments

`bShowImageMaps`

- The *`bShowImageMaps`* argument is a Boolean value, `true` turns on the Image Maps option; `false` turns it off.

Returns

Nothing.

dom.setShowInvisibleElements()

Availability

Dreamweaver 3.

Description

This function toggles the View > Invisible Elements option on and off.

Arguments

`bViewInvisibleElements`

- The *`bViewInvisibleElements`* argument is a Boolean value: `true` turns on the Invisible Elements option; `false` turns it off.

Returns

Nothing.

dom.setShowLayerBorders()

Availability

Dreamweaver 3.

Description

This function toggles the View > Layer Borders option on and off.

Arguments

`bShowLayerBorders`

- The *`bShowLayerBorders`* argument is a Boolean value, `true` turns on the Layer Borders option; `false` turns it off.

Returns

Nothing.

dom.setShowLineNumbers()

Availability

Dreamweaver 4.

Description

This function shows or hides the line numbers in the Code view of the document window.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: `true` indicates the line numbers should be visible; `false` hides them.

Returns

Nothing.

dom.setShowRulers()

Availability

Dreamweaver 3.

Description

This function toggles the View >Rulers > Show option on and off.

Arguments

`bShowRulers`

- The *`bShowRulers`* argument is a Boolean value: `true` turns on the Show option; `false` turns it off.

Returns

Nothing.

dom.setShowSyntaxColoring()

Availability

Dreamweaver 4.

Description

This function turns syntax coloring on or off in the Code view of the document window.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: `true` indicates that syntax coloring should be visible; `false` otherwise.

Returns

Nothing.

dom.setShowTableBorders()

Availability

Dreamweaver 3.

Description

This function toggles the View > Table Borders option on and off.

Arguments

`bShowTableBorders`

- The *`bShowTableBorders`* argument is a Boolean value: `true` turns on the Table Borders option; `false` turns it off.

Returns

Nothing.

dom.setShowToolbar()

Availability

Dreamweaver 4.

Description

This function shows or hides the Toolbar.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: `true` indicates the toolbar should be visible; `false` otherwise.

Returns

Nothing.

dom.setShowTracingImage()

Availability

Dreamweaver 3.

Description

This function toggles the View > Tracing Image > Show option on and off.

Arguments

`bShowTracingImage`

- The *`bShowTracingImage`* argument is a Boolean value: `true` turns on the Show option; `false` turns it off.

Returns

Nothing.

dom.setShowWordWrap()

Availability

Dreamweaver 4.

Description

This function toggles the Word Wrap option off or on in the Code view of the document window.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: `true` indicates that the lines should wrap; `false` otherwise.

Returns

Nothing.

dom.setSnapToGrid()

Availability

Dreamweaver 3.

Description

This function toggles the View > Grid > Snap To option on or off.

Arguments

`bSnapToGrid`

- The *`bSnapToGrid`* argument is a Boolean value: `true` turns on the Snap To option; `false` turns it off.

Returns

Nothing.

dreamweaver.getHideAllFloaters()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Hide Panels option.

Arguments

None.

Returns

A Boolean value: `true` indicates whether the Hide Panels option is on; `false` indicates the Show Panels option is on.

dreamweaver.getShowStatusBar()

Availability

Dreamweaver 3.

Description

This function gets the current state of the View > Status Bar option.

Arguments

None.

Returns

A Boolean value: `true` indicates the status bar is visible; `false` otherwise.

dreamweaver.htmlInspector.getShowAutoIndent()

Availability

Dreamweaver 4.

Description

This function determines whether the Auto Indent option is on in the Code inspector.

Arguments

None.

Returns

A Boolean value: `true` if auto-indenting is on; `false` otherwise.

dreamweaver.htmlInspector.getShowInvalidHTML()

Availability

Dreamweaver 4.

Description

This function determines whether invalid HTML code is currently highlighted in the Code inspector.

Arguments

None.

Returns

A Boolean value: `true` if invalid HTML code is highlighted; `false` otherwise.

dreamweaver.htmlInspector.getShowLineNumbers()

Availability

Dreamweaver 4.

Description

This function determines whether line numbers appear in the Code inspector.

Arguments

None.

Returns

A Boolean value: `true` if line numbers appear; `false` otherwise.

dreamweaver.htmlInspector.getShowSyntaxColoring()

Availability

Dreamweaver 4.

Description

This function determines whether syntax coloring is on in the Code inspector.

Arguments

None.

Returns

A Boolean value: `true` if syntax coloring is on; `false` otherwise.

dreamweaver.htmlInspector.getShowWordWrap()

Availability

Dreamweaver 4.

Description

This function determines whether the Word Wrap is on in the Code inspector.

Arguments

None.

Returns

A Boolean value: `true` if word wrap is on; `false` otherwise.

dreamweaver.htmlInspector.setShowAutoIndent()

Availability

Dreamweaver 4.

Description

This function turns the Auto Indent option on or off in the Code inspector.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: true turns the auto-indenting on; `false` turns it off.

Returns

Nothing.

dreamweaver.htmlInspector.setShowInvalidHTML()

Availability

Dreamweaver 4.

Description

This function turns highlighting of invalid HTML code on or off in the Code inspector.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: `true` indicates that the highlighting of invalid HTML code should be visible; `false` indicates it should not.

Returns

Nothing.

dreamweaver.htmlInspector.setShowLineNumbers()

Availability

Dreamweaver 4.

Description

This function shows or hides the line numbers in the Code view of the Code inspector.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value: `true` sets the line numbers to visible; `false` hides them.

Returns

Nothing.

dreamweaver.htmlInspector.setShowSyntaxColoring()

Availability

Dreamweaver 4.

Description

This function turns syntax coloring on or off in the Code view of the Code inspector.

Arguments

`bShow`

- The `bShow` argument is a Boolean value: `true` indicates that the syntax coloring should be visible; `false` turns it off.

Returns

Nothing.

dreamweaver.htmlInspector.setShowWordWrap()

Availability

Dreamweaver 4.

Description

This function turns the Word Wrap option off or on in the Code inspector.

Arguments

`bShow`

- The `bShow` argument is a Boolean value: `true` turns Word Wrap on; `false` turns it off.

Returns

Nothing.

dreamweaver.setHideAllFloaters()

Availability

Dreamweaver 3.

Description

This function sets either the Hide Panels option or the Show Panels option.

Arguments

`bShowFloatingPalettes`

- The `bShowFloatingPalettes` argument is a Boolean value: `true` turns on the Hide Panels option; `false` turns on the Show Panels option.

Returns

Nothing.

dreamweaver.setShowStatusBar()

Availability

Dreamweaver 3.

Description

This function toggles the View > Status Bar option on or off.

Arguments

`bShowStatusBar`

- The *`bShowStatusBar`* argument is a Boolean value: `true` turns on the Status Bar option; `false` turns it off.

Returns

Nothing.

site.getShowDependents()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Show Dependent Files option.

Arguments

None.

Returns

A Boolean value: `true` indicates that dependent files are visible in the site map; `false` indicates dependent files are not visible.

site.getShowHiddenFiles()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Show Files Marked as Hidden option.

Arguments

None.

Returns

A Boolean value: `true` indicates that hidden files are visible in the site map; `false` otherwise.

site.getShowPageTitles()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Show Page Titles option.

Arguments

None.

Returns

A Boolean value: `true` indicates that the page titles are visible in the site map; `false` otherwise.

site.getShowToolTips()

Availability

Dreamweaver 3.

Description

This function gets the current state of the Tool Tips option.

Arguments

None.

Returns

A Boolean value: `true` indicates that the tool tips are visible in the Site panel; `false` otherwise.

site.setShowDependents()

Availability

Dreamweaver 3.

Description

This function toggles the Show Dependent Files option in the site map on or off.

Arguments

`bShowDependentFiles`

- The *`bShowDependentFiles`* argument is a Boolean value: `true` turns on the Show Dependent Files option; `false` turns it off.

Returns

Nothing.

site.setShowHiddenFiles()

Availability

Dreamweaver 3.

Description

This function toggles the Show Files Marked as Hidden option in the site map on or off.

Arguments

`bShowHiddenFiles`

- The *`bShowHiddenFiles`* argument is a Boolean value: `true` turns on the Show Files Marked as Hidden option; `false` turns it off.

Returns

Nothing.

site.setShowPageTitles()

Availability

Dreamweaver 3.

Description

This function toggles the Show Page Titles option in the site map on or off.

Arguments

`bShowPageTitles`

- The *`bShowPageTitles`* argument is a Boolean value: `true` turns on the Show Page Titles option; `false` turns it off.

Returns

Nothing.

Enabler

"site.canShowPageTitles()" on page 462.

site.setShowToolTips()

Availability

Dreamweaver 3.

Description

This function toggles the Tool Tips option on or off.

Arguments

`bShowToolTips`

- The *`bShowToolTips`* argument is a Boolean value: `true` turns on the Tool Tips option; `false` turns it off.

Returns

Nothing.

Toolbar functions

The following JavaScript functions let you get and set the visibility of toolbars and toolbar labels, obtain the labels of toolbar items in the current window, position toolbars, and obtain toolbar IDs. For more information on creating or modifying toolbars, see “Toolbars” in Extending Dreamweaver Help.

dom.forceToolbarUpdate()

Availability

Dreamweaver MX 2004.

Description

Forces the update handlers to run for all the items on the specified toolbar.

Arguments

toolbarID

- The *toolbarID* argument is the ID of the toolbar with the items Dreamweaver should update.

Returns

Nothing.

dom.getShowToolbarIconLabels()

Availability

Dreamweaver MX.

Description

This function determines whether labels for buttons are visible in the current document window. Dreamweaver always shows labels for non-button controls, if the labels are defined.

Arguments

None.

Returns

A Boolean value: `true` if labels for buttons are visible in the current document window; `false` otherwise.

Example

The following example makes labels for buttons visible:

```
var dom = dw.getDocumentDom();
if (dom.getShowToolbarIconLabels() == false)
{
  dom.setShowToolbarIconLabels(true);
}
```

dom.getToolbarIdArray()

Availability

Dreamweaver MX.

Description

This function returns an array of the IDs of all the toolbars in the application. You can use `dom.getToolbarIdArray()` to turn off all toolbars so you can reposition them and make only a specific set visible.

Arguments

None.

Returns

An array of all toolbar IDs.

Example

The following example stores the array of toolbar IDs in the `tb_ids` variable:

```
var tb_ids = new Array();
tb_ids = dom.getToolbarIdArray();
```

dom.getToolbarItemValue()

Availability

Dreamweaver MX 2004.

Description

Gets the value of the specified toolbar item.

Arguments

`toolbarID, itemID`

- The *`toolbarID`* argument is a string that specifies the ID of the toolbar that contains the item for which you want a value.
- The *`itemID`* argument is a string that specifies the ID of the item for which you want the value.

Returns

A string that represents the value of the toolbar item.

Example

The following example of `receiveArguments()` is in a toolbar command that controls the behavior of a Size text field; it gets the value of the Size field as an argument and then reads the value of the Units field in order to produce a valid value for the CSS property `font-size` function:

```
receiveArguments(newSize){
var dom = dw.getDocumentDOM();
if (newSize != ""){
  dom.applyFontMarkupAsStyle('font-size', newSize +
  dom.getToolbarItemValue("DW_Toolbar_Text","DW_Text_Units"));
  }
else{
   dom.removeFontMarkupAsStyle('font-size');
   }
}
```

dom.getToolbarLabel()

Availability

Dreamweaver MX.

Description

This function obtains the label of the specified toolbar. You can use `dom.getToolbarLabel()` for menus that show or hide toolbars.

Arguments

`toolbar_id`

- The *`toolbar_id`* argument is the ID of the toolbar, which is the value of the ID attribute on the toolbar tag in the toolbars.xml file.

Returns

The *`label`* name string that is assigned as an attribute on the `toolbar` tag.

Example

The following example stores the label for `myEditbar` in the variable `label`:

```
var label = dom.getToolbarLabel("myEditbar");
```

dom.getToolbarVisibility()

Availability

Dreamweaver MX.

Description

This function returns a Boolean value that indicates whether the toolbar that is specified by *`toolbar_id`* is visible.

Arguments

`toolbar_id`

- The *`toolbar_id`* argument is the ID string that is assigned to the toolbar.

Returns

A Boolean value: `true` if the toolbar is visible, `false` if the toolbar is not visible or does not exist.

Example

The following example checks whether the toolbar `myEditbar` is visible in the document window, and then stores that value in the `retval` variable:

```
var retval = dom.getToolbarVisibility("myEditbar");
return retval;
```

dom.setToolbarItemAttribute()

Availability

Dreamweaver MX 2004.

Description

Changes an attribute value for the three image attributes or the tooltip attribute on a toolbar item.

Arguments

toolbarID, toolbarItemId, attrName, attrValue

- The *toolbarID* argument is a string that specifies the ID of the toolbar.
- The *toolbarItemId* argument is a string that specifies the ID of the toolbar item.
- The *attrName* argument is a string that specifies the name of the attribute to set. Valid values are `'image'`, `'overImage'`, `'disabledImage'`, or `'tooltip'`.
- The *attrValue* argument is a string that specifies the value to set.

Returns

Nothing.

Example

The following example calls `dom.setToolbarItemAttribute()` three times to set the `image`, `imageOver`, and `tooltip` attributes for the toolbar item `MyButton` on the toolbar having the ID `DW_Toolbar_Main`:

```
var dom = dw.getDocumentDOM();
dom.setToolbarItemAttribute('DW_Toolbar_Main', 'MyButton, 'image', 'Toolbars/
   imgs/newimage.gif');
dom.setToolbarItemAttribute('DW_Toolbar_Main', 'MyButton, 'imageOver',
   'Toolbars/imgs/newimageOver.gif');
dom.setToolbarItemAttribute('DW_Toolbar_Main', 'MyButton, 'tooltip', 'One fine
   button');
```

dom.setShowToolbarIconLabels()

Availability

Dreamweaver MX.

Description

This function tells Dreamweaver to show the labels of buttons that have labels. Dreamweaver always shows labels for non-button controls, if the labels are defined.

Arguments

bShow

- The *bShow* argument is a Boolean value: `true` shows the labels for buttons; `false` otherwise.

Returns

Nothing.

Example

The following example tells Dreamweaver to show the labels for the buttons on the toolbars:

```
dom.setShowToolbarIconLabels(true);
```

dom.setToolbarPosition()

Availability

Dreamweaver MX.

Description

This function moves the specified toolbar to the specified position.

Note: There is no way to determine the current position of a toolbar.

Arguments

`toobar_id, position, relative_to`

- The *`toolbar_id`* argument is the ID of the toolbar, which is the value of the ID attribute on the toolbar tag in the toolbars.xml file.
- The *`position`* argument specifies where Dreamweaver positions the toolbar, relative to other toolbars. The possible values for *`position`* are described in the following list:
  - `top` is the default position. The toolbar appears at the top of the document window.
  - `below` makes the toolbar appear at the beginning of the row immediately below the toolbar that *`relative_to`* specifies. Dreamweaver reports an error if the toolbar does not find the toolbar that *`relative_to`* specifies.
  - `floating` makes the toolbar float above the document. Dreamweaver automatically places the toolbar so it is offset from other floating toolbars. On the Macintosh, `floating` is treated the same way as `top`.
- *`relative_to`*="*`toolbar_id`*" is required if *`position`* specifies `below`. Otherwise, it is ignored. Specifies the ID of the toolbar below which this toolbar should be positioned.

Returns

Nothing.

Example

The following example sets the position of myEditbar below the myPicturebar toolbar:

```
dom.setToolbarPosition("myEditbar", "below", "myPicturebar");
```

dom.setToolbarVisibility()

Availability

Dreamweaver MX.

Description

This function shows or hides the specified toolbar.

Arguments

toolbar_id, bShow

- The *toolbar_id* argument is the ID of the toolbar, the value of the ID attribute on the toolbar tag in the toolbars.xml file.
- The *bShow* argument is a Boolean value that indicates whether to show or hide the toolbar. If *bshow* is `true`, `dom.setToolbarVisibility()` makes the toolbar visible. If *bShow* is `false`, `dom.setToolbarVisibility()` makes the toolbar invisible.

Returns

Nothing.

Example

The following example checks to see if the toolbar myEditbar is visible in the document window; if it is not visible, it sets myEditbar to be visible:

```
var dom = dw.getDocumentDOM();
if(dom != null && dom.getToolbarVisibility("myEditbar") == false)
{
  dom.setToolbarVisibility("myEditbar", true);
{
```

Window functions

Window functions handle operations that are related to the document window and the floating panels. The window functions show and hide floating panels, determine which part of the Document window has focus, and set the active document. For operations that are related specifically to the Site panel, see "Site functions" on page 220.

Note: Some of the functions in this section operate only on Windows. The description of a function indicates whether this is the case.

dom.getFocus()

Availability

Dreamweaver 3.

Description

This function determines the part of the document that is currently in focus.

Arguments

None.

Returns

One of the following strings:

- The `"head"` string if the `HEAD` area is active
- The `"body"` string if the `BODY` or `NOFRAMES` area is active
- The `"frameset"` string if a frameset or any of its frames is selected
- The `"none"` string if the focus is not in the document (for example, if it's in the Property inspector or another floating panel)

dom.getView()

Availability

Dreamweaver 4.

Description

This function determines which view is visible.

Arguments

None.

Returns

`"design"`, `"code"`, or `"split"`, depending on the visible view.

dom.getWindowTitle()

Availability

Dreamweaver 3.

Description

This function gets the title of the window that contains the document.

Arguments

None.

Returns

A string that contains the text that appears between the `TITLE` tags in the document, or nothing, if the document is not in an open window.

dom.setView()

Availability

Dreamweaver 4.

Description

This function shows or hides the Design or Code view to produce a design-only, code-only, or split view.

Arguments

`viewString`

- The *`viewString`* argument is the view to produce; it must be one of the following values: `"design"`, `"code"`, or `"split"`.

Returns

Nothing.

dreamweaver.bringAttentionToFloater()

Availability

Dreamweaver MX.

Description

Brings the specified panel or inspector to the front, and draws attention to the panel or inspector by making it flash, which is slightly different functionality than `dw.toggleFloater()`.

Arguments

`floaterName`

- The *`floaterName`* argument is the name of the window, panel, or inspector.

Returns

Nothing.

Example

The following example opens and flashes the Assets panel:

```
dw.bringAttentionToFloater("library");
```

dreamweaver.cascade()

Availability

Dreamweaver MX (Windows only).

Description

Cascades the document windows, starting in the upper left corner and positioning each window below and slightly offset from the previous one.

Arguments

None.

Returns

Nothing.

Example

The following example cascades the open documents:

```
dw.cascade()
```

dreamweaver.getActiveWindow()

Availability

Dreamweaver 3.

Description

This function gets the document in the active window.

Arguments

None.

Returns

The document object that corresponds to the document in the active window; or, if the document is in a frame, the document object that corresponds to the frameset.

dreamweaver.getDocumentList()

Availability

Dreamweaver 3.

Description

This function gets a list of all the open documents.

Arguments

None.

Returns

An array of document objects, each corresponding to an open Document window. If a document window contains a frameset, the document object refers to the frameset, not the contents of the frames.

dreamweaver.getFloaterVisibility()

Availability

Dreamweaver 3.

Description

This function checks whether the specified panel or inspector is visible.

Arguments

`floaterName`

- The `floaterName` argument is the name of a floating panel. If `floaterName` does not match one of the built-in panel names, Dreamweaver searches in the Configuration/Floaters folder for a file called `floaterName.htm` where `floaterName` is the name of a floating panel.

The *`floaterName`* values for built-in Dreamweaver panels are the strings to the right of the panel names in the following list:

Assets = `"assets"`

Behaviors = `"behaviors"`

Code inspector = `"html"`

Components = `"server components"`

CSS Styles = `"css styles"`

Bindings = `"data sources"`

Frames = `"frames"`

History = `"history"`

HTML Styles = `"html styles"`

Insert bar = `"objects"`

Layers = `"layers"`

Library = `"library"`

Link Checker Results = `"linkchecker"`

Properties = `"properties"`

Reference = `"reference"`

Report Results = `"reports"`

Search Results = `"search"`

Server Behaviors = `"server behaviors"`

Site = `"site"`

Site Files = `"site files"`

Site Map - “`site map`”

Snippets = `"snippets"`

Tag inspector = `"tag inspector"`

Target Browser Check Results = `"btc"`

Templates = `"templates"`

Validation Results = `"validation"`

Returns

A Boolean value: `true` if the floating panel is visible and in the front; `false` otherwise or if Dreamweaver cannot find a floating panel named *`floaterName`*.

dreamweaver.getFocus()

Availability

Dreamweaver 4.

Description

This function determines what part of the application is currently in focus.

Arguments

`bAllowFloaters`

- The `bAllowFloaters` argument is a Boolean value: `true` if you want the function to return the name of the floating panel, if a floating panel has focus; `false` otherwise.

Returns

One of the following strings:

- The `"document"` string if the document window is in focus
- The `"site"` string if the Site panel is in focus
- The `"textView"` string if the Text view is in focus
- The `"html"` string if the Code inspector is in focus
- The `floaterName` string, if `bAllowFloaters` is `true` and a floating panel has focus, where `floaterName` is `"objects"`, `"properties"`, `"launcher"`, `"library"`, `"css styles"`, `"html styles"`, `"behaviors"`, `"timelines"`, `"layers"`, `"frames"`, `"templates"`, or `"history"`
- (Macintosh) The `"none"` string if neither the Site panel nor any document windows are open

dreamweaver.getPrimaryView()

Availability

Dreamweaver 4.

Description

This function determines which view is visible as the primary view in the front.

Arguments

None.

Returns

The `"design"` or `"code"` strings, depending on which view is visible or on the top in a split view.

dreamweaver.getSnapDistance()

Availability

Dreamweaver 4.

Description

This function returns the snapping distance in pixels.

Arguments

None.

Returns

An integer that represents the snapping distance in pixels. The default is 10 pixels; 0 indicates that the Snap feature is off.

dreamweaver.minimizeRestoreAll()

Availability

Dreamweaver 4.

Description

This function minimizes (reduces to an icon) or restores all windows in Dreamweaver.

Arguments

`bMinimize`

- The *`bMinimize`* argument is a Boolean value: `true` if windows should be minimized; `false` if the minimized windows should be restored.

Returns

Nothing.

dreamweaver.setActiveWindow()

Availability

Dreamweaver 3.

Description

This function activates the window that contains the specified document.

Arguments

`documentObject`, *`{bActivateFrame}`*

- The *`documentObject`* argument is the object at the root of a document's DOM tree (the value that the `dreamweaver.getDocumentDOM()` function returns).
- The *`bActivateFrame`* argument is optional, and is applicable only if *`documentObject`* is inside a frameset. The *`bActivateFrame`* argument is a Boolean value: `true` activates the frame that contains the document as well as the window that contains the frameset; `false` otherwise.

Returns

Nothing.

dreamweaver.setFloaterVisibility()

Availability

Dreamweaver 3.

Description

This function specifies whether to make a particular floating panel or inspector visible.

Arguments

`floaterName`, `bIsVisible`

- The `floaterName` argument is the name of a floating panel. If `floaterName` does not match one of the built-in panel names, Dreamweaver searches in the Configuration/Floaters folder for a file called `floaterName.htm` where `floaterName` is the name of a floating panel. If Dreamweaver cannot find a floating panel named `floaterName`, this function has no effect.

  The `floaterName` values for built-in Dreamweaver panels are the strings to the right of the panel names in the following list:

  Assets = `"assets"`

  Behaviors = `"behaviors"`

  Code inspector = `"html"`

  Components = `"server components"`

  CSS Styles = `"css styles"`

  Bindings = `"data sources"`

  Frames = `"frames"`

  History = `"history"`

  HTML Styles = `"html styles"`

  Insert bar = `"objects"`

  Layers = `"layers"`

  Library = `"library"`

  Link Checker Results = `"linkchecker"`

  Properties = `"properties"`

  Reference = `"reference"`

  Report Results = `"reports"`

  Search Results = `"search"`

  Server Behaviors = `"server behaviors"`

  Site = `"site"`

  Site Files = `"site files"`

  Site Map = `"site map"`

  Snippets = `"snippets"`

Tag inspector = `"tag inspector"`	
Target Browser Check Results = `"btc"`	
Templates = `"templates"`	
Validation Results = `"validation"`	

The *`bIsVisible`* argument is a Boolean value that indicates whether to make the floating panel visible.

Returns

Nothing.

dreamweaver.setPrimaryView()

Availability

Dreamweaver 4.

Description

This function displays the specified view at the top of the document window.

Arguments

`viewString`

- The *`viewString`* argument is the view to display at the top of the document window; it can be one of the following values: `"design"` or `"code"`.

Returns

Nothing.

dreamweaver.setSnapDistance()

Availability

Dreamweaver 4.

Description

This function sets the snapping distance in pixels.

Arguments

`snapDistance`

- The *`snapDistance`* argument is an integer that represents the snapping distance in pixels. The default is 10 pixels. Specify 0 to turn off the Snap feature.

Returns

Nothing.

dreamweaver.showProperties()

Availability

Dreamweaver 3.

Description

This function makes the Property inspector visible and gives it focus.

Arguments

None.

Returns

Nothing.

dreamweaver.tileHorizontally()

Availability

Dreamweaver MX (Windows only).

Description

Tiles the document windows horizontally, positioning each window next to another one without overlapping the documents. This process is similar to splitting the workspace vertically.

Arguments

None.

Returns

Nothing.

Example

The following example tiles the open documents horizontally:

```
dw.tileHorizontally()
```

dreamweaver.tileVertically()

Availability

Dreamweaver MX (Windows only).

Description

Tiles the document window vertically, positioning one document window behind the other without overlapping documents. This is similar to splitting the workspace horizontally.

Arguments

None.

Returns

Nothing.

Example

The following example tiles the open documents vertically:

```
dw.tileVertically()
```

dreamweaver.toggleFloater()

Availability

Dreamweaver 3.

Description

This function shows, hides, or brings to the front the specified panel or inspector.

Note: This function is meaningful only in the menus.xml file. To show, bring forward, or hide a floating panel, use `dw.setFloaterVisibility()`.

Arguments

`floaterName`

- The *`floaterName`* argument is the name of the window. If the floating panel name is `reference`, the visible/invisible state of the Reference panel is updated by the user's selection in Code view. All other panels track the selection all the time, but the Reference panel tracks the selection in Code view only when the user invokes tracking.

Returns

Nothing.

dreamweaver.updateReference()

Availability

Dreamweaver 4.

Description

This function updates the Reference floating panel. If the Reference floating panel is not visible, `dw.updateReference()` makes it visible and then updates it.

Arguments

None.

Returns

Nothing.

CHAPTER 13
Site

Site functions perform operations related to managing a website. These operations include customizing a report, defining a new site, checking in and checking out files, running validation on a site and so on.

Report functions

Report functions provide access to the Macromedia Dreamweaver MX 2004 reporting features so you can initiate, monitor, and customize the reporting process. For more information, see "Reports" in Extending Dreamweaver Help.

dreamweaver.isReporting()

Availability

Dreamweaver 4.

Description

Checks to see if a reporting process is currently running.

Arguments

None.

Returns

A Boolean value: `true` if a process is running; `false` otherwise.

dreamweaver.showReportsDialog()

Availability

Dreamweaver 4.

Description

Opens the Reports dialog box.

Arguments

None.

Returns

Nothing.

Site functions

Site functions handle operations that are related to files in the site files or site map. These functions let you perform the following tasks:

- Create links between files
- Get, put, check in, and check out files
- Select and deselect files
- Create and remove files
- Get information about the sites that the user has defined
- Import and export site information

dreamweaver.loadSitesFromPrefs()

Availability

Dreamweaver 4.

Description

Loads the site information for all the sites from the system registry (Windows) or the Dreamweaver Preferences file (Macintosh) into Dreamweaver. If a site is connected to a remote server when this function is called, the site is automatically disconnected.

Arguments

None.

Returns

Nothing.

dreamweaver.saveSitesToPrefs()

Availability

Dreamweaver 4.

Description

Saves all information for each site that the user has defined to the system registry (Windows) or the Dreamweaver Preferences file (Macintosh).

Arguments

None.

Returns

Nothing.

site.addLinkToExistingFile()

Availability

Dreamweaver 3.

Description

Opens the Select HTML File dialog box to let the user select a file and creates a link from the selected document to that file.

Arguments

None.

Returns

Nothing.

Enabler

"site.canAddLink()" on page 455.

site.addLinkToNewFile()

Availability

Dreamweaver 3.

Description

Opens the Link to New File dialog box to let the user specify details for the new file and creates a link from the selected document to that file.

Arguments

None.

Returns

Nothing.

Enabler

"site.canAddLink()" on page 455.

site.canEditColumns()

Description

Determines whether a site exists.

Arguments

None.

Returns

A Boolean value: `true` if a site exists; `false` otherwise.

site.changeLinkSitewide()

Availability

Dreamweaver 3.

Description

Opens the Change Link Sitewide dialog box.

Arguments

None.

Returns

Nothing.

site.changeLink()

Availability

Dreamweaver 3.

Description

Opens the Select HTML File dialog box to let the user select a new file for the link.

Arguments

None.

Returns

Nothing.

Enabler

"site.canChangeLink()" on page 456.

site.checkIn()

Availability

Dreamweaver 3.

Description

Checks in the selected files and handles dependent files in one of the following ways:

- If the user selects Prompt on Put/Check In in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked Yes, dependent files are uploaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked No, dependent files are not uploaded and no dialog box appears.

Arguments

`siteOrURL`

- The `siteOrURL` argument must be the keyword `"site"`, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

Nothing.

Enabler

"site.canCheckIn()" on page 456.

site.checkLinks()

Availability

Dreamweaver 3.

Description

Opens the Link Checker dialog box and checks links in the specified files.

Arguments

`scopeOfCheck`

- The `scopeOfCheck` argument specifies the scope of the link checking. The value must be `"document"`, `"selection"`, or `"site"`.

Returns

Nothing.

site.checkOut()

Availability

Dreamweaver 3.

Description

Checks out the selected files and handles dependent files in one of the following ways:

- If the user selects Prompt on Get/Check Out in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked Yes, dependent files are downloaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked No, dependent files are not downloaded and no dialog box appears.

Arguments

`siteOrURL`

- The `siteOrURL` argument must be the keyword `"site"`, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

Nothing.

Enabler

"site.canCheckOut()" on page 456.

site.checkTargetBrowsers()

Availability

Dreamweaver 3.

Description

Runs a target browser check on the selected files.

Arguments

None.

Returns

Nothing.

site.cloak()

Availability

Dreamweaver MX.

Description

Cloaks the current selection in the Site panel or the specified folder.

Arguments

`siteOrURL`

The *`siteOrURL`* argument must contain one of the following two values:

- The keyword `"site"`, which indicates that `cloak()` should act on the selection in the Site panel
- The URL of a particular folder, which indicates that `cloak()` should act on the specified folder and all its contents

Returns

Nothing.

Enabler

"site.canCloak()" on page 457.

site.defineSites()

Availability

Dreamweaver 3.

Description

This function opens the Edit Sites dialog box.

Arguments

None.

Returns

Nothing.

site.deleteSelection()

Availability

Dreamweaver 3.

Description

Deletes the selected files.

Arguments

None.

Returns

Nothing.

site.deployFilesToTestingServerBin()

Availability

Dreamweaver MX.

Description

Puts a specified file (or files) in the testing server's bin folder. If the current site does not have any settings defined for deploying supporting files, this function invokes the Deploy Supporting Files To Testing Server dialog box.

Arguments

`filesToDeploy`

- The *`filesToDeploy`* argument is an array of filenames that Dreamweaver will deploy.

Returns

A Boolean value: `true` if the files deploy successfully; `false` otherwise.

Example

This example deploys the files image1.jpg and script1.js to the testing server's bin folder:

```
site.deployFilesToTestingServerBin("image1.jpg", "script1.js");
```

site.editColumns()

Description

This function displays the Edit Sites dialog box, which shows the File View Columns section.

Arguments

None.

Returns

Nothing.

site.exportSite()

Availability

Dreamweaver MX.

Description

Exports a Dreamweaver site to an XML file, which can be imported into another Dreamweaver instance to duplicate the former site.

All the information that is contained in the Site Definition dialog box is saved in an XML file that includes the list of cloaked folders and information about the default document type. The exception is that the user can omit the user login and password when FTP access is set. The following example shows a sample XML file that Dreamweaver creates when you export a site:

```
<?xml version="1.0" ?>
<site>
  <localinfo
    sitename="DW00"
    localroot="C:\Documents and Settings\jlondon\Desktop\DWServer\"
    imagefolder="C:\Documents and Settings\jlondon\Desktop\DWServer\Images\"
    spacerfilepath=""
    refreshlocal="TRUE"
    cache="FALSE"
    httpaddress="http://" curserver="webserver" />
  <remoteinfo
    accesstype="ftp"
    host="dreamweaver"
    remoteroot="kojak/"
    user="dream"
    checkoutname="Jay"
    emailaddress="jay@macromedia.com"
    usefirewall="FALSE"
    usepasv="TRUE"
    enablecheckin="TRUE"
    checkoutwhenopen="TRUE" />
  <designnotes
    usedesignnotes="TRUE"
    sharedesignnotes="TRUE" />
  <sitemap
    homepage="C:\Documents and Settings\jlondon\Desktop\DWServer\Untitled-2.htm"
    pagesperrow="200" columnwidth="125" showdependentfiles="TRUE"
    showpagetitles="FALSE" showhiddenfiles="TRUE" />
  <fileviewcolumns sharecolumns="TRUE">
    <column name="Local Folder"
      align="left" show="TRUE" share="FALSE" builtin="TRUE"
```

```
        localwidth="180" remotewidth="180" />
      <column name="Notes"
        align="center" show="TRUE" share="FALSE" builtin="TRUE"
        localwidth="36" remotewidth="36" />
      <column name="Size"
        align="right" show="TRUE" share="FALSE" builtin="TRUE"
        localwidth="-2" remotewidth="-2" />
      <column name="Type"
        align="left" show="TRUE" share="FALSE" builtin="TRUE"
        localwidth="60" remotewidth="60" />
      <column name="Modified"
        align="left" show="TRUE" share="FALSE" builtin="TRUE"
        localwidth="102" remotewidth="102" />
      <column name="Checked Out By"
        align="left" show="TRUE" share="FALSE" builtin="TRUE"
        localwidth="50" remotewidth="50" />
      <column name="Status" note="status"
        align="left" show="TRUE" share="FALSE" builtin="FALSE"
        localwidth="50" remotewidth="50" />
    </fileviewcolumns>
    <appserverinfo
      servermodel="ColdFusion"
      urlprefix="http://dreamweaver/kojak/"
      serverscripting="CFML"
      serverpageext=""
      connectionsmigrated="TRUE"
      useUD4andUD5pages="TRUE"
      defaultdoctype=""
      accesstype="ftp"
      host="dreamweaver"
      remoteroot="kojak/"
      user="dream"
      usefirewall="FALSE"
      usepasv="TRUE" />
    <cloaking enabled="TRUE" patterns="TRUE">
      <cloakedfolder folder="databases/" />
      <cloakedpattern pattern=".png" />
      <cloakedpattern pattern=".jpg" />
      <cloakedpattern pattern=".jpeg" />
    </cloaking>
  </site>
```

Arguments

`siteName`

- The `siteName` argument identifies the site to export. If `siteName` is an empty string, Dreamweaver exports the current site.

Returns

A Boolean value: `true` if the named site exists and if the XML file is successfully exported; `false` otherwise.

site.findLinkSource()

Availability

Dreamweaver 3.

Description

Opens the file that contains the selected link or dependent file, and highlights the text of the link or the reference to the dependent file. This function operates only on files in the Site Map view.

Arguments

None.

Returns

Nothing.

Enabler

"site.canFindLinkSource()" on page 458.

site.get()

Availability

Dreamweaver 3.

Description

Gets the specified files and handles dependent files in one of the following ways:

- If the user selects Prompt on Get/Check Out in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked Yes, dependent files are downloaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked No, dependent files are not downloaded and no dialog box appears.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the keyword `"site"`, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

Nothing.

Enabler

"site.canGet()" on page 458.

site.getAppServerAccessType()

Availability

Dreamweaver MX.

Description

Returns the access method that is used for all files on the current site's application server. The current site is the site that is associated with the document that currently has focus. If no document has focus, the site that you opened in Dreamweaver is used.

Note: ColdFusion Component Explorer uses this function; see site.getAppServerPathToFiles() and site.getLocalPathToFiles().

Arguments

None.

Returns

One of the following strings:

- `"none"`
- `"local/network"`
- `"ftp"`
- `"source_control"`

site.getAppServerPathToFiles()

Availability

Dreamweaver MX.

Description

Determines the path to the remote files on the application server that is defined for the current site. The current site is the site that is associated with the document that currently has focus. If no document has focus, the site that you opened in Dreamweaver is used.

Note: ColdFusion Component Explorer uses this function; see site.getAppServerAccessType() and site.getLocalPathToFiles().

Arguments

None.

Returns

If the access type to the application server file is `local/network`, this function returns a path; otherwise, this function returns an empty string.

site.getAppURLPrefixForSite()

Availability

Dreamweaver MX.

Description

Gets the value of the URL prefix of the currently selected site.

Arguments

None.

Returns

A string that contains the URL prefix of the currently selected site.

Example

```
sitePrefix = getAppURLPrefixForSite();
```

site.getCheckOutUser()

Availability

Dreamweaver 3.

Description

Gets the login and check-out name that is associated with the current site.

Arguments

None.

Returns

A string that contains a login and check-out name, if defined, or an empty string if Check In/ Check Out is disabled.

Example

A call to `site.getCheckOutUser()` might return `"denise (deniseLaptop)"`. If no check-out name is specified, only the login name returns (for example, `"denise"`).

site.getCheckOutUserForFile()

Availability

Dreamweaver 3.

Description

Gets the login and check-out name of the user who has the specified file checked out.

Arguments

`fileName`

- The *`fileName`* argument is the path to the file being queried, which is expressed as a file:// URL.

Returns

A string that contains the login and check-out name of the user who has the file checked out or an empty string if the file is not checked out.

Example

A call to `site.getCheckOutUserForFile("file://C:/sites/avocado8/index.html")` might return `"denise (deniseLaptop)"`. If no check-out name is specified, only the login name returns (for example, `"denise"`).

site.getCloakingEnabled()

Availability

Dreamweaver MX.

Description

Determines whether cloaking is enabled for the current site.

Arguments

None.

Returns

A Boolean value: `true` if cloaking is enabled for the current site; `false` otherwise.

site.getConnectionState()

Availability

Dreamweaver 3.

Description

Gets the current connection state.

Arguments

None.

Returns

A Boolean value that indicates whether the remote site is connected.

Enabler

"site.canConnect()" on page 457.

site.getCurrentSite()

Availability

Dreamweaver 3.

Description

Gets the current site.

Arguments

None.

Returns

A string that contains the name of the current site.

Example

If you defined several sites, a call to `site.getCurrentSite()` returns the one that is currently showing in the Current Sites List in the Site panel.

site.getFocus()

Availability

Dreamweaver 3.

Description

Determines which pane of the Site panel has focus.

Arguments

None.

Returns

One of the following strings:

- `"local"`
- `"remote"`
- `"site map"`

site.getLinkVisibility()

Availability

Dreamweaver 3.

Description

Checks whether all the selected links in the site map are visible (that is, not marked hidden).

Arguments

None.

Returns

A Boolean value: `true` if all the selected links are visible; `false` otherwise.

site.getLocalPathToFiles()

Availability

Dreamweaver MX.

Description

Determines the path to the local files that are defined for the current site. The current site is the site that is associated with the document that currently has focus. If no document has focus, the site that you opened in Dreamweaver is used.

Note: ColdFusion Component Explorer uses this function; see site.getAppServerAccessType() and site.getAppServerPathToFiles().

Arguments

None.

Returns

The path to the files residing on the local computer for the current site.

site.getSelection()

Availability

Dreamweaver 3.

Description

Determines which files are currently selected in the Site panel.

Arguments

None.

Returns

An array of strings that represents the paths of the selected files and folders, which is expressed as a file:// URL or an empty array if no files or folders are selected.

site.getSiteForURL()

Availability

Dreamweaver MX.

Description

Gets the name of the site, if any, that is associated with a specific file.

Arguments

`fileURL`

- The *`fileURL`* argument is the fully qualified URL (including the string "*`file://`*") for a named file.

Returns

A string that contains the name of the site, if any, in which the specified file exists. The string is empty when the specified file does not exist in any defined site.

site.getSites()

Availability

Dreamweaver 3.

Description

Gets a list of the defined sites.

Arguments

None.

Returns

An array of strings that represents the names of the defined sites, or an empty array if no sites are defined.

site.importSite()

Availability

Dreamweaver MX.

Description

Creates a Dreamweaver site from an XML file. During import, if the folder that is specified by the `localroot` attribute of the `<localinfo>` element does not exist on the local computer, Dreamweaver prompts for a different local root folder. Dreamweaver behaves the same way when it tries to locate the default images folder that is specified by the `imagefolder` attribute of the `<localinfo>` element.

Arguments

`fileURL`

- The *`fileURL`* argument is a string that contains the URL for the XML file. Dreamweaver uses this XML file to create a new site. If `fileURL` is an empty string, Dreamweaver prompts the user to select an XML file to import.

Returns

A Boolean value: `true` if the named XML file exists and if the site is created successfully; `false` otherwise.

site.invertSelection()

Availability

Dreamweaver 3.

Description

Inverts the selection in the site map.

Arguments

None.

Returns

Nothing.

site.isCloaked()

Availability

Dreamweaver MX.

Description

Determines whether the current selection in the Site panel or the specified folder is cloaked.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must contain one of the following two values:
  - The keyword `"site"`, which indicates that the `isCloaked()` function should test the selection in the Site panel
  - The file URL of a particular folder, which indicates that `isCloaked()` should test the specified folder

Returns

A Boolean value: `true` if the specified object is cloaked; `false` otherwise.

site.locateInSite()

Availability

Dreamweaver 3.

Description

Locates the specified file (or files) in the specified pane of the Site panel and selects the files.

Arguments

`localOrRemote`, *`siteOrURL`*

- The *`localOrRemote`* argument must be either `"local"` or `"remote"`.
- The *`siteOrURL`* argument must be the keyword `"site"`, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

Nothing.

Enabler

"site.canLocateInSite()" on page 458.

site.makeEditable()

Availability

Dreamweaver 3.

Description

Turns off the read-only flag on the selected files.

Arguments

None.

Returns

Nothing.

Enabler

"site.canMakeEditable()" on page 459.

site.makeNewDreamweaverFile()

Availability

Dreamweaver 3.

Description

Creates a new Dreamweaver file in the Site panel in the same folder as the first selected file or folder.

Arguments

None.

Returns

Nothing.

Enabler

"site.canMakeNewFileOrFolder()" on page 459.

site.makeNewFolder()

Availability

Dreamweaver 3.

Description

Creates a new folder in the Site panel in the same folder as the first selected file or folder.

Arguments

None.

Returns

Nothing.

Enabler

"site.canMakeNewFileOrFolder()" on page 459.

site.newHomePage()

Availability

Dreamweaver 3.

Description

Opens the New Home Page dialog box to let the user create a new home page.

Note: This function operates only on files in the Site Map view.

Arguments

None.

Returns

Nothing.

site.newSite()

Availability

Dreamweaver 3.

Description

Opens the Site Definition dialog box for a new, unnamed site.

Arguments

None.

Returns

Nothing.

site.open()

Availability

Dreamweaver 3.

Description

Opens the files that are currently selected in the Site panel. If any folders are selected, they are expanded in the Site Files view.

Arguments

None.

Returns

Nothing.

Enabler

"site.canOpen()" on page 460.

site.put()

Availability

Dreamweaver 3.

Description

Puts the selected files and handles dependent files in one of the following ways:

- If the user selects Prompt on Put/Check In in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked Yes, dependent files are uploaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and clicked No, dependent files are not uploaded and no dialog box appears.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the keyword `"site"`, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

Nothing.

Enabler

"site.canPut()" on page 460.

site.recreateCache()

Availability

Dreamweaver 3.

Description

Re-creates the cache for the current site.

Arguments

None.

Returns

Nothing.

Enabler

"site.canRecreateCache()" on page 460.

site.refresh()

Availability

Dreamweaver 3.

Description

Refreshes the file listing on the specified side of the Site panel.

Arguments

`whichSide`

- The *`whichSide`* argument must be `"local"`, or `"remote"`. If the site map has focus and *`whichSide`* is `"local"`, the site map refreshes.

Returns

Nothing.

Enabler

"site.canRefresh()" on page 461.

site.remoteIsValid()

Availability

Dreamweaver 3.

Description

Determines whether the remote site is valid.

Arguments

None.

Returns

A Boolean value that indicates whether a remote site has been defined and, if the server type is Local/Network, whether the drive is mounted.

site.removeLink()

Availability

Dreamweaver 3.

Description

Removes the selected link from the document above it in the site map.

Arguments

None.

Returns

Nothing.

Enabler

"site.canRemoveLink()" on page 461.

site.renameSelection()

Availability

Dreamweaver 3.

Description

Turns the name of the selected file into an text field, so the user can rename the file. If more than one file is selected, this function acts on the last selected file.

Arguments

None.

Returns

Nothing.

site.runValidation()

Availability

Dreamweaver MX.

Description

Runs the Validator on the entire site or only highlighted items.

Arguments

`selection`

- The *`selection`* argument is the parameter that specifies that the Validator should check only the highlighted items; otherwise, the Validator checks the entire current site.

Returns

Nothing.

site.saveAsImage()

Availability

Dreamweaver 3.

Description

Opens the Save As dialog box to let the user save the site map as an image.

Arguments

`fileType`

- The `fileType` argument is the type of image that should be saved. Valid values for Windows are `"bmp"` and `"png"`; valid values for the Macintosh are `"pict"` and `"jpeg"`. If the argument is omitted, or if the value is not valid on the current platform, the default is `"bmp"` in Windows and `"pict"` on the Macintosh.

Returns

Nothing.

site.selectAll()

Availability

Dreamweaver 3.

Description

Selects all files in the active view (either the site map or the site files).

Arguments

None.

Returns

Nothing.

site.selectHomePage()

Availability

Dreamweaver 3.

Description

Opens the Open File dialog box to let the user select a new home page.

Note: This function operates only on files in the Site Map view.

Arguments

None.

Returns

Nothing.

site.selectNewer()

Availability

Dreamweaver 3.

Description

Selects all files that are newer on the specified side of the Site panel.

Arguments

`whichSide`

- The `whichSide` argument must be either `"local"` or `"remote"`.

Returns

Nothing.

Enabler

"site.canSelectNewer()" on page 462.

site.setAsHomePage()

Availability

Dreamweaver 3.

Description

Designates the file that is selected in the Site Files view as the home page for the site.

Arguments

None.

Returns

Nothing.

site.setCloakingEnabled()

Availability

Dreamweaver MX.

Description

Determines whether cloaking should be enabled for the current site.

Arguments

`enable`

- The `enable` argument is a Boolean value that indicates whether cloaking should be enabled. A value of `true` enables cloaking for the current site; a value of `false` disables cloaking for the current site.

Returns

None.

site.setConnectionState()

Availability

Dreamweaver 3.

Description

Sets the connection state of the current site.

Arguments

`bConnected`

- The *`bConnected`* argument is a Boolean value that indicates if there is a connection (`true`) or not (`false`) to the current site.

Returns

Nothing.

site.setCurrentSite()

Availability

Dreamweaver 3.

Description

Opens the specified site in the local pane of the Site panel.

Arguments

`whichSite`

- The *`whichSite`* argument is the name of a defined site (as it appears in the Current Sites list in the Site panel or the Edit Sites dialog box).

Returns

Nothing.

Example

If three sites are defined (for example, avocado8, dreamcentral, and testsite), a call to `site.setCurrentSite("dreamcentral");` makes dreamcentral the current site.

site.setFocus()

Availability

Dreamweaver 3.

Description

Gives focus to a specified pane in the Site panel. If the specified pane is not showing, this function displays the pane and gives it focus.

Arguments

`whichPane`

- The *`whichPane`* argument must be one of the following strings: `"local"`, `"remote"`, or `"site map"`.

Returns

Nothing.

site.setLayout()

Availability

Dreamweaver 3.

Description

Opens the Site Map Layout pane in the Site Definition dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"site.canSetLayout()" on page 461.

site.setLinkVisibility()

Availability

Dreamweaver 3.

Description

Shows or hides the current link.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value that indicates whether to remove the Hidden designation from the current link.

Returns

Nothing.

site.setSelection()

Availability

Dreamweaver 3.

Description

Selects files or folders in the active pane in the Site panel.

Arguments

`arrayOfURLs`

- The `arrayOfURLs` argument is an array of strings where each string is a path to a file or folder in the current site, which is expressed as a file:// URL.

Note: Omit the trailing slash (/) when specifying folder paths.

Returns

Nothing.

site.synchronize()

Availability

Dreamweaver 3.

Description

Opens the Synchronize Files dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"site.canSynchronize()" on page 463.

site.uncloak()

Availability

Dreamweaver MX.

Description

Uncloaks the current selection in the Site panel or the specified folder.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must contain one of the following values:
  - The keyword `"site"`, which indicates that the `unCloak()` function should act on the selection in the Site panel
  - The URL of a particular folder, which indicates that the `unCloak()` function should act on the specified folder and all its contents

Returns

Nothing.

Enabler

"site.canUncloak()" on page 463.

site.uncloakAll()

Availability

Dreamweaver MX.

Description

Uncloaks all folders in the current site and deselects the Cloak Files Ending With: checkbox in the Cloaking settings.

Arguments

Nothing.

Returns

Nothing.

Enabler

"site.canUncloak()" on page 463.

site.undoCheckOut()

Availability

Dreamweaver 3.

Description

Removes the lock files that are associated with the specified files from the local and remote sites, and replaces the local copy of the specified files with the remote copy.

Arguments

siteOrURL

- The *siteOrURL* argument must be the keyword `"site"`, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

Nothing.

Enabler

"site.canUndoCheckOut()" on page 463.

site.viewAsRoot()

Availability

Dreamweaver 3.

Description

Temporarily moves the selected file to the top position in the site map.

Arguments

None.

Returns

Nothing.

Enabler

"site.canViewAsRoot()" on page 464.

CHAPTER 14
Document

The Document functions in Macromedia Dreamweaver MX 2004 perform operations that affect the document on which the user is working. These functions perform tasks that convert tables to layers, run a command in the Configuration/Commands folder, browse for a file URL, check spelling or set page properties, convert a relative URL to an absolute URL, get the currently selected node, perform URL encoding on a string, or run a translator on the document.

Conversion functions

Conversion functions convert tables to layers, layers to tables, and cascading style sheets (CSS) to HTML markup. Each function exactly duplicates the behavior of one of the conversion commands in the File or Modify menu.

dom.convertLayersToTable()

Availability

Dreamweaver 3.

Description

Opens the Convert Layers to Table dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canConvertLayersToTable()" on page 433.

dom.convertTablesToLayers()

Availability

Dreamweaver 3.

Description

Opens the Convert Tables to Layers dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canConvertTablesToLayers()" on page 434.

Command functions

Command functions help you make the most of the files in the Configuration/Commands folder. They manage the Command menu and call commands from other types of extension files.

dreamweaver.editCommandList()

Availability

Dreamweaver 3.

Description

Opens the Edit Command List dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.popupCommand() (deprecated)

Availability

Dreamweaver 2; deprecated in 3 in favor of dreamweaver.runCommand().

Description

This function executes the specified command. To the user, the effect is the same as selecting the command from a menu; if a dialog box is associated with the command, it appears. This function provides the ability to call a command from another extension file. It blocks other edits until the user closes the dialog box.

Note: This function can be called within the `objectTag()` function, from any script in a command file, or from the Property inspector file.

Arguments

`commandFile`

- The *`commandFile`* argument is the name of a command file within the Configuration/ Commands folder (for example, "`Format Table.htm`").

Returns

Nothing.

dreamweaver.runCommand()

Availability

Dreamweaver 3.

Description

Executes the specified command; it works the same as selecting the command from a menu. If a dialog box is associated with the command, it appears and the command script blocks other edits until the user closes the dialog box. This function provides the ability to call a command from another extension file.

Note: This function can be called within the `objectTag()` function, from any script in a command file, or from the Property inspector file.

Arguments

`commandFile`, *`{commandArg1}`*, *`{commandArg2}`*,...*`{commandArgN}`*

- The *`commandFile`* argument is a filename in the Configuration/Commands folder.
- The remaining arguments, *`commandArg1, commandArg2`*, and so on, which are optional, pass to the `receiveArguments()` function in the *`commandFile`* argument.

Returns

Nothing.

Example

You can write a custom Property inspector for tables that lets users get to the Format Table command from a button on the inspector by calling the following function from the button's `onClick` event handler:

```
function callFormatTable(){
  dreamweaver.runCommand('Format Table.htm');
}
```

File manipulation functions

File manipulation functions handle creating, opening, and saving documents (including XML and XHTML), converting existing HTML documents into XHTML, and exporting CSS to external files. These functions accomplish such tasks as browsing for files or folders, creating files based on templates, closing documents, and getting information about recently opened files.

dom.cleanupXHTML()

Availability

Dreamweaver MX.

Description

This function is similar to the `convertToXHTML()` function, but it cleans up an existing XHTML document. This function can run on a selection within the document. You can run the `cleanupXHTML()` function to clean up the syntax in an entire XHTML document or in the current selection of a document.

Arguments

`bWholeDoc`

- The *`bWholeDoc`* argument holds a Boolean value. If the value is `true`, the `cleanupXHTML()` function cleans up the entire document; otherwise, this function cleans up only the selection.

Returns

An array of six integers that quantify the number of the following elements:

- XHTML errors that Dreamweaver fixed
- The `map` elements that do not have an `id` attribute and cannot be fixed
- The `script` elements that do not have a `type` attribute and cannot be fixed
- The `style` elements that do not have a `type` attribute and cannot be fixed
- The `img` elements that do not have an `alt` attribute and cannot be fixed
- The `area` elements that do not have an `alt` attribute and cannot be fixed

dom.convertToXHTML()

Availability

Dreamweaver MX.

Description

Parses the HTML into a DOM tree, inserts missing items that are required for XHTML, cleans up the tree, and then writes the tree as clean XHTML. The missing directives, declarations, elements, and attributes that the `convertToXHTML()` function adds to the DOM tree, as necessary, include the following items:

- An XML directive
- A `doctype` declaration
- The `xmlns` attribute in the `html` element
- A `head` section

- A `title` element
- A `body` section

During the conversion, the `dom.convertToXHTML()` function converts pure HTML tags and attributes to lowercase, writes HTML tags and attributes with correct XHTML syntax, and adds missing HTML attributes where it can. This function treats third-party tags and attributes according to the settings in the Preferences dialog box.

If the document is a template, the `dom.convertToXHTML()` function alerts the user but does not perform the conversion.

Arguments

None.

Returns

An array of six integers that quantify the following items:

- XHTML errors that Dreamweaver fixed
- The `map` elements that do not have an `id` attribute and cannot be fixed
- The `script` elements that do not have a `type` attribute and cannot be fixed
- The `style` elements that do not have a `type` attribute and cannot be fixed
- The `img` elements that do not have an `alt` attribute and cannot be fixed
- The `area` elements that do not have an `alt` attribute and cannot be fixed

Example

In normal use, an extension first calls the `dreamweaver.openDocument()` or `dreamweaver.getDocumentDOM()` functions to get a reference to the document. The extension then calls the `dom.getIsXHTMLDocument()` function to determine whether the document is already in XHTML form. If it is not, the extension calls the `dom.convertToXHTML()` function to convert the document into XHTML. Then the extension calls the `dreamweaver.saveDocument()` function to save the converted file with a new filename.

dom.getIsXHTMLDocument()

Availability

Dreamweaver MX.

Description

Checks a document (specifically, the `<!DOCTYPE>` declaration) to see whether it is XHTML.

Arguments

None.

Returns

A `true` value if the document is XHTML; `false` otherwise.

dreamweaver.browseForFileURL()

Availability

Dreamweaver 1, enhanced in 2, 3, and 4.

Description

Opens the specified type of dialog box with the specified label in the title bar.

Arguments

openSelectOrSave, *{titleBarLabel}*, *{bShowPreviewPane}*, ¬
{bSupressSiteRootWarnings}, *{arrayOfExtensions}*

- The *openSelectOrSave* argument is a string that indicates the type of dialog box as `"open"`, `"select"`, or `"save"`.
- The *titleBarLabel* argument (added in Dreamweaver 2) is the label that should appear in the title bar of the dialog box. If this argument is omitted, Dreamweaver uses the default label that the operating system supplies.
- The *bShowPreviewPane* argument (added in Dreamweaver 2) is a Boolean value that indicates whether to display the Image Preview Pane in the dialog box. If this argument is a value of `true`, the dialog box filters for image files; if omitted, it defaults to a value of `false`.
- The *bSupressSiteRootWarnings* argument (added in Dreamweaver 3) is a Boolean value that indicates whether to suppress warnings about the selected file being outside the site root. If this argument is omitted, it defaults to a value of `false`.
- The *arrayOfExtensions* argument (added in Dreamweaver 4) is an array of strings for specifying default content for the Files of type list menu at the bottom of the dialog box. The proper syntax is `menuEntryText|.xxx[;.yyy;.zzz]|CCCC|`, where *menuEntryText* is the name of the file type to appear. The extensions can be specified as *.xxx[;.yyy;.zzz]* or *CCCC*, where *.xxx* specifies the file extension for the file type (optionally, *.yyy* and *.zzz* specify multiple file extensions) and `CCCC` is the four-character file type constant for the Macintosh.

Returns

A string that contains the name of the file, which is expressed as a file:// URL.

dreamweaver.browseForFolderURL()

Availability

Dreamweaver 3.

Description

Opens the Choose Folder dialog box with the specified label in the title bar.

Arguments

{titleBarLabel}, *{directoryToStartIn}*

- The *titleBarLabel* argument is the label that should appear in the title bar of the dialog box. If it is omitted, the *titleBarLabel* argument defaults to Choose Folder.
- The *directoryToStartIn* argument is the path where the folder should open, which is expressed as a file:// URL.

Returns

A string that contains the name of the folder, which is expressed as a file:// URL.

Example

The following code returns the URL of a folder:

```
return dreamweaver.browseForFolderURL('Select a Folder', ¬
dreamweaver.getSiteRoot());
```

dreamweaver.closeDocument()

Availability

Dreamweaver 2.

Description

Closes the specified document.

Arguments

`documentObject`

- The *`documentObject`* argument is the object at the root of a document's DOM tree (the value that the `dreamweaver.getDocumentDOM()` function returns). If the *`documentObject`* argument refers to the active document, the Document window might not close until the script that calls this function finishes executing.

Returns

Nothing.

dreamweaver.createDocument()

Availability

Dreamweaver 2, enhanced in Dreamweaver 4.

Description

Depending on the argument that you pass to this function, it opens a new document either in the same window or in a new window. The new document becomes the active document.

Note: This function can be called only from the menus.xml file, a command, or the Property inspector file. If a behavior action or object tries to call this function, Dreamweaver displays an error message.

Arguments

`{bOpenInSameWindow}, {type}`

- The *`bOpenInSameWindow`* argument is a Boolean value that indicates whether to open the new document in the current window. If the *`bOpenInSameWindow`* argument is a value of `false`, if it is omitted, or if the function is called on the Macintosh, the new document opens in a separate window.

- The *type* argument specifies the type of document to create, as declared in the Dreamweaver Configuration/DocumentTypes/MMDocumentTypes.xml file as the `id` attribute of the `documenttype` tag. For example, the *type* argument could be `"HTML"`, `"ASP-JS"`, `"ASP-VB"`, `"ColdFusion"`, `"CFC"`, `"JSP"`, `"ASP.NET_VB"`, and so on. For a complete list of possible types, see the MMDocumentTypes.xml file. If you do not specify *type*, the value defaults to `"HTML"`.

Note: You can extend the MMDocumentTypes file by adding your own document types. For information on extending document types, see *Extending Dreamweaver*.

Returns

The document object for the newly created document. This is the same value that the `dreamweaver.getDocumentDOM()`function returns.

dreamweaver.createXHTMLDocument()

Availability

Dreamweaver MX.

Description

Depending on the argument that you pass to this function, it opens a new XHTML document either in the same window or in a new window. The new document becomes the active document. It is similar to the `dreamweaver.createDocument()` function.

When Dreamweaver creates a new XHTML document, it reads a file named default.xhtml, which is located in the Configuration/Templates folder, and, using the content of that file, creates an output file that contains the following skeleton declarations:

```
<?xml version="1.0">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=" />
</head>

<body bgcolor="#FFFFFF" text="#000000">

</body>
</html>
```

The default document type definition (DTD) declaration is `XHTML 1.0 Transitional`, rather than `Strict`. If the user adds a frameset to the document, Dreamweaver switches the DTD to `XHTML 1.0 Frameset`. `Content-Type` is `text/html`, and `charset` is intentionally left out of the default.xhtml file but is filled in before the user views the new document. The `?xml` directive is not required if the document uses UTF-8 or UTF-16 character encoding; if it is present, it might be rendered by some older browsers. However, because this directive should be in an XHTML document, by default, Dreamweaver uses it (for both new and converted documents). Users can manually delete the directive. The `?xml` directive includes the encoding attribute, which matches the `charset` in the `Content-Type` attribute.

Arguments

{bOpenInSameWindow}

- The *bOpenInSameWindow* argument is a Boolean value that indicates whether to open the new document in the current window. If this value is `false` or omitted, or if the function is called on the Macintosh, the new document opens in a separate window.

Returns

The document object for the newly created document, which is the same value that the `dreamweaver.getDocumentDOM()` function returns.

dreamweaver.createXMLDocument()

Availability

Dreamweaver MX.

Description

Creates and opens a new XML file, which is empty except for the XML directive.

Arguments

None.

Returns

The DOM of the new XML file.

Example

The following example creates a new document, which is empty except for the XML directive:

```
var theDOM = dreamweaver.createXMLDocument("document");
```

dreamweaver.exportCSS()

Availability

Dreamweaver 3.

Description

Opens the Export Styles as a CSS File dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canExportCSS()" on page 444.

dreamweaver.exportEditableRegionsAsXML() (deprecated)

Availability

Dreamweaver 3; deprecated in MX.

Description

This function opens the Export Editable Regions as XML dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.exportTemplateDataAsXML()

Availability

Dreamweaver MX.

Description

Exports the current document to the specified file as XML. This function operates on the document that has focus, which must be a template. If you do not specify a filename argument, Dreamweaver MX opens a dialog box to request the export file string.

Arguments

{filePath}

- The *filePath* argument, which is optional, is a string that specifies the filename to which Dreamweaver exports the template. Express the *filePath* argument as a URL file string, such as `"file:///c|/temp/mydata.txt"`.

Returns

Nothing.

Enabler

"dreamweaver.canExportTemplateDataAsXML()" on page 444.

Example

```
if(dreamweaver.canExportTemplateDataAsXML())
{
   dreamweaver.exportTemplateDataAsXML("file:///c|/dw_temps/mytemplate.txt")
}
```

dreamweaver.getDocumentDOM()

Availability

Dreamweaver 2.

Description

Provides access to the objects tree for the specified document. After the tree of objects returns to the caller, the caller can edit the tree to change the contents of the document.

Arguments

{sourceDoc}

- The *sourceDoc* argument must be `"document"`, `"parent"`, `"parent.frames[`*number*`]"`, `"parent.frames['`*frameName*`']"`, or a URL. The `sourceDoc` value defaults to `"document"` if you do not supply a value. These argument values have the following meanings:
  - The `document` value specifies the document that has focus and contains the current selection.
  - The `parent` value specifies the parent frameset (if the currently selected document is in a frame).
  - The `parent.frames[`*number*`]` and `parent.frames['`*frameName*`']` values specify a document that is in a particular frame within the frameset that contains the current document.
  - If the argument is a relative URL, it is relative to the extension file.

Note: If the argument is `"document"`, the calling function must be the `applyBehavior()`, `deleteBehavior()`, `objectTag()` function, or any function in a command or Property inspector file that can perform edits to the document.

Returns

The JavaScript document object at the root of the tree.

Examples

The following example uses the `dreamweaver.getDocumentDOM()` function to access the current document:

```
var theDOM = dreamweaver.getDocumentDOM("document");
```

In the following example, the current document DOM identifies a selection and pastes it at the end of another document:

```
var currentDOM = dreamweaver.getDocumentDOM('document');
currentDOM.setSelection(100,200);
currentDOM.clipCopy();
var otherDOM = dreamweaver.openDocument(dreamweaver.¬
getSiteRoot() + "html/foo.htm");
otherDOM.endOfDocument();
otherDOM.clipPaste();
```

Note: The `openDocument()` argument is used because `DOM` methods normally operate only on open documents. Running a function on a document that isn't open causes a Dreamweaver error. The `DOM` methods that can operate only on the active document or on closed documents indicate this fact in their descriptions.

dreamweaver.getNewDocumentDOM()

Availability

Dreamweaver MX.

Description

Provides access to the editable tree for a new, empty document. This function works in the same way as the `getDocumetDOM()` function, except that it points to a new document, not an existing one, and does not open the document.

Arguments

None.

Returns

A pointer to new, empty document.

Example

The following code returns the DOM for a new, empty document:

```
var theDOM = dreamweaver.getNewDocumentDOM();
```

dreamweaver.getRecentFileList()

Availability

Dreamweaver 3.

Description

Gets a list of all the files in the recent files list at the bottom of the File menu.

Arguments

None.

Returns

An array of strings that represent the paths of the most recently accessed files. Each path is expressed as a file:// URL. If there are no recent files, the function returns nothing.

dreamweaver.importXMLIntoTemplate()

Availability

Dreamweaver 3.

Description

Imports an XML text file into the current template document. This function operates on the document that has focus, which must be a template. If you do not specify a filename argument, Dreamweaver opens a dialog box to request the import file string.

Arguments

{filePath}

- The *filePath* argument, which is optional, is a string that specifies the filename to which Dreamweaver exports the template. Express the *filePath* argument as a URL file string, such as `"file:///c|/temp/mydata.txt"`.

Returns

Nothing.

dreamweaver.newDocument()

Availability

Dreamweaver MX.

Description

Opens a document in the current site and invokes the New Document dialog box.

Arguments

{bopenWithCurSiteAndShowDialog}

- The *bopenWithCurSiteAndShowDialog* argument, which is optional, has a value of `true` or `false`. Specify `true` to open a document with the current site and to cause the New Document dialog box to appear; `false` otherwise.

Returns

Nothing.

dreamweaver.newFromTemplate()

Availability

Dreamweaver 3.

Description

Creates a new document from the specified template. If no argument is supplied, the Select Template dialog box appears.

Arguments

{templateURL}, bMaintain

- The *templateURL* argument is the path to a template in the current site, which is expressed as a file:// URL.
- The *bMaintain* argument is a Boolean value, `true` or `false`, that indicates whether to maintain the link to the original template.

Returns

Nothing.

dreamweaver.openDocument()

Availability

Dreamweaver 2.

Description

Opens a document for editing in a new Dreamweaver window and gives it the focus. For a user, the effect is the same as selecting File > Open and selecting a file. If the specified file is already open, the window that contains the document comes to the front. The window that contains the specified file becomes the currently selected document. In Dreamweaver 2, if Check In/Check Out is enabled, the file is checked out before it opens. In Dreamweaver 3 and later, you must use dreamweaver.openDocumentFromSite() to get this behavior.

Note: This function will cause an error if called from Behavior action or object files.

Arguments

`fileName`

- The *`fileName`* argument is the name of the file to open, which is expressed as a URL. If the URL is relative, it is relative to the file that contains the script that called this function.

Returns

The document object for the specified file, which is the same value that the `dreamweaver.getDocumentDOM()` function returns.

dreamweaver.openDocumentFromSite()

Availability

Dreamweaver 3.

Description

Opens a document for editing in a new Dreamweaver window and gives it the focus. For a user, the effect is the same as double-clicking a file in the Site panel. If the specified file is already open, the window that contains the document comes to the front. The window that contains the specified file becomes the currently selected document.

Note: This function cannot be called from Behavior action or object files because it causes an error.

Arguments

`fileName`

- The *`fileName`* argument is the file to open, which is expressed as a URL. If the URL is relative, it is relative to the file that contains the script that called this function.

Returns

The document object for the specified file, which is the same value that the `dreamweaver.getDocumentDOM()` function returns.

dreamweaver.openInFrame()

Availability

Dreamweaver 3.

Description

Opens the Open In Frame dialog box. When the user selects a document, it opens into the active frame.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canOpenInFrame()" on page 445.

dreamweaver.releaseDocument()

Availability

Dreamweaver 2.

Description

Explicitly releases a previously referenced document from memory.

Documents that are referenced by the `dreamweaver.getObjectTags()`, `dreamweaver.getObjectRefs()`, `dreamweaver.getDocumentPath()`, or `dreamweaver.getDocumentDOM()` functions are automatically released when the script that contains the call finishes executing. If the script opens many documents, you must use this function to explicitly release documents before finishing the script to avoid running out of memory.

Note: This function is relevant only for documents that were referenced by a URL, are not currently open in a frame or document window, and are not extension files. Extension files are loaded into memory at startup and are not released until you quit Dreamweaver.

Arguments

`documentObject`

- The *`documentObject`* argument is the object at the root of a document's DOM tree, which is the value that the `dreamweaver.getDocumentDOM()` function returns.

Returns

Nothing.

dreamweaver.revertDocument()

Availability

Dreamweaver 3.

Description

Reverts the specified document to the previously saved version.

Arguments

`documentObject`

- The *`documentObject`* argument is the object at the root of a document's DOM tree, which is the value that the `dreamweaver.getDocumentDOM()` function returns.

Returns

Nothing.

Enabler

"dreamweaver.canRevertDocument()" on page 446.

dreamweaver.saveAll()

Availability

Dreamweaver 3.

Description

Saves all open documents, opening the Save As dialog box for any documents that have not been saved previously.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canSaveAll()" on page 446.

dreamweaver.saveDocument()

Availability

Dreamweaver 2.

Description

Saves the specified file on a local computer.

Note: In Dreamweaver 2, if the file is read-only, Dreamweaver tries to check it out. If the document is still read-only after this attempt, or if it cannot be created, an error message appears.

Arguments

documentObject, *{fileURL}*

- The *documentObject* argument is the object at the root of a document's DOM tree, which is the value that the `dreamweaver.getDocumentDOM()` function returns.
- The *fileURL* argument, which is optional, is a URL that represents a location on a local computer. If the URL is relative, it is relative to the extension file. In Dreamweaver 2, this argument is required. If the *fileURL* argument is omitted in Dreamweaver 4, the file is saved to its current location if it has been previously saved; otherwise, a Save dialog box appears.

Returns

A Boolean value that indicates success (`true`) or failure (`false`).

Enabler

"dreamweaver.canSaveDocument()" on page 447.

dreamweaver.saveDocumentAs()

Availability

Dreamweaver 3.

Description

Opens the Save As dialog box.

Arguments

documentObject

- The *documentObject* argument is the object at the root of a document's DOM tree, which is the value that the `dreamweaver.getDocumentDOM()` function returns.

Returns

Nothing.

dreamweaver.saveDocumentAsTemplate()

Availability

Dreamweaver 3.

Description

Opens the Save As Template dialog box.

Arguments

documentObject, {fileName}

- The *documentObject* argument is the object at the root of a document's DOM tree, which is the value that `dreamweaver.getDocumentDOM()` returns.
- The *fileName* argument, which is optional, is the name of the file to open, expressed as an absolute URL.

Returns

Nothing.

Enabler

"dreamweaver.canSaveDocumentAsTemplate()" on page 447.

dreamweaver.saveFrameset()

Availability

Dreamweaver 3.

Description

Saves the specified frameset or opens the Save As dialog box if the frameset has not previously been saved.

Arguments

`documentObject`

- The *`documentObject`* argument is the object at the root of a document's DOM tree, which is the value that the `dreamweaver.getDocumentDOM()` function returns.

Returns

Nothing.

Enabler

"dreamweaver.canSaveFramesetAs()" on page 448.

dreamweaver.saveFramesetAs()

Availability

Dreamweaver 3.

Description

Opens the Save As dialog box for the frameset file that includes the specified DOM.

Arguments

`documentObject`

- The *`documentObject`* argument is the object at the root of a document's DOM tree, which is the value that the `dreamweaver.getDocumentDOM()` function returns.

Returns

Nothing.

Enabler

"dreamweaver.canSaveFramesetAs()" on page 448.

Global document functions

Global document functions act on an entire document. They check spelling, check target browsers, set page properties, and determine correct object references for elements in the document.

dom.checkSpelling()

Availability

Dreamweaver 3.

Description

Checks the spelling in the document, opening the Check Spelling dialog box if necessary, and notifies the user when the check is complete.

Arguments

None.

Returns

Nothing.

dom.checkTargetBrowsers()

Availability

Dreamweaver 3.

Description

Runs a target browser check on the document. To run a target browser check on a folder or group of files, see "site.checkTargetBrowsers()" on page 224.

Arguments

None.

Returns

Nothing.

dom.getParseMode()

Availability

Dreamweaver MX 2004

Description

Gets the current parsing mode of the document, which controls how the document is validated and whether it shows up in the main document window as HTML.

Arguments

None.

Returns

A string that specifies the current parsing mode: `"html"`, `"xml"`, `"css"`, or `"text"`.

dom.hideInfoMessagePopup()

Availability

Dreamweaver MX 2004.

Description

Hides the tooltip-like message, if it is visible, for the document window.

Arguments

None.

Returns

Nothing.

See also

"dom.showInfoMessagePopup()" on page 269.

dom.runValidation()

Availability

Dreamweaver MX, optional arguments added in Dreamweaver MX 2004.

Description

Runs the Validator on a single specified document (this function is similar to site.runValidation()). The Validator checks the document for conformance with the language specified in the document doctype (such as HTML 4.0 or HTML 3.2) and the language specified by the server model (such as ColdFusion or ASP). If the document has no doctype, then the Validator uses the language setting specified in the Validator section of the Preferences dialog box.

Arguments

`{controlString}`, *`{bOpenResultsWindow}`*, *`{bShowInfoMessage}`*

- The *`controlString`* argument is an optional string with four possible values: an empty string, `"xml"`, `"auto-explicit"`, or `"auto-implicit"`.
  - If the argument is an empty string, the Validator performs a default validation. If the argument is `"xml"`, the Validator validates the document as XML.
  - If the argument is `"auto-explicit"` or `"auto-implicit"`, Dreamweaver performs an automatic validation (also known as an *inline* validation), which underlines errors in the Code view instead of opening the Validation results window (see "dom.source.getValidationErrorsForOffset()" on page 415 and "dom.getAutoValidationCount()" on page 409).
  - If the *`controlString`* argument is `"auto-explicit"`, Dreamweaver will prompt the user to save an unsaved document before running the validation.
  - If the *`controlString`* argument is `"auto-implicit"`, the validation will fail without notifying the user that the current document is unsaved.

Note: An automatic validation (defined by the *`controlString`* value `"auto-explicit"` or `"auto-implicit"`) is currently available only for a Target Browser Check.

- The *`bOpenResultsWindow`* argument is an optional Boolean value: `true` opens the Validation results window; false otherwise. The default value is `true`.
- The *`bShowInfoMessage`* argument is used only when the *`controlString`* argument is defined as `"auto-explicit"` or `"auto-implicit"`. The *`bShowInfoMessage`* argument is a Boolean value: `true` shows an informational message under the toolbar item, `DW_ValidatorErrors`, with the number of errors found; `false` displays nothing. The default value is `false`.

Returns

The Validation results window object.

Example

The following example runs a regular validation when the user selects the File > Check Page > Validate Markup menu option (or Validate Current Document in the Validation panel):

```
dw.getDocumentDOM().runValidation('');
```

The following example prompts the user to save an unsaved document, runs an automatic validation, does not open the Validation results window, but does show the total number of errors over the document toolbar button for `DW_ValidatorErrors`:

```
dw.getDocumentDOM().runValidation('auto-explicit', false, true);
```

The following example does not prompt the user to save an unsaved document. If the document has not been saved, the validation will not start. If the document has been saved, Dreamweaver runs an automatic validation, does not open the Validation results window, and does not indicate the total number of errors encountered on the document toolbar:

```
dw.getDocumentDOM().runValidation('auto-implicit', false);
```

dom.showInfoMessagePopup()

Availability

Dreamweaver MX 2004.

Description

Shows a tooltip-like message in the document window or under a toolbar item.

Arguments

`location, message, timeout`

- The *`location`* argument is a string that specifies a toolbar item, or is an empty string, or is one of the following keywords: `"top"`, `"topright"`, `"right"`, `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, or `"topleft"`. The tooltip is placed against the specified edge or corner and is centered. An empty string causes it to be centered in the document. To specify a toolbar item, use `"toolbar:`*`toolbarID`*`:`*`itemID`*`"`, where the toolbar ID and toolbar item ID match the IDs in the toolbars.xml file.
- The *`message`* argument is a string that contains the message.
- The *`timeout`* argument is a number that specifies the milliseconds for which to display the message. The default is 0. If the value is 0, the message stays indefinitely. Dreamweaver automatically dismisses it if the user clicks it or switches documents, or if the time out expires.

Returns

Nothing.

Example

The following example displays two tooltip messages. The first line of code displays the message `"This message is in the center"` in the center of the document. The second call to `showInfoMessagePopup()` displays the tooltip message `"Don't forget the title for the Window"` for the Title text edit box, which has the ID `DW_SetTitle`, on the toolbar with the ID `DW_Toolbar_Main`.

```
dw.getDocumentDOM.showInfoMessagePopup('', 'This message is in the center',
  5000);
dw.getDocumentDOM.showInfoMessagePopup('toolbar:DW_Toolbar_Main:DW_SetTitle',
  'Don't forget the title for the window', 5000);
```

See also

"dom.hideInfoMessagePopup()" on page 268.

dom.showPagePropertiesDialog()

Availability

Dreamweaver 3.

Description

Opens the Page Properties dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.doURLDecoding()

Availability

Dreamweaver MX.

Description

Uses the internal Dreamweaver URL decoding mechanism to decode special characters and symbols in URL strings. For example, this function decodes `%20` to a space character and the name `&quot` to ".

Arguments

`inStr`

- The `inStr` argument is the string to decode.

Returns

A string that contains the decoded URL.

Example

The following example calls `dw.doURLDecoding()` to decode the special characters in its argument and store the resulting string in `outstr`:

```
outStr = dreamweaver.doURLDecoding("http://maps.yahoo.com/py/
   ddResults.py?Pyt=Tmap&tarname=&tardesc=&newname=&newdesc=&newHash=&newTHash
   =&newSts=&newTSts=&tlt=&tln=&slt=&sln=&newFL=Use+Address+Below&newaddr=2000
   +Shamrock+Rd&newcsz=Metroo+Park%2C+CA&newcountry=us&newTFL=Use+Address+Belo
   w&newtaddr=500+El+Camino&newtcsz=Santa+Clara%2C+CA&newtcountry=us&Submit=Ge
   t+Directions")
```

dreamweaver.getElementRef()

Availability

Dreamweaver 2.

Description

Gets the Netscape Navigator or Internet Explorer object reference for a specific tag object in the DOM tree.

Arguments

`NSorIE`, *`tagObject`*

- The *`NSorIE`* argument must be either `"NS 4.0"` or `"IE 4.0"`. The DOM and rules for nested references differ in Netscape Navigator 4.0 and Internet Explorer 4.0. This argument specifies for which browser to return a valid reference.
- The *`tagObject`* argument is a tag object in the DOM tree.

Returns

A string that represents a valid JavaScript reference to the object, such as `document.layers['myLayer']`. The string is subject to the following conditions:

- Dreamweaver returns correct references for Internet Explorer for `A`, `AREA`, `APPLET`, `EMBED`, `DIV`, `SPAN`, `INPUT`, `SELECT`, `OPTION`, `TEXTAREA`, `OBJECT`, and `IMG` tags.
- Dreamweaver returns correct references for Netscape Navigator for `A`, `AREA`, `APPLET`, `EMBED`, `LAYER`, `ILAYER`, `SELECT`, `OPTION`, `TEXTAREA`, `OBJECT`, and `IMG` tags, and for absolutely positioned `DIV` and `SPAN` tags. For `DIV` and `SPAN` tags that are not absolutely positioned, Dreamweaver returns `"cannot reference <`*`tag`*`>"`.
- Dreamweaver does not return references for unnamed objects. If an object does not contain either a `NAME` or an `ID` attribute, Dreamweaver returns `"unnamed <`*`tag`*`>"`. If the browser does not support a reference by name, Dreamweaver references the object by index (for example, `document.myform.applets[3]`).
- Dreamweaver returns references for named objects that are contained in unnamed forms and layers (for example, `document.forms[2].myCheckbox`).

dreamweaver.getObjectRefs() (deprecated)

Availability

Dreamweaver 1; deprecated in 3.

Description

This function scans the specified document for instances of the specified tags or, if no tags are specified, for all tags in the document and formulates browser-specific references to those tags. This function is equivalent to calling `getElementsByTagName()` and then calling `dreamweaver.getElementRef()` for each tag in the `nodelist`.

Arguments

`NSorIE`, *`sourceDoc`*, *`{tag1}`*, *`{tag2}`*,...*`{tagN}`*

- The *`NSorIE`* argument must be either `"NS 4.0"` or `"IE 4.0"`. The DOM and rules for nested references differ in Netscape Navigator 4.0 and Internet Explorer 4.0. This argument specifies for which browser to return a valid reference.
- The *`sourceDoc`* argument must be `"document"`, `"parent"`, `"parent.frames[`*`number`*`]"`, `"parent.frames['`*`frameName`*`']"`, or a URL. The `document` value specifies the document that has the focus and contains the current selection. The `parent` value specifies the parent frameset (if the currently selected document is in a frame), and `parent.frames[`*`number`*`]` and `parent.frames['`*`frameName`*`']` specify a document that is in a particular frame within the frameset that contains the current document. If the argument is a relative URL, it is relative to the extension file.
- The third and subsequent arguments, if supplied, are the names of tags (for example, `"IMG"`, `"FORM"`, or `"HR"`).

Returns

An array of strings where each array is a valid JavaScript reference to a named instance of the requested tag type in the specified document (for example, `"document.myLayer.document.myImage"`) for the specified browser:

- Dreamweaver returns correct references for Internet Explorer for `A`, `AREA`, `APPLET`, `EMBED`, `DIV`, `SPAN`, `INPUT`, `SELECT`, `OPTION`, `TEXTAREA`, `OBJECT`, and `IMG` tags.
- Dreamweaver returns correct references for Netscape Navigator for `A`, `AREA`, `APPLET`, `EMBED`, `LAYER`, `ILAYER`, `SELECT`, `OPTION`, `TEXTAREA`, `OBJECT`, and `IMG` tags, and for absolutely positioned `DIV` and `SPAN` tags. For `DIV` and `SPAN` tags that are not absolutely positioned, Dreamweaver returns `"cannot reference <`*`tag`*`>"`.
- Dreamweaver does not return references for unnamed objects. If an object does not contain either a `NAME` or an `ID` attribute, Dreamweaver returns `"unnamed <`*`tag`*`>"`. If the browser does not support a reference by name, Dreamweaver references the object by index (for example, `document.myform.applets[3]`).
- Dreamweaver does return references for named objects that are contained in unnamed forms and layers (for example, `document.forms[2].myCheckbox`).

When the same list of arguments passes to `getObjectTags()`, the two functions return arrays of the same length and with parallel content.

dreamweaver.getObjectTags() (deprecated)

Availability

Dreamweaver1; deprecated in 3.

Description

This function scans the specified document for instances of the specified tags or, if no tags are specified, for all tags in the document. This function is equivalent to calling `getElementsByTagName()` and then getting `outerHTML` for each element in the `nodelist`.

Arguments

`sourceDoc`, *`{tag1}`*, *`{tag2}`*,...*`{tagN}`*

- The *`sourceDoc`* argument must be `"document"`, `"parent"`, `"parent.frames[`*`number`*`]"`, `"parent.frames['`*`frameName`*`']"`, or a URL. The `document` value specifies the document that has the focus and contains the current selection. The `parent` value specifies the parent frameset (if the currently selected document is in a frame), and `parent.frames[`*`number`*`]` and `parent.frames['`*`frameName`*`']` specify a document that is in a particular frame within the frameset that contains the current document. If the argument is a relative URL, it is relative to the extension file.
- The second and subsequent arguments, if supplied, are the names of tags (for example, `"IMG"`, `"FORM"`, `"HR"`).

Returns

An array of strings where each array is the source code for an instance of the requested tag type in the specified document.

- If one of the *`tag`* arguments is `LAYER`, the function returns all `LAYER` and `ILAYER` tags and all absolutely positioned `DIV` and `SPAN` tags.
- If one of the *`tag`* arguments is `INPUT`, the function returns all form elements. To get a particular type of form element, specify `INPUT/`*`TYPE`*, where *`TYPE`* is `button`, `text`, `radio`, `checkbox`, `password`, `textarea`, `select`, `hidden`, `reset`, or `submit`.

When the same list of arguments passes to `getObjectRefs()`, the two functions return arrays of the same length.

Example

`dreamweaver.getObjectTags("document", "IMG")`, depending on the contents of the active document, might return an array with the following items:

- `"<IMG SRC="/images/dot.gif" WIDTH="10" HEIGHT="10" NAME="bullet">"`
- `"<IMG SRC="header.gif" WIDTH="400" HEIGHT="32" NAME="header">"`
- `"<IMG SRC="971208_nj.jpg" WIDTH="119" HEIGHT="119" NAME="headshot">"`

dreamweaver.getPreferenceInt()

Availability

Dreamweaver MX.

Description

Lets you retrieve an integer preference setting for an extension.

Arguments

`section, key, default_value`

- The *`section`* argument is a string that specifies the preferences section that contains the entry.
- The *`key`* argument is a string that specifies the entry of the value to retrieve.
- The *`default_value`* argument is the default value that Dreamweaver returns if it cannot find the entry. This value must be an unsigned integer in the range 0 through 65,535 or a signed integer in the range -32,768 through 32,767.

Returns

Integer value of the specified entry in the specified section or the default value if the function does not find the entry. Returns 0 if the value of the specified entry is not an integer.

Example

The following example returns the integer value of the Snap Distance setting in the My Extension section of Preferences. If there is no MyExtension section or no Snap Distance entry, the function returns the specified default value of 0.

```
var snapDist; //default value if entry not found
snapDist = dreamweaver.getPreferenceInt("My Extension", "Snap Distance", 0);
```

dreamweaver.getPreferenceString()

Availability

Dreamweaver MX.

Description

Lets you retrieve a string preference setting that you stored for an extension.

Arguments

`section, key, default_value`

- The *`section`* argument is a string that specifies the preferences section that contains the entry.
- The *`key`* argument is a string that specifies the value to retrieve.
- The *`default_value`* argument is the default string value that Dreamweaver returns if it cannot find the entry.

Returns

The requested preference string, or if the string cannot be found, the default value.

Example

The following example returns the String value of the Text Editor setting in the My Extension section of Preferences. If there is no MyExtension section or no Text Editor entry, the function returns the default value specified by the variable `txtEditor`.

```
var txtEditor = getExternalTextEditor(); //set default text Editor value
txtEditor = dreamweaver.getPreferenceString("My Extension", "Text Editor", 
  txtEditor);
```

dreamweaver.setPreferenceInt()

Availability

Dreamweaver MX.

Description

Lets you set an integer preference setting for an extension. This setting is stored with Dreamweaver preferences when Dreamweaver is not running.

Arguments

`section`, *`key`*, *`new_value`*

- The *`section`* argument is a string that specifies the preferences category in which the option is set. If the category does not exist, Dreamweaver creates it.
- The *`key`* argument is a string that specifies the category option that the function sets. If the option does not exist, Dreamweaver creates it.
- The *`new_value`* argument is an integer that specifies the value of the category option.

Returns

A `true` value if successful; `false` otherwise.

Example

The following example sets the Snap Distance entry to the value of the variable `snapDist` in the My Extension category of Preferences:

```
var snapDist = getSnapDistance();
if(snapDist > 0)
{
  dreamweaver.setPreferenceInt("My Extension", "Snap Distance", snapDist);
}
```

dreamweaver.setPreferenceString()

Availability

Dreamweaver MX.

Description

Lets you write a string preference setting for an extension. This setting is stored with Dreamweaver preferences when Dreamweaver is not running.

Arguments

`section`, *`key`*, *`new_value`*

- The *`section`* argument is a string that specifies the Preferences category in which the option is set. If the category does not exist, Dreamweaver creates it.
- The *`key`* argument is a string that specifies the category option that the functions sets. If the category option does not exist, Dreamweaver creates it.
- The *`new_value`* argument is a string that specifies the value of the category option.

Returns

A `true` value if successful; `false` otherwise.

Example

```
var txtEditor = getExternalTextEditor();
dreamweaver.setPreferenceString("My Extension", "Text Editor", txtEditor);
```

dreamweaver.showTargetBrowsersDialog()

Availability

Dreamweaver MX 2004.

Description

Opens the Target Browsers dialog box. The Target Browsers dialog box lets a user specify which browser versions the Browser Target Check feature should use for checking the current page's browser compatibility issues.

Arguments

None.

Returns

Nothing.

Path functions

Path functions get and manipulate the paths to various files and folders on a user's hard disk. These functions determine the path to the root of the site in which the current document resides, convert relative paths to absolute URLs, and more.

dreamweaver.getConfigurationPath()

Availability

Dreamweaver 2.

Description

Gets the path to the Dreamweaver Configuration folder, which is expressed as a file:// URL.

For information on how Dreamweaver accesses Configuration folders on a multiuser platform, see "C-Level Extensibility" in Extending Dreamweaver Help.

Arguments

None.

Returns

The path to the application configurations.

Example

The following function is useful when referencing other extension files, which are stored in the Configuration folder in the Dreamweaver application folder:

```
var sortCmd = dreamweaver.getConfigurationPath() + ¬
"/Commands/Sort Table.htm"
var sortDOM = dreamweaver.getDocumentDOM(sortCmd);
```

dreamweaver.getDocumentPath()

Availability

Dreamweaver 1.2.

Description

Gets the path of the specified document, which is expressed as a file:// URL. This function is equivalent to calling `dreamweaver.getDocumentDOM()` and reading the `URL` property of the return value.

Arguments

`sourceDoc`

- The value of the *`sourceDoc`* argument must be `"document"`, `"parent"`, `"parent.frames[`*`number`*`]"`, or `"parent.frames['`*`frameName`*`']"`. The `"document"` value specifies the document that has the focus and contains the current selection. The `"parent"` value specifies the parent frameset (if the currently selected document is in a frame), and the `"parent.frames[`*`number`*`]"` and `"parent.frames['`*`frameName`*`']"` values specify a document that is in a particular frame within the frameset that contains the current document.

Returns

Either a string that contains the URL of the specified document if the file was saved or an empty string if the file was not saved.

dreamweaver.getSiteRoot()

Availability

Dreamweaver 1.2.

Description

Gets the local root folder (as specified in the Site Definition dialog box) for the site that is associated with the currently selected document, which is expressed as a file:// URL.

Arguments

None.

Returns

Either a string that contains the URL of the local root folder of the site where the file is saved or an empty string if the file is not associated with a site.

dreamweaver.getTempFolderPath()

Availability

Dreamweaver MX.

Description

Gets the full path to a temporary folder where you can store temporary or transient files. This function looks for a Temp folder inside the Dreamweaver Configuration folder. If the system supports multiple users, it looks in the user's Configuration folder. If a Temp folder does not exist, the function creates it. Shared files that are not transient should be stored in the Configuration/ Shared folder.

Arguments

None.

Returns

The full path to the folder, which is expressed as a file:// URL.

Example

The following line of code returns the full path for the specified file. The `dw.getTempFolderPath()` function does not return a slash (/) at the end of the path, as do other Dreamweaver functions (such as `dreamweaver.getSiteRoot()`):

```
var myTempfile = dw.getTempFolderPath() + "/myTempFile.txt";
```

dreamweaver.relativeToAbsoluteURL()

Availability

Dreamweaver 2.

Description

Given a relative URL and a point of reference (either the path to the current document or the site root), this function converts the relative URL to an absolute file:// URL.

Arguments

`docPath`, *`siteRoot, relURL`*

- The *`docPath`* argument is the path to a document on the user's computer (for example, the current document), which is expressed as a file:// URL or an empty string if *`relURL`* is a root-relative URL.
- The *`siteRoot`* argument is the path to the site root, which is expressed as a file:// URL or an empty string if *`relURL`* is a document-relative URL.
- The *`relURL`* argument is the URL to convert.

Returns

An absolute URL string. The return value is generated, as described in the following list:

- If *`relURL`* is an absolute URL, no conversion occurs, and the return value is the same as *`relURL`*.
- If *`relURL`* is a document-relative URL, the return value is the combination of *`docPath`* + *`relURL`*.
- If *`relURL`* is a root-relative URL, the return value is the combination of *`siteRoot`* + *`relURL`*.

Selection functions

Selection functions get and set the selection in open documents. For information on getting or setting the selection in the Site panel, see "Site functions" on page 220.

dom.getSelectedNode()

Availability

Dreamweaver 3.

Description

Gets the selected node. Using this function is equivalent to calling the `dom.getSelection()` function and passing the return value to the `dom.offsetsToNode()` function.

Arguments

None.

Returns

The tag, text, or comment object that completely contains the specified range of characters.

dom.getSelection()

Availability

Dreamweaver 3.

Description

Gets the selection, which is expressed as character offsets into the document's source code.

Arguments

{bAllowMultiple}

- The *bAllowMultiple* argument, which is optional, is a Boolean value that indicates whether the function should return multiple offsets if more than one table cell, image map hotspot, or layer is selected.

  If this argument is omitted, it defaults to `false`.

Returns

For simple selections, an array that contains two integers. The first integer is the character offset of the opening of the selection. The second integer is the character offset at the closing of the selection. If the two numbers are the same, the current selection is an insertion point.

For complex selections (multiple table cells, multiple layers, or multiple image map hotspots), an array that contains 2*n* integers, where *n* is the number of selected items. The first integer in each pair is the character offset of the opening of the selection (including the opening `TD`, `DIV`, `SPAN`, `LAYER`, `ILAYER`, or `MAP` tag); the second integer in each pair is the character offset of the closing of the selection (including the closing `TD`, `DIV`, `SPAN`, `LAYER`, `ILAYER`, or `MAP` tag). If multiple table rows are selected, the offsets of each cell in each row return. The selection never includes the `TR` tags.

dom.nodeToOffsets()

Availability

Dreamweaver 3.

Description

Gets the position of a specific node in the DOM tree, which is expressed as character offsets into the document's source code. It is valid for any document on a local drive.

Arguments

node

- The *node* argument must be a tag, comment, or range of text that is a node in the tree that the `dreamweaver.getDocumentDOM()` function returns.

Returns

An array that contains two integers. The first integer is the character offset of the beginning of the tag, text, or comment. The second integer is the character offset of the end of the node, from the beginning of the HTML document.

Example

The following code selects the first image object in the current document:

```
var theDOM = dw.getDocumentDOM();
var theImg = theDOM.images[0];
var offsets = theDom.nodeToOffsets(theImg);
theDom.setSelection(offsets[0], offsets[1]);
```

dom.offsetsToNode()

Availability

Dreamweaver 3.

Description

Gets the object in the DOM tree that completely contains the range of characters between the specified opening and closing points. It is valid for any document on a local drive.

Arguments

`offsetBegin`, *`offsetEnd`*

- The *`offsetBegin`* argument specifies the offset from the beginning of the document to the beginning of a range of characters that is an object in the DOM tree.
- The *`offsetEnd`* argument specifies the offset from the beginning of the document to the end of a range of characters that is an object in the DOM tree.

Returns

The tag, text, or comment object that completely contains the specified range of characters.

Example

The following code displays an alert if the selection is an image:

```
var offsets = dom.getSelection();
var theSelection = dreamweaver.offsetsToNode(offsets[0], ¬
offsets[1]);
if (theSelection.nodeType == Node.ELEMENT_NODE && ¬
theSelection.tagName == 'IMG'){
  alert('The current selection is an image.');
}
```

dom.selectAll()

Availability

Dreamweaver 3.

Description

Performs a Select All operation.

Note: In most cases, this function selects all the content in the active document. In some cases (for example, when the insertion point is inside a table), it selects only part of the active document. To set the selection to the entire document, use `dom.setSelection()`.

Arguments

None.

Returns

Nothing.

dom.setSelectedNode()

Availability

Dreamweaver 3.

Description

Sets the selected node. This function is equivalent to calling the `dom.nodeToOffsets()` function and passing the return value to the `dom.setSelection()` function.

Arguments

`node`, *`{bSelectInside}`*, *`{bJumpToNode}`*

- The *`node`* argument is a text, comment, or element node in the document.
- The *`bSelectInside`* argument, which is optional, is a Boolean value that indicates whether to select the `innterHTML` of the node. This argument is relevant only if *`node`* is an element node, and it defaults to `false` if it is omitted.
- The *`bJumpToNode`* argument, which is optional, is a Boolean value that indicates whether to scroll the Document window, if necessary, to make the selection visible. If it is omitted, this argument defaults to `false`.

Returns

Nothing.

dom.setSelection()

Availability

Dreamweaver 3.

Description

Sets the selection in the document.

Arguments

`offsetBegin`, *`offsetEnd`*

- These arguments are the opening and closing points, respectively, for the new selection, which is expressed as character offsets into the document's source code. If the two numbers are the same, the new selection is an insertion point. If the new selection is not a valid HTML selection, it is expanded to include the characters in the first valid HTML selection. For example, if *`offsetBegin`* and *`offsetEnd`* define the range `SRC="myImage.gif"` within `<IMG SRC="myImage.gif">`, the selection expands to include the entire `IMG` tag.

Returns

Nothing.

dreamweaver.getSelection() (deprecated)

Availability

Dreamweaver 2; deprecated in 3. See “dom.getSelection()” on page 280.

Description

Gets the selection in the current document, which is expressed as byte offsets into the document’s source code.

Arguments

None.

Returns

An array that contains two integers. The first integer is the byte offset for the beginning of the selection; the second integer is the byte offset for the end of the selection. If the two numbers are the same, the current selection is an insertion point.

dreamweaver.nodeExists()

Available

Dreamweaver 3.

Description

Determines whether the reference to the specified node is still good. Often when writing extensions, you reference a node and then perform an operation that deletes it (such as setting the `innerHTML` or `outerHTML` properties of its parent). This function lets you confirm that the node hasn’t been deleted before you attempt to reference any of its properties or methods. The referenced node does not need to be in the current document.

Arguments

`node`

- The *`node`* argument is the node that you want to check.

Returns

A Boolean value: `true` if the node exists; `false` otherwise.

Example

The following example gets the current node, locates a table within it, and later calls `dw.nodeExists()` to see if the original node still exists:

```
function applyFormatToSelectedTable(){

   // get current selection
   var selObj = dw.getDocumentDOM().getSelectedNode();

   alternateRows(dwscripts.findDOMObject("presetNames").selectedIndex,
     findTable());

// restore original selection, if it still exists; if not, just select the
// table.
```

```
    var selArr;

    if (dw.nodeExists(selObj))
      selArr = dom.nodeToOffsets(selObj);
    else
      selArr = dom.nodeToOffsets(findTable());

    dom.setSelection(selArr[0],selArr[1]);
}
```

dreamweaver.nodeToOffsets() (deprecated)

Availability

Dreamweaver 2; deprecated in 3 in favor of "`dom.nodeToOffsets()`" on page 280.

Description

Gets the position of a specific node in the DOM tree, which is expressed as byte offsets into the document's source code.

Arguments

`node`

- The *`node`* argument must be a tag, comment, or range of text that is a node in the tree that the `dreamweaver.getDocumentDOM()` function returns.

Returns

An array that contains two integers. The first integer is the byte offset for the opening of the tag, text, or comment; the second integer is the byte offset for the closing of the node.

dreamweaver.offsetsToNode() (deprecated)

Availability

Dreamweaver 2; deprecated in 3 in favor of "`dom.offsetsToNode()`" on page 281.

Description

Gets the object in the DOM tree that completely contains the range of characters between the specified opening and closing points.

Arguments

`offsetBegin`, *`offsetEnd`*

- These arguments are the opening and closing points, respectively, of a range of characters, which is expressed as byte offsets into the document's source code.

Returns

The tag, text, or comment object that completely contains the specified range of characters.

dreamweaver.selectAll()

Availability

Dreamweaver 3.

Description

Performs a Select All operation in the active document window, the Site panel or, on the Macintosh, the text field that has focus in a dialog box or floating panel.

Note: If the operation takes place in the active document, it usually selects all the content in the active document. In some cases (for example, when the insertion point is inside a table), however, it selects only part of the active document. To set the selection to the entire document, use the `dom.setSelection()` function.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canSelectAll()" on page 448.

dreamweaver.setSelection() (deprecated)

Availability

Dreamweaver 2; deprecated in 3 in favor of "`dom.setSelection()`" on page 282.

Description

Sets the selection in the current document. This function can move the selection only within the current document; it cannot change the focus to a different document.

Arguments

`offsetBegin`, *`offsetEnd`*

- These arguments are the opening and closing points, respectively, for the new selection, which is expressed as byte offsets into the document's source code. If the two numbers are the same, the new selection is an insertion point. If the new selection is not a valid HTML selection, it is expanded to include the characters in the first valid HTML selection. For example, if *`offsetBegin`* and *`offsetEnd`* define the range `SRC="myImage.gif"` within `<IMG SRC="myImage.gif">`, the selection expands to include the entire `IMG` tag.

Returns

Nothing.

String manipulation functions

String manipulation functions help you get information about a string as well as convert a string from Latin 1 encoding to platform-native encoding and back.

dreamweaver.doURLEncoding()

Availability

Dreamweaver 1.

Description

Takes a string and returns a URL-encoded string by replacing all the spaces and special characters with specified entities.

Arguments

`stringToConvert`

- The *`stringToConvert`* argument is a string that contains the unencoded URL that the function encodes.

Returns

A URL-encoded string.

Example

The following example shows the `URL.value` for `"My URL-encoded string"`:

```
var URL = dw.doURLEncoding(theURL.value);
returns "My%20URL-encoded%20string"
```

dreamweaver.getTokens()

Availability

Dreamweaver 1.

Description

Accepts a string and splits it into tokens.

Arguments

`searchString`, *`separatorCharacters`*

- The *`searchString`* argument is the string to separate into tokens.
- The *`separatorCharacters`* argument is the character or characters that signifies the end of a token. Separator characters in quoted strings are ignored. Any white-space characters that occur in `separatorCharacters` (such as tabs) are treated as separator characters, as if they are explicitly specified. Two or more consecutive white space characters are treated as a single separator.

Returns

An array of token strings.

Example

The following call to the `dw.getTokens()` function returns the tokens that come after it:

```
dreamweaver.getTokens('foo("my arg1", 34)', '(),')
```

- `foo`
- `"my arg 1"`
- `34`

dreamweaver.latin1ToNative()

Availability

Dreamweaver 2.

Description

Converts a string in Latin 1 encoding to the native encoding on the user's computer. This function is intended to display the UI of an extension file in another language.

Note: This function has no effect in Windows because Windows encodings are already based on Latin 1.

Arguments

`stringToConvert`

- The *`stringToConvert`* argument is the string to convert from Latin 1 encoding to native encoding.

Returns

The converted string.

dreamweaver.nativeToLatin1()

Availability

Dreamweaver 2.

Description

Converts a string in native encoding to Latin 1 encoding.

Note: This function has no effect in Windows because Windows encodings are already based on Latin 1.

Arguments

`stringToConvert`

- The *`stringToConvert`* argument is the string to convert from native encoding to Latin 1 encoding.

Returns

The converted string.

dreamweaver.scanSourceString()

Availability

Dreamweaver UltraDev 1.

Description

Scans a string of HTML and finds the tags, attributes, directives, and text. For each tag, attribute, directive, and text span that it finds, the `scanSourceString()` function invokes a callback function that you must supply. Dreamweaver supports the following callback functions:

- `openTagBegin()`
- `openTagEnd()`
- `closeTagBegin()`
- `closeTagEnd()`
- `directive()`
- `attribute()`
- `text()`

Dreamweaver calls the seven callback functions on the following occasions:

1 Dreamweaver calls `openTagBegin()` for each opening tag (for example, `<font>`, as opposed to `</font>`) and each empty tag (for example, `<img>` or `<hr>`). The `openTagBegin()` function accepts two arguments: the name of the tag (for example, `"font"` or `"img"`) and the document offset, which is the number of bytes in the document before the beginning of the tag. The function returns `true` if scanning should continue or `false` if it should stop.

2 After `openTagBegin()` executes, Dreamweaver calls `attribute()` for each HTML attribute. The `attribute()` function accepts two arguments, a string that contains the attribute name (for example, `"color"` or `"src"`) and a string that contains the attribute value (for example, `"#000000"` or `"foo.gif"`). The `attribute()` function returns a Boolean value that indicates whether scanning should continue.

3 After all the attributes in the tag have been scanned, Dreamweaver calls `openTagEnd()`. The `openTagEnd()` function accepts one argument, the document offset, which is the number of bytes in the document before the end of the opening tag. It returns a Boolean value that indicates whether scanning should continue.

4 Dreamweaver calls `closeTagBegin()` for each closing tag (for example, `</font>`). The function accepts two arguments, the name of the tag to close (for example, `"font"`) and the document offset, which is the number of bytes in the document before the beginning of the closing tag. The function returns a Boolean value that indicates whether scanning should continue.

5 After `closeTagBegin()` returns, Dreamweaver calls the `closeTagEnd()` function. The `closeTagEnd()` function accepts one argument, the document offset, which is the number of bytes in the document before the end of the closing tag. It returns a Boolean value that indicates whether scanning should continue.

6 Dreamweaver calls the `directive()` function for each HTML comment, ASP script, JSP script, or PHP script. The `directive()` function accepts two arguments, a string that contains the directive and the document offset, which is the number of bytes in the document before the end of the closing tag. The function returns a Boolean value that indicates whether scanning should continue.

7 Dreamweaver calls the `text()` function for each span of text in the document (that is, everything that is not a tag or a directive). Text spans include text that is not visible to the user, such as the text inside a `<title>` or `<option>` tag. The `text()` function accepts two arguments, a string that contains the text and the document offset, which is the number of bytes in the document before the closing of the closing tag. The `text()` function returns a Boolean value that indicates whether scanning should continue.

Arguments

`HTMLstr`, *`parserCallbackObj`*

- The *`HTMLstr`* argument is a string that contains code.
- The *`parserCallbackObj`* argument is a JavaScript object that has one or more of the following methods: `openTagBegin()`, `openTagEnd()`, `closeTagBegin()`, `closeTagEnd()`, `directive()`, `attribute()`, and `text()`. For best performance, *`parserCallbackObj`* should be a shared library that is defined using the C-Level Extensibility interface. Performance is also improved if *`parserCallbackObj`* defines only the callback functions that it needs.

Returns

A Boolean value: `true` if the operation completed successfully; `false` otherwise.

Example

The following sequence of steps provide an example of how to use the `dreamweaver.scanSourceString()` function:

1 Create an implementation for one or more of the seven callback functions.

2 Write a script that calls the `dreamweaver.scanSourceString()` function.

3 The `dreamweaver.scanSourceString()` function passes a string that contains HTML and pointers to the callback functions that you wrote. For example, the string of HTML is `"<font size=2>hello</font>"`.

4 Dreamweaver analyzes the string and determines that the string contains a font tag. Dreamweaver calls the callback functions in the following sequence:

  - The `openTagBegin()` function
  - The `attribute()` function (for the size attribute)
  - The `openTagEnd()` function
  - The `text()` function (for the `"hello"` string)
  - The `closeTagBegin()` and `closeTagEnd()` functions

Translation functions

Translation functions deal either directly with translators or with translation results. These functions get information about or run a translator, edit content in a locked region, and specify that the translated source should be used when getting and setting selection offsets.

dom.runTranslator()

Availability

Dreamweaver 3.

Description

This function runs the specified translator on the document. This function is valid only for the active document.

Arguments

`translatorName`

- The *`translatorName`* argument is the name of a translator as it appears in the Translation preferences.

Returns

Nothing.

dreamweaver.editLockedRegions()

Availability

Dreamweaver 2.

Description

Depending on the value of the argument, this function makes locked regions editable or non-editable. By default, locked regions are non-editable; if you try to edit a locked region before specifically making it editable with this function, Dreamweaver beeps and does not allow the change.

Note: Editing locked regions can have unintended consequences for library items and templates. You should not use this function outside the context of data translators.

Arguments

`bAllowEdits`

- The *`bAllowEdits`* argument is a Boolean value: `true` indicates that edits are allowed; `false` otherwise. Dreamweaver automatically restores locked regions to their default (non-editable) state when the script that calls this function finishes executing.

Returns

Nothing.

dreamweaver.getTranslatorList()

Availability

Dreamweaver 3.

Description

This function gets a list of the installed translators.

Arguments

None.

Returns

An array of strings where each string represents the name of a translator as it appears in the Translation preferences.

dreamweaver.useTranslatedSource()

Availability

Dreamweaver 2.

Description

This function specifies that the values that `dom.nodeToOffsets()` and `dom.getSelection()` return. These are used by `dom.offsetsToNode()` and `dom.setSelection()` and should be offsets into the translated source (the HTML that is contained in the DOM after a translator runs), not the untranslated source.

Note: This function is relevant only in Property inspector files.

Arguments

`bUseTranslatedSource`

- The *`bUseTranslatedSource`* argument is a Boolean value: `true` if the function uses offsets into the translated source; `false` if the function uses the untranslated source.

  The default value of the argument is `false`. Dreamweaver automatically uses the untranslated source for subsequent calls to `dw.getSelection()`, `dw.setSelection()`, `dw.nodeToOffsets()`, and `dw.offsetsToNode()` when the script that calls `dw.useTranslatedSource()` finishes executing, if `dw.useTranslatedSource()` is not explicitly called with an argument of `false` before then.

Returns

Nothing.

CHAPTER 15
Page Content

The page content functions perform operations that affect the content of a web page. These operations include manipulating assets in the Assets panel, adding behaviors, cutting and pasting elements from the Clipboard, applying a template, or inserting a code snippet.

Assets panel functions

Assets panel functions, which are programmed into the API as an asset panel, let you manage and use the elements in the Assets panel (templates, libraries, images, Macromedia Shockwave and Macromedia Flash content, URLs, colors, and scripts).

dreamweaver.assetPalette.addToFavoritesFromDocument()

Availability

Dreamweaver 4.

Description

Adds the element that is selected in the Document window to the Favorites list. This function handles only images, Shockwave files, Flash files, text font colors, and URLs.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.addToFavoritesFromSiteAssets()

Availability

Dreamweaver 4.

Description

Adds elements that are selected in the Site list to the Favorites list and gives each item a nickname in the Favorites list. This function does not remove the element from the Site list.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.addToFavoritesFromSiteWindow()

Availability

Dreamweaver 4.

Description

Adds the elements that are selected in the Site panel or site map to the Favorites list. This function handles only images, movies, scripts, Shockwave files, Flash files, and URLs (in the case of the site map). If other folders or files are selected, they are ignored.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.copyToSite()

Availability

Dreamweaver 4.

Description

Copies selected elements to another site and puts them in that site's Favorites list. If the elements are files (other than colors or URLs), the actual file is copied into that site.

Arguments

`targetSite`

- The *`targetSite`* argument is the name of the target site, which the `site.getSites()` call returns.

Returns

Nothing.

dreamweaver.assetPalette.edit()

Availability

Dreamweaver 4.

Description

Edits selected elements with primary external editor or Custom Edit control. For colors, the color picker appears. For URLs, a dialog box appears and prompts the user for a URL and a nickname. This function is not available for the Site list of colors and URLs.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.assetPalette.canEdit()" on page 442.

dreamweaver.assetPalette.getSelectedCategory()

Availability

Dreamweaver 4.

Description

Returns the currently selected category.

Arguments

None.

Returns

The currently selected category, which can be one of the following: `"templates"`, `"library"`, `"images"`, `"movies"`, `"shockwave"`, `"flash"`, `"scripts"`, `"colors"`, or `"urls"`.

dreamweaver.assetPalette.getSelectedItems()

Availability

Dreamweaver 4.

Description

Returns an array of the selected items in the Assets panel, either in the Site or Favorites list.

Arguments

None.

Returns

An array of the following three strings for each selected item:

- The `name` string, which is the name/filename or nickname that appears in the Assets panel.
- The `value` string, which is the full path, full URL, or color value, depending on the selected item.
- The `type` string, which is either `"folder"` or one of the following categories: `"templates"`, `"library"`, `"images"`, `"movies"`, `"shockwave"`, `"flash"`, `"scripts"`, `"colors"`, or `"urls"`.

Note: If nothing is selected in the Assets panel, this function returns an array that contains one empty string.

Example

If URLs is the category, and a folder MyFolderName and a URL MyFavoriteURL are both selected in the Favorites list, the function returns the following list:

```
items[0] = "MyFolderName"
items[1] = "//path/FolderName"
items[2] = "folder"
items[3] = "MyFavoriteURL"
items[4] = "http://www.MyFavoriteURL.com"
items[5] = "urls"
```

dreamweaver.assetPalette.getSelectedView()

Availability

Dreamweaver 4.

Description

Indicates which list is currently shown in the Assets panel.

Arguments

None.

Returns

Returns a string that has a value of either `"site"` or `"favorites"`.

dreamweaver.assetPalette.insertOrApply()

Availability

Dreamweaver 4.

Description

Inserts selected elements or applies the element to the current selection. It applies templates, colors, and URLs to the selection; it also inserts URLs and other elements at the insertion point. If a document isn't open, the function is not available.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.assetPalette.canInsertOrApply()" on page 442.

dreamweaver.assetPalette.locateInSite()

Availability

Dreamweaver 4.

Description

Selects files that are associated with the selected elements in the local side of the Site panel. This function does not work for colors or URLs. It is available in the Site list and the Favorites list. If a folder is selected in the Favorites list, the folder is ignored.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.newAsset()

Availability

Dreamweaver 4.

Description

Creates a new element for the current category in the Favorites list. For library and templates, this is a new blank library or template file that the user can name immediately. For colors, the color picker appears. For URLs, a dialog box appears and prompts the user for a URL and a nickname. This function is not available for images, Shockwave files, Flash files, or scripts.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.newFolder()

Availability

Dreamweaver 4.

Description

Creates a new folder in the current category with the default name (untitled) and puts a text box around the default name. It is available only in the Favorites list.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.recreateLibraryFromDocument()

Availability

Dreamweaver 4.

Description

Replaces the deprecated `libraryPalette` function, `recreateLibraryFromDocument()`. It creates a Library item (LBI) file for the selected instance of a library item in the current document. This function is equivalent to clicking Recreate in the Property inspector.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.refreshSiteAssets()

Availability

Dreamweaver 4.

Description

Scans the site, switches to the Site list, and populates the list.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.removeFromFavorites()

Availability

Dreamweaver 4.

Description

Removes the selected elements from the Favorites list. This function does not delete the actual file on disk, except in the case of a library or template where the user is prompted before the file is deleted. It works only in the Favorites list or if the category is Library or Templates.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.renameNickname()

Availability

Dreamweaver 4.

Description

Edits the folder name or the file's nickname by displaying a text box around the existing nickname. It is available only in the Favorites list or in the Library or Template category.

Arguments

None.

Returns

Nothing.

dreamweaver.assetPalette.setSelectedCategory()

Availability

Dreamweaver 4.

Description

Switches to show a different category.

Arguments

`categoryType`

- The *`categoryType`* argument can be one of the following categories: `"templates"`, `"library"`, `"images"`, `"movies"`, `"shockwave"`, `"flash"`, `"scripts"`, `"colors"`, or `"urls"`.

Returns

Nothing.

dreamweaver.assetPalette.setSelectedView()

Availability

Dreamweaver 4.

Description

Switches the display to show either the Site list or the Favorites list.

Arguments

`viewType`

- The *`viewType`* argument is a string that can be `"site"` or `"favorites"`.

Returns

Nothing.

dreamweaver.libraryPalette.deleteSelectedItem() (deprecated)

Availability

Dreamweaver 3; deprecated in Dreamweaver 4 in favor of using dreamweaver.assetPalette.setSelectedCategory(), and then calling dreamweaver.assetPalette.removeFromFavorites() .

Description

This function removes the selected library item from the Library panel and deletes its associated Dreamweaver LBI file from the Library folder at the root of the current site. Instances of the deleted item might still exist in pages throughout the site.

Arguments

None.

Returns

Nothing.

dreamweaver.libraryPalette.getSelectedItem() (deprecated)

Availability

Dreamweaver 3; deprecated in 4 in favor of dreamweaver.assetPalette.getSelectedItems() .

Description

This function gets the path of the selected library item.

Arguments

None.

Returns

A string that contains the path of the library item, which is expressed as a file:// URL.

dreamweaver.libraryPalette.newFromDocument() (deprecated)

Availability

Dreamweaver 3; deprecated in Dreamweaver 4 in favor of using dreamweaver.assetPalette.setSelectedCategory(), and then calling dreamweaver.assetPalette.newAsset() .

Description

This function creates a new library item based on the selection in the current document.

Arguments

`bReplaceCurrent`

- The *`bReplaceCurrent`* argument is a Boolean value that indicates whether to replace the selection with an instance of the newly created library item.

Returns

Nothing.

dreamweaver.libraryPalette.recreateFromDocument() (deprecated)

Availability

Dreamweaver 3; deprecated in Dreamweaver 4 in favor of dreamweaver.assetPalette.recreateLibraryFromDocument().

Description

This function creates an LBI file for the selected instance of a library item in the current document. This function is equivalent to clicking Recreate in the Property inspector.

Arguments

None.

Returns

Nothing.

dreamweaver.libraryPalette.renameSelectedItem() (deprecated)

Availability

Dreamweaver 3; deprecated in Dreamweaver 4 in favor of using dreamweaver.assetPalette.setSelectedCategory() with `"library"` as the argument value, and then calling dreamweaver.assetPalette.renameNickname().

Description

This function turns the name of the selected library item into a text field, so the user can rename the selection.

Arguments

None.

Returns

Nothing.

dreamweaver.referencePalette.getFontSize()

Availability

Dreamweaver 4.

Description

Returns the current font size of the Reference panel display region.

Arguments

None.

Returns

The relative font size as `small`, `medium`, or `large`.

dreamweaver.referencePalette.setFontSize()

Availability

Dreamweaver 4.

Description

Changes the font size that appears in the Reference panel.

Arguments

`fontSize`

- The *`fontSize`* argument is one of the following relative sizes: `small`, `medium`, or `large`.

Returns

Nothing.

dreamweaver.templatePalette.deleteSelectedTemplate() (deprecated)

Availability

Dreamweaver 3; deprecated in Dreamweaver 4 in favor of using dreamweaver.assetPalette.setSelectedCategory() with `"templates"` as the argument value, and then calling dreamweaver.assetPalette.removeFromFavorites().

Description

This function deletes the selected template from the Templates folder.

Arguments

None.

Returns

Nothing.

dreamweaver.templatePalette.getSelectedTemplate() (deprecated)

Availability

Dreamweaver 3; deprecated in 4 in favor of dreamweaver.assetPalette.getSelectedItems().

Description

This function gets the path of the selected template.

Arguments

None.

Returns

A string that contains the path of the template, which is expressed as a file:// URL.

dreamweaver.templatePalette.renameSelectedTemplate() (deprecated)

Availability

Dreamweaver 3; deprecated in Dreamweaver 4 in favor of using dreamweaver.assetPalette.setSelectedCategory() with `"templates"` as the argument value, and then calling dreamweaver.assetPalette.renameNickname().

Description

This function turns the name of the selected template into a text field, so the user can rename the selection.

Arguments

None.

Returns

Nothing.

Behavior functions

Behavior functions let you add behaviors to and remove them from an object, find out which behaviors are attached to an object, get information about the object to which a behavior is attached, and so on. Methods of the `dreamweaver.behaviorInspector` object either control or act on only the selection in the Behaviors panel, not the selection in the current document.

dom.addBehavior()

Availability

Dreamweaver 3.

Description

Adds a new event/action pair to the selected element. This function is valid only for the active document.

Arguments

`event`, *`action`*, *`{eventBasedIndex}`*

- The *`event`* argument is the JavaScript event handler that should attach the behavior to the element (for example, `onClick`, `onMouseOver`, or `onLoad`).
- The *`action`* argument is the function call that `applyBehavior()` returns if the action is added using the Behaviors panel (for example, `"MM_popupMsg('Hello World')"`).
- The *`eventBasedIndex`* argument, which is optional, is the position at which this action should be added. The *`eventBasedIndex`* argument is a zero-based index; if two actions already are associated with the specified event, and you specify *`eventBasedIndex`* as 1, this action executes between the other two. If you omit this argument, the action is added after all existing actions for the specified event.

Returns

Nothing.

dom.getBehavior()

Availability

Dreamweaver 3.

Description

Gets the action at the specified position within the specified event. This function acts on the current selection and is valid only for the active document.

Arguments

`event`, *`{eventBasedIndex}`*

- The *`event`* argument is the JavaScript event handler through which the action is attached to the element (for example, `onClick`, `onMouseOver`, or `onLoad`).
- The *`eventBasedIndex`* argument, which is optional, is the position of the action to get. For example, if two actions are associated with the specified event, 0 is first and 1 is second. If you omit this argument, the function returns all the actions for the specified event.

Returns

A string that represents the function call (for example, `"MM_swapImage('document.Image1','document.Image1','foo.gif','#933292969950')"`) or an array of strings if *`eventBasedIndex`* is omitted.

dom.reapplyBehaviors()

Availability

Dreamweaver 3.

Description

Checks to make sure that the functions that are associated with any behavior calls on the specified node are in the `HEAD` section of the document and inserts them if they are missing.

Arguments

`elementNode`

- The *`elementNode`* argument is an element node within the current document. If you omit the argument, Dreamweaver checks all element nodes in the document for orphaned behavior calls.

Returns

Nothing.

dom.removeBehavior()

Availability

Dreamweaver 3.

Description

Removes the action at the specified position within the specified event. This function acts on the current selection and is valid only for the active document.

Arguments

`event`, {*`eventBasedIndex`*}

- The *`event`* argument is the event handler through which the action is attached to the element (for example, `onClick`, `onMouseOver`, or `onLoad`). If you omit this argument, all actions are removed from the element.
- The *`eventBasedIndex`* argument, which is optional, is the position of the action to be removed. For example, if two actions are associated with the specified event, 0 is first and 1 is second. If you omit this argument, all the actions for the specified event are removed.

Returns

Nothing.

dreamweaver.getBehaviorElement()

Availability

Dreamweaver 2.

Description

Gets the DOM object that corresponds to the tag to which the behavior is being applied. This function is applicable only in Behavior action files.

Arguments

None.

Returns

A DOM object or a `null` value. This function returns a `null` value under the following circumstances:

- When the current script is not executing within the context of the Behaviors panel
- When the Behaviors panel is being used to edit a behavior in a timeline
- When the currently executing script is invoked by `dreamweaver.popupAction()`
- When the Behaviors panel is attaching an event to a link wrapper and the link wrapper does not yet exist
- When this function appears outside of an action file

Example

The `dreamweaver.getBehaviorElement()` function can be used in the same way as dreamweaver.getBehaviorTag() to determine whether the selected action is appropriate for the selected HTML tag, except that it gives you access to more information about the tag and its attributes. As shown in the following example, if you write an action that can be applied only to a hypertext link (`A HREF`) that does not target another frame or window, you can use the `getBehaviorElement()` function as part of the function that initializes the user interface for the Parameters dialog box:

```
function initializeUI(){
  var theTag = dreamweaver.getBehaviorElement();
  var CANBEAPPLIED = (theTag.tagName == "A" && ¬
  theTag.getAttribute("HREF") != null && ¬
  theTag.getAttribute("TARGET") == null);
  if (CANBEAPPLIED) {
    // display the action UI
  } else{
    // display a helpful message that tells the user
    // that this action can only be applied to a
    // hyperlink without an explicit target]
  }
}
```

dreamweaver.getBehaviorEvent() (deprecated)

Availability

Dreamweaver 1.2; deprecated in Dreamweaver 2 because actions are now selected before events.

Description

In a Behavior action file, this function gets the event that triggers this action.

Arguments

None.

Returns

A string that represents the event. This is the same string that is passed as an argument (*`event`*) to the `canAcceptBehavior()` function.

dreamweaver.getBehaviorTag()

Availability

Dreamweaver 1.2.

Description

Gets the source of the tag to which the behavior is being applied. This function is applicable only in action files.

Arguments

None.

Returns

A string that represents the source of the tag. This is the same string that passes as an argument (*`HTMLelement`*) to the `canAcceptBehavior()` function. If this function appears outside an action file, the return value is an empty string.

Example

If you write an action that can be applied only to a hypertext link (`A HREF`), you can use the `getBehaviorTag()` function, as the following example shows, in the function that initializes the user interface for the Parameters dialog box:

```
function initializeUI(){
  var theTag = dreamweaver.getBehaviorTag().toUpperCase();
  var CANBEAPPLIED = (theTag.indexOf('HREF') != -1));
  if (CANBEAPPLIED) {
    // display the action UI
  } else{
    // display a helpful message that tells the user
    // that this action can only be applied to a
    // hyperlink
  }
}
```

dreamweaver.popupAction()

Availability

Dreamweaver 2.

Description

Invokes a Parameters dialog box for the specified behavior action. To the user, the effect is the same as selecting the action from the Actions pop-up menu in the Behaviors panel. This function lets extension files other than actions attach behaviors to objects in the user's document. It blocks other edits until the user dismisses the dialog box.

Note: This function can be called within the `objectTag()` function or in any script in a command file or in the Property inspector file.

Arguments

`actionName`, *`{funcCall}`*

- The *`actionName`* argument is a string that contains the name of a file in the Configuration/Behaviors/Actions folder that contains a JavaScript behavior action (for example, `"Timeline/Play Timeline.htm"`).
- The *`funcCall`* argument, which is optional, is a string that contains a function call for the action that is specified in `actionName`; for example, `"MM_playTimeline(...)"`. This argument, if specified, is supplied by the `applyBehavior()` function in the action file.

Returns

The function call for the behavior action. When the user clicks OK in the Parameters dialog box, the behavior is added to the current document (the appropriate functions are added to the `HEAD` section of the document, HTML might be added to the top of the `BODY` section, and other edits might be made to the document). The function call (for example, `"MM_playTimeline(...)"`) is not added to document but becomes the return value of this function.

dreamweaver.behaviorInspector.getBehaviorAt()

Availability

Dreamweaver 3.

Description

Gets the event/action pair at the specified position in the Behaviors panel.

Arguments

positionIndex

- The *positionIndex* argument is the position of the action in the Behaviors panel. The first action in the list is at position 0.

Returns

An array of two items:

- An event handler
- A function call or JavaScript statement

Example

Because `positionIndex` is a zero-based index, if the Behaviors panel displays the list, a call to the `dreamweaver.behaviorInspector.getBehaviorAt(2)` function returns an array that contains two strings: `"onMouseOver"` and `"MM_changeProp('document.moon','document.moon','src','sun.gif','MG')"`.

dreamweaver.behaviorInspector.getBehaviorCount()

Availability

Dreamweaver 3.

Description

Counts the number of actions that are attached to the currently selected element through event handlers.

Arguments

None.

Returns

An integer that represents the number of actions that are attached to the element. This number is equivalent to the number of actions that are visible in the Behaviors panel and includes Dreamweaver behavior actions and custom JavaScript.

Example

A call to `dreamweaver.behaviorInspector.getBehaviorCount()` for the selected link `<A HREF="javascript:setCookie()" onClick="MM_popupMsg('A cookie has been set.');parent.rightframe.location.href='aftercookie.html'">` returns 2.

dreamweaver.behaviorInspector.getSelectedBehavior()

Availability

Dreamweaver 3.

Description

Gets the position of the selected action in the Behaviors panel.

Arguments

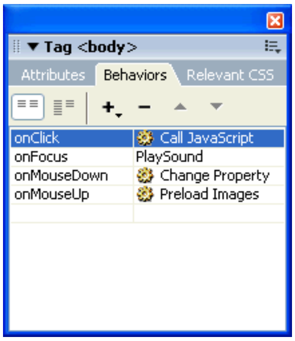
None.

Returns

An integer that represents the position of the selected action in the Behaviors panel, or –1 if no action is selected.

Example

If the first action in the Behaviors panel is selected, as shown in the following figure, a call to the `dreamweaver.behaviorInspector.getSelectedBehavior()` function returns 0:



onClick	Call JavaScript
onFocus	PlaySound
onMouseDown	Change Property
onMouseUp	Preload Images

dreamweaver.behaviorInspector.moveBehaviorDown()

Availability

Dreamweaver 3.

Description

Moves a behavior action lower in sequence by changing its execution order within the scope of an event.

Arguments

`positionIndex`

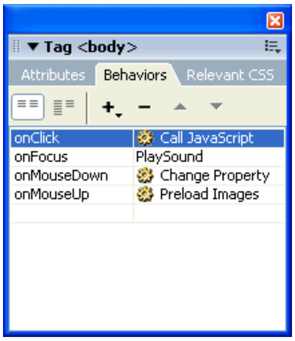
- The *`positionIndex`* argument is the position of the action in the Behaviors panel. The first action in the list is at position 0.

Returns

Nothing.

Example

If the Behaviors panel is set up as shown in the following figure, calling the `dreamweaver.behaviorInspector.moveBehaviorDown(2)` function swaps the positions of the Preload Images and the Change Property actions on the `onMouseDown` event. Calling the `dreamweaver.behaviorInspector.moveBehaviorDown()` function for any other position has no effect because the `onClick` and `onFocus` events each have only one associated behavior, and the behavior at position 3 is already at the bottom of the `onMouseDown` event group.



onClick	Call JavaScript
onFocus	PlaySound
onMouseDown	Change Property
onMouseUp	Preload Images

dreamweaver.behaviorInspector.moveBehaviorUp()

Availability

Dreamweaver 3.

Description

Moves a behavior higher in sequence by changing its execution order within the scope of an event.

Arguments

`positionIndex`

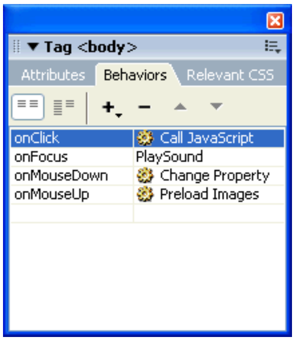
- The *`positionIndex`* argument is the position of the action in the Behaviors panel. The first action in the list is at position 0.

Returns

Nothing.

Example

If the Behaviors panel is set up as shown in the following figure, calling the `dreamweaver.behaviorInspector.moveBehaviorUp(3)` function swaps the positions of the Preload Images and the Change Property actions on the `onMouseOver` event. Calling the `dreamweaver.behaviorInspector.moveBehaviorUp()` function for any other position has no effect because the `onClick` and `onFocus` events each have only one associated behavior, and the behavior at position 2 is already at the top of the `onMouseDown` event group.



onClick	Call JavaScript
onFocus	PlaySound
onMouseDown	Change Property
onMouseUp	Preload Images

dreamweaver.behaviorInspector.setSelectedBehavior()

Availability

Dreamweaver 3.

Description

Selects the action at the specified position in the Behaviors panel.

Arguments

`positionIndex`

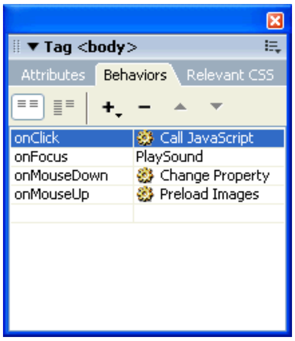
- The *`positionIndex`* argument is the position of the action in the Behaviors panel. The first action in the list is at position 0. To deselect all actions, specify a *`positionIndex`* of –1. Specifying a position for which no action exists is equivalent to specifying –1.

Returns

Nothing.

Example

If the Behaviors panel is set up as shown in the following figure, calling the `dreamweaver.behaviorInspector.setSelection(2)` function selects the Change Property action that is associated with the `onMouseDown` event:



onClick	Call JavaScript
onFocus	PlaySound
onMouseDown	Change Property
onMouseUp	Preload Images

Clipboard functions

Clipboard functions are related to cutting, copying, and pasting. On the Macintosh, some Clipboard functions can also apply to text fields in dialog boxes and floating panels. Functions that can operate in text fields are implemented as methods of the `dreamweaver` object and as methods of the DOM object. The `dreamweaver` version of the function operates on the selection in the active window: the current Document window, the Code inspector, or the Site panel. On the Macintosh, the function can also operate on the selection in a text box if it is the current field. The DOM version of the function always operates on the selection in the specified document.

dom.clipCopy()

Availability

Dreamweaver 3.

Description

Copies the selection, including any HTML markup that defines the selection, to the Clipboard.

Arguments

None.

Returns

Nothing.

dom.clipCopyText()

Availability

Dreamweaver 3.

Description

Copies the selected text to the Clipboard, ignoring any HTML markup.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canClipCopyText()" on page 432.

dom.clipCut()

Availability

Dreamweaver 3.

Description

Removes the selection, including any HTML markup that defines the selection, to the Clipboard.

Arguments

None.

Returns

Nothing.

dom.clipPaste()

Availability

Dreamweaver 3.

Description

Pastes the contents of the Clipboard into the current document at the current insertion point or in place of the current selection. If the Clipboard contains HTML, it is interpreted as such.

Arguments

None.

Returns

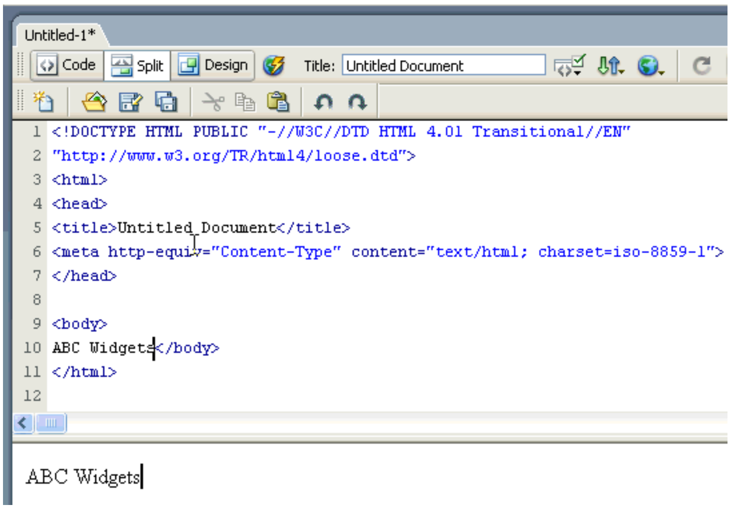
Nothing.

Enabler

"dom.canClipPaste()" on page 433.

Example

If the Clipboard contains `ABC Widgets`, a call to `dw.getDocumentDOM().clipPaste()` results in the following figure:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
ABC Widgets</body>
</html>
```

ABC Widgets

dom.clipPasteText()

Availability

Dreamweaver 3.

Description

Pastes the contents of the Clipboard into the current document at the insertion point or in place of the current selection. It replaces any linefeeds in the Clipboard content with `BR` tags. If the Clipboard contains HTML, it is not interpreted; angle brackets are pasted as `&lt;` and `&gt;`.

Arguments

None.

Returns

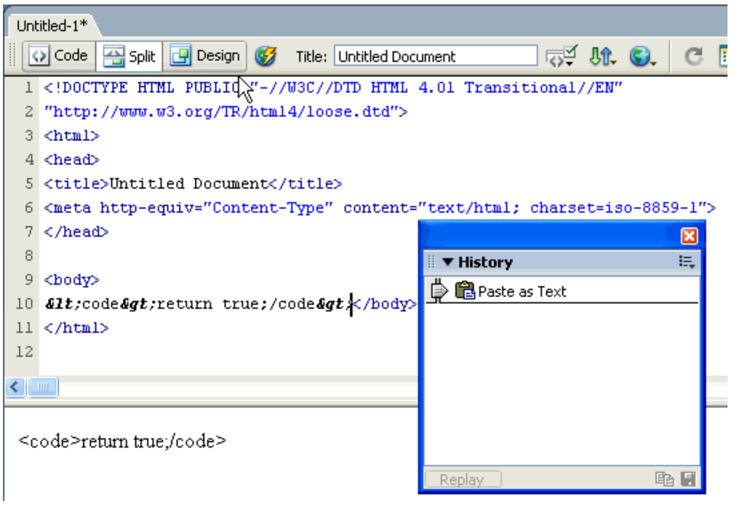
Nothing.

Enabler

"dom.canClipPasteText()" on page 433.

Example

If the Clipboard contains `<code>return true;</code>`, a call to `dw.getDocumentDOM().clipPasteText()` results in the following figure:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
&lt;code&gt;return true;/code&gt;</body>
</html>
```

<code>return true;/code>

dreamweaver.clipCopy()

Availability

Dreamweaver 3.

Description

Copies the current selection from the active Document window, dialog box, floating panel, or Site panel to the Clipboard.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canClipCopy()" on page 442.

dreamweaver.clipCut()

Availability

Dreamweaver 3.

Description

Removes the selection from the active Document window, dialog box, floating panel, or Site panel to the Clipboard.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canClipCut()" on page 443.

dreamweaver.clipPaste()

Availability

Dreamweaver 3.

Description

Pastes the contents of the Clipboard into the current document, dialog box, floating panel, or Site panel.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canClipPaste()" on page 443.

dreamweaver.getClipboardText()

Availability

Dreamweaver 3.

Description

Gets all the text that is stored on the Clipboard.

Arguments

{bAsText}

- The *bAsText* Boolean value, which is optional, specifies whether the Clipboard content is retrieved as text. If *bAsText* is a value of `true`, the Clipboard content is retrieved as text. If *bAsText* is a value of `false`, the content retains formatting. This argument defaults to `false`.

Returns

A string that contains the contents of the Clipboard, if the Clipboard contains text (which can be HTML); otherwise, it returns nothing.

Example

If `dreamweaver.getClipboardText()` returns `"text <b>bold</b> text"`, then `dreamweaver.getClipboardText(true)` returns `"text bold text"`.

Library and template functions

Library and template functions handle operations that are related to library items and templates, such as creating, updating, and breaking links between a document and a template or library item. Methods of the `dreamweaver.libraryPalette` object either control or act on the selection in the Assets panel library items, not in the current document. Likewise, methods of the `dreamweaver.templatePalette` object control or act on the selection in the Assets panel template objects.

dom.applyTemplate()

Availability

Dreamweaver 3.

Description

Applies a template to the current document. If no argument is supplied, the Select Template dialog box appears. This function is valid only for the document that has focus.

Arguments

{templateURL}, *bMaintainLink*

- The *templateURL* argument is the path to a template in the current site, which is expressed as a file:// URL.
- The *bMaintainLink* argument is a Boolean value that indicates whether to maintain the link to the original template (`true`) or not (`false`).

Returns

Nothing.

Enabler

"dom.canApplyTemplate()" on page 432.

dom.detachFromLibrary()

Availability

Dreamweaver 3.

Description

Detaches the selected library item instance from its associated LBI file by removing the locking tags from around the selection. This function is equivalent to clicking Detach from Original in the Property inspector.

Arguments

None.

Returns

Nothing.

dom.detachFromTemplate()

Availability

Dreamweaver 3.

Description

Detaches the current document from its associated template.

Arguments

None.

Returns

Nothing.

dom.getAttachedTemplate()

Availability

Dreamweaver 3.

Description

Gets the path of the template that is associated with the document.

Arguments

None.

Returns

A string that contains the path of the template, which is expressed as a file:// URL.

dom.getEditableRegionList()

Availability

Dreamweaver 3.

Description

Gets a list of all the editable regions in the body of the document.

Arguments

None.

Returns

An array of element nodes.

Example

"dom.getSelectedEditableRegion()" on page 320.

dom.getIsLibraryDocument()

Availability

Dreamweaver 3.

Description

Determines whether the document is a library item.

Arguments

None.

Returns

A Boolean value that indicates whether the document is an LBI file.

dom.getIsTemplateDocument()

Availability

Dreamweaver 3.

Description

Determines whether the document is a template.

Arguments

None.

Returns

A Boolean value that indicates whether the document is a DWT file.

dom.getSelectedEditableRegion()

Availability

Dreamweaver 3.

Description

If the selection or insertion point is inside an editable region, this function gets the position of the editable region among all others in the body of the document.

Arguments

None.

Returns

An index into the array that the `dom.getEditbableRegionList()` function returns. For more information, see "dom.getEditableRegionList()" on page 319.

Example

The following code shows a dialog box with the contents of the selected editable region:

```
var theDOM = dw.getDocumentDOM();
var edRegs = theDOM.getEditableRegionList();
var selReg = theDOM.getSelectedEditableRegion();
alert(edRegs[selReg].innerHTML);
```

dom.insertLibraryItem()

Availability

Dreamweaver 3.

Description

Inserts an instance of a library item into the document.

Arguments

`libraryItemURL`

- The *`libraryItemURL`* argument is the path to an LBI file, which is expressed as a file:// URL.

Returns

Nothing.

dom.markSelectionAsEditable()

Availability

Dreamweaver 3.

Description

Displays the New Editable Region dialog box. When the user clicks New Region, Dreamweaver marks the selection as editable and doesn't change any text.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canMarkSelectionAsEditable()" on page 437.

dom.newEditableRegion()

Availability

Dreamweaver 3.

Description

Displays the New Editable Region dialog box. When the user clicks New Region, Dreamweaver inserts the name of the region, surrounded by curly braces ({ }), into the document at the insertion point location.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canMakeNewEditableRegion()" on page 437.

dom.removeEditableRegion()

Availability

Dreamweaver 3.

Description

Removes an editable region from the document. If the editable region contains any content, the content is preserved; only the editable region markers are removed.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canRemoveEditableRegion()" on page 439.

dom.updateCurrentPage()

Availability

Dreamweaver 3.

Description

Updates the document's library items, templates, or both. This function is valid only for the active document.

Arguments

{typeOfUpdate}

- The optional *`typeOfUpdate`* argument must be `"library"`, `"template"`, or `"both"`. If you omit the argument, the default is `"both"`.

Returns

Nothing.

dreamweaver.updatePages()

Availability

Dreamweaver 3.

Description

Opens the Update Pages dialog box and selects the specified options.

Arguments

{typeOfUpdate}

- The optional *`typeOfUpdate`* argument must be `"library"`, `"template"`, or `"both"`, if you specify it. If the argument is omitted, it defaults to `"both"`.

Returns

Nothing.

Snippets panel functions

Using Dreamweaver, web developers can edit and save reusable blocks of code in the Snippets panel and retrieve them as needed.

The Snippets panel stores each code snippet in a CSN file within the Configuration/Snippets folder. Snippets that come with Dreamweaver are stored in the following folders:

- Accessible
- Comments
- Content_tables
- Filelist.txt
- Footers
- Form_elements
- Headers
- Javascript
- Meta
- Navigation
- Text

Snippet files are XML documents, so you can specify the encoding in the XML directive, as shown in the following example:

```
<?XML version="1.0" encoding="utf-8">
```

The following sample shows a snippet file:

```
<snippet name="Detect Flash" description="VBscript to check for Flash ActiveX
  control" preview="code" factory="true" type="wrap" >
     <insertText location="beforeSelection">
          <![CDATA[ ------- code --------- ]]>
     </insertText>
     <insertText location="afterSelection">
          <![CDATA[ ------- code --------- ]]>
     </insertText>
</snippet>
```

Snippet tags in CSN files have the following attributes:

Attribute	Description
`name`	Name of snippet
`description`	Snippet description
`preview`	Type of preview: `"code"` to display the snippet in preview area or `"design"` to display the snippet rendered in HTML in the Preview area.
`type`	If the snippet is used to wrap a user selection, `"wrap"`; if the snippet should be inserted before the selection, `"block"`.

You can use the following methods to add Snippets panel functions to your extensions.

dreamweaver.snippetPalette.getCurrentSnippetPath()

Availability

Dreamweaver MX 2004.

Description

Returns the path to the snippet that is currently selected in the Snippets panel.

Arguments

None.

Returns

The path, relative to the Snippets folder, to the snippet selected in the Snippets panel. Returns an empty string if no snippet is selected.

dreamweaver.snippetPalette.newFolder()

Availability

Dreamweaver MX.

Description

Creates a new folder with the default name *`untitled`* and puts an text box around the default name.

Arguments

None.

Returns

Nothing.

dreamweaver.snippetPalette.newSnippet()

Availability

Dreamweaver MX.

Description

Opens the Add Snippet dialog box and gives it focus.

Arguments

None.

Returns

Nothing.

dreamweaver.snippetPalette.editSnippet()

Availability

Dreamweaver MX.

Description

Opens the Edit Snippet dialog box and gives it focus, enabling editing for the selected element.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.snippetpalette.canEditSnippet()" on page 454.

dreamweaver.snippetPalette.insert()r

Availability

Dreamweaver MX.

Description

Applies the selected snippet from the Snippets panel to the current selection.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.snippetpalette.canInsert()" on page 455.

dreamweaver.snippetPalette.insertSnippet()

Availability

Dreamweaver MX.

Description

Inserts the indicated snippet into the current selection.

Arguments

- A string that specifies the path to the snippet relative to the Snippets folder.

Returns

A Boolean value.

Enabler

"dreamweaver.snippetpalette.canInsert()" on page 455.

Example

The following call to the `dw.snippetPalette.insertSnippet()` function inserts the code snippet at the location specified by the argument into the current document at the insertion point:

```
dw.snippetPalette.insertSnippet('Text\\Different_Link_Color.csn');
```

dreamweaver.snippetPalette.rename()

Availability

Dreamweaver MX.

Description

Activates a text box around the selected folder name or file nickname and lets you edit the selected element.

Arguments

None.

Returns

Nothing.

dreamweaver.snippetPalette.remove()

Availability

Dreamweaver MX.

Description

Deletes the selected element or folder from the Snippets panel and deletes the file from the disk.

Returns

Nothing.

CHAPTER 16
Dynamic Documents

The dynamic documents functions in Macromedia Dreamweaver MX 2004 perform operations that are related to web server pages. These operations include returning a property for the selected node in the Components panel, getting a list of all data sources in the user's document, displaying dynamic content in Design view, applying a server behavior to a document, or getting the names of all currently defined server models.

Server Components functions

Server Components functions let you access the currently selected node of the Server Components tree control that appears in the Components panel. Using these functions, you can also refresh the view of the Components tree.

dreamweaver.serverComponents.getSelectedNode()

Availability

Dreamweaver MX.

Description

Returns the currently selected `ComponentRec` property in the Server Components tree control.

Arguments

None.

Returns

The `ComponentRec` property.

dreamweaver.serverComponents.refresh()

Availability

Dreamweaver MX.

Description

Refreshes the view of the Components tree.

Arguments

None.

Returns

Nothing.

Data source functions

Data source files are stored in the Configuration/DataSources folder. Each server model has its own folder: ASP.Net/C#, ASP.Net/VisualBasic, ASP/JavaScript, ASP/VBScript, ColdFusion, JSP, and PHP/MySQL. Each server model subfolder contains HTML and EDML files that are associated with the data sources for that server model.

For more information about using data sources in Dreamweaver, see "Data Sources" in *Extending Dreamweaver*.

dreamweaver.dbi.getDataSources

Availability

Dreamweaver UltraDev 4.

Description

Calls the `findDynamicSources()` function for each file in the Configuration/DataSources folder. You can use this function to generate a list of all the data sources in the user's document. This function iterates through all the files in the Configuration/DataSources folder, calls the `findDynamicSources()` function in each file, concatenates all the returned arrays, and returns the concatenated array of data sources.

Arguments

None.

Returns

An array that contains a concatenated list of all the data sources in the user's document. Each element in the array is an object, and each object has the following properties:

- The `title` property is the label string that appears to the right of the icon for each parent node. The `title` property is always defined.
- The `imageFile` property is the path of a file that contains the icon (a GIF image) that represents the parent node in Dynamic Data or the Dynamic Text dialog box or in the Bindings panel. The `imageFile` property is always defined.

- The `allowDelete` property is optional. If this property is set to a value of `false`, when the user clicks on this node in the Bindings panel, the Minus (-) button is disabled. If it is set to a value of `true`, the Minus (-) button is enabled. If the property is not defined, the Minus (-) button is enabled when the user clicks on the item (as if the property is set to a value of `true`).
- The `dataSource` property is the simple name of the file in which the `findDynamicSources()` function is defined. For example, the `findDynamicSources()` function in the Session.htm file, which is located in the Configuration/DataSources/ASP_Js folder, sets the `dataSource` property to `session.htm`. This property is always defined.
- The `name` property is the name of the server behavior associated with the data source, `dataSource`, if one exists. The `name` property is always defined but can be an empty string (`""`) if no server behavior is associated with the data source (such as a session variable).

Extension Data Manager functions

The APIs in this section comprise the Extension Data Manager (EDM). You can programmatically access and manipulate the data that is contained in the group and participant files by calling these functions. The EDM performs in the following manner:

- The EDM performs all EDML file input/output for group and participant files.
- The EDM acts as a server model filter by performing all data requests for the current server model.

dreamweaver.getExtDataValue()

Availability

Dreamweaver UltraDev 4.

Description

This function retrieves the field values from an EDML file for the specified nodes.

Arguments

`qualifier(s)`

- The *`qualifier(s)`* argument is a variable-length list (depending on the level of information you need) of comma-separated node qualifiers that includes group or participant name, subblock (if any), and field name.

Returns

Dreamweaver expects a field value. If a value is not specified, Dreamweaver uses the default value.

Example

The following example retrieves the location attribute value for the insertText tag of the recordset_main participant:

```
dw.getExtDataValue("recordset_main", "insertText", "location");
```

dreamweaver.getExtDataArray()

Availability

Dreamweaver UltraDev 4.

Description

This function retrieves an array of values from an EDML file for the specified nodes.

Arguments

`qualifier(s)`

- The *`qualifier(s)`* argument is a variable-length list of comma-separated node qualifiers, including group or participant name, subblock (if any), and field name.

Returns

Dreamweaver expects an array of child-node names.

dreamweaver.getExtParticipants()

Availability

Dreamweaver UltraDev 4.

Description

This function retrieves the list of participants from an EDML group file or participant files.

Arguments

`value`, *`qualifier(s)`*

- The *`value`* argument is a property value, or it is blank and is ignored. For example `dreamweaver.getExtParticipants("", "participant");`
- The *`qualifier(s)`* argument is a variable-length list of comma-separated node qualifiers of the required property.

Returns

Dreamweaver expects an array of participant names that have the specified property, if it is given, and the property matches the specified value, if it is given.

dreamweaver.getExtGroups()

Availability

Dreamweaver UltraDev 4.

Description

Retrieves the name of the group, which is the equivalent to the server behavior's name, from an EDML group file.

Arguments

`value`, `qualifier(s)`

- The `value` argument is a property value or is blank to ignore.
- The `qualifier(s)` argument is a variable length list of comma-separated node qualifiers of the required property.

Returns

Dreamweaver expects an array of group names that have the specified property, if it is given, and the property matches the specified value, if it is given.

dreamweaver.refreshExtData()

Availability

Dreamweaver UltraDev 4.

Description

Reloads all extension data files.

Tip: You can make a useful command from this function, letting edits to server-behavior EDML files be reloaded without restarting Dreamweaver.

Arguments

None.

Returns

Dreamweaver expects reloaded data.

Live data functions

You can use the following live data functions to mimic menu functionality:

- The `showLiveDataDialog()` function is used for the View > Live Data Settings menu item.
- The `setLiveDataMode()` function is used for the View > Live Data and View > Refresh Live Data menu items.
- The `getLiveDataMode()` function determines whether Live Data mode is active.

You can use the remaining live data functions when you implement the translator API `liveDataTranslateMarkup()` function.

dreamweaver.getLiveDataInitTags()

Availability

Dreamweaver UltraDev 1.

Description

Returns the initialization tags for the currently active document. The initialization tags are the HTML tags that the user supplies in the Live Data Settings dialog box. This function is typically called from a translator's `liveDataTranslateMarkup()` function, so that the translator can pass the tags to the `liveDataTranslate()` function.

Arguments

None.

Returns

A string that contains the initialization tags.

dreamweaver.getLiveDataMode()

Availability

Dreamweaver UltraDev 1.

Description

Determines whether the Live Data window is currently visible.

Arguments

None.

Returns

A Boolean value: `true` if the Live Data window is visible; false otherwise.

dreamweaver.getLiveDataParameters ()

Availability

Dreamweaver MX.

Description

Obtains the URL parameters that are specified as Live Data settings.

Live Data mode lets you view a web page in the design stage (as if it has been translated by the application server and returned). Generating dynamic content to display in Design view lets you view your page layout with live data and adjust it, if necessary.

Before you view live data, you must enter Live Data settings for any URL parameters that you reference in your document. This prevents the web server from returning errors for parameters that are otherwise undefined in the simulation.

You enter the URL parameters in name-value pairs. For example, if you reference the URL variables `ID` and `Name` in server scripts in your document, you must set these URL parameters before you view live data.

You can enter Live Data settings through Dreamweaver in the following ways:

- Through the Live Data Settings dialog box, which you can activate from the View menu
- In the URL text field that appears at the top of the document when you click the Live Data View button on the toolbar

For the `ID` and `Name` parameters, you can enter the following pairs:

```
ID     22
Name   Samuel
```

In the URL, these parameters would appear as shown in the following example:

```
http://someURL?ID=22&Name=Samuel
```

This function lets you obtain these live data settings through JavaScript.

Arguments

None.

Returns

An array that contains the URL parameters for the current document. The array contains an even number of parameter strings. Each two elements form a URL parameter name-value pair. The even element is the parameter name and the odd element is the value. For example, `getLiveDataParameters()` returns the following array for the `ID` and `Name` parameters in the preceding example: `['ID,'22','Name','Samuel']`.

Example

The following example returns the parameters that are specified as Live Data settings and stores them in the `paramsArray`:

```
var paramsArray = dreamweaver.getLiveDataParameters();
```

dreamweaver.liveDataTranslate()

Availability

Dreamweaver UltraDev 1.

Description

Sends an entire HTML document to an application server, asks the server to execute the scripts in the document, and returns the resulting HTML document. This function can be called only from a translator's `liveDataTranslateMarkup()` function; if you try to call it at another time, an error occurs. The `dreamweaver.liveDataTranslate()` function performs the following operations:

- Makes the animated image (that appears near the right edge of the Live Data window) play.
- Listens for user input. If the Stop icon is clicked, the function returns immediately.
- Accepts a single string argument from the caller. (This string is typically the entire source code of the user's document. It is the same string that is used in the next operation.)
- Saves the HTML string from the user's document as a temporary file on the live data server.
- Sends an HTTP request to the live-data server, using the parameters specified in the Live Data Settings dialog box.
- Receives the HTML response from the live data server.
- Removes the temporary file from the live data server.
- Makes the animated image stop playing.
- Returns the HTML response to the caller.

Arguments

- A single string, which typically is the entire source code of the user's current document.

Returns

An `httpReply` object. This object is the same as the value that the `MMHttp.getText()` function returns. If the user clicks the Stop icon, the return value's `httpReply.statusCode` is equal to 200 (Status OK) and its `httpReply.data` is equal to the empty string. For more information on the `httpReply` object, see Chapter 3, "The HTTP API," on page 43.

dreamweaver.setLiveDataError()

Availability

Dreamweaver UltraDev 1.

Description

Specifies the error message that appears if an error occurs while the `liveDataTranslateMarkup()` function executes in a translator. If the document that Dreamweaver passed to `liveDataTranslate()` contains errors, the server passes back an error message that is formatted using HTML. If the translator (the code that called `liveDataTranslate()`) determines that the server returned an error message, it calls `setLiveDataError()` to display the error message in Dreamweaver. This message appears after the `liveDataTranslateMarkup()` function finishes executing; Dreamweaver displays the description in an error dialog box. The `setLiveDataError()` function should be called only from the `liveDataTranslateMarkup()` function.

Arguments

`source`

- The *`source`* argument is a string that contains source code, which is parsed and rendered in the error dialog box.

Returns

Nothing.

dreamweaver.setLiveDataMode()

Availability

Dreamweaver UltraDev 1.

Description

Toggles the visibility of the Live Data window.

Arguments

`bIsVisible`

- The *`bIsVisible`* argument is a Boolean value that indicates whether the Live Data window should be visible. If you pass `true` to this function and Dreamweaver currently displays the Live Data window, the effect is the same as if you clicked the Refresh button.

Returns

Nothing.

dreamweaver.setLiveDataParameters()

Availability

Dreamweaver MX.

Description

Sets the URL parameters that you reference in your document for use in Live Data mode.

Live Data mode lets you view a web page in the design stage (as if it has been translated by the application server and returned). Generating dynamic content to display in Design view lets you view your page layout with live data and adjust it, if necessary.

Before you view live data, you must enter Live Data settings for any URL parameters that you reference in your document. This prevents the web server from returning errors for parameters that are otherwise undefined in the simulation.

You enter the URL parameters in name-value pairs. For example, if you reference the URL variables `ID` and `Name` in server scripts in your document, you must set these URL parameters before you view live data.

This function lets you set Live Data values through JavaScript.

Arguments

`liveDataString`

- The *`liveDataString`* argument is a string that contains the URL parameters that you want to set, in name-value pairs.

Returns

Nothing.

Example

```
dreamweaver.setLiveDataParameters("ID=22&Name=Samuel")
```

dreamweaver.showLiveDataDialog()

Availability

Dreamweaver UltraDev 1.

Description

Displays the Live Data Settings dialog box.

Arguments

None.

Returns

Nothing.

Server behavior functions

Server behavior functions let you manipulate the Server Behaviors panel, which you can display by selecting Window > Server Behaviors. Using these functions, you can find all the server behaviors on a page and programmatically apply a new behavior to the document or modify an existing behavior.

Note: You can abbreviate `dw.serverBehaviorInspector` to `dw.sbi`.

dreamweaver.getParticipants()

Availability

Dreamweaver UltraDev 4.

Description

The JavaScript function, `dreamweaver.getParticipants()`, gets a list of participants from the user's document. After Dreamweaver finds all the behavior's participants, it stores those lists. Typically, you use this function with the `findServerBehaviors()` function (for more information, see "Server Behaviors" in *Extending Dreamweaver*) to locate instances of a behavior in the user's document.

Arguments

`edmlFilename`

- The *`edmlFilename`* argument is the name of the group or participant file that contains the names of the participants to locate in the user's document. This string is the filename, without the .edml extension.

Returns

This function returns an array that contains all instances of the specified participant (or, in the case of a group file, any instance of any participant in the group) that appear in the user's document. The array contains JavaScript objects, with one element in the array for each instance of each participant that is found in the user's document. The array is sorted in the order that the participants appear in the document. Each JavaScript object has the following properties:

- The *`participantNode`* property is a pointer to the participant node in the user's document.
- The *`participantName`* property is the name of the participant's EDML file (without the .edml extension).
- The *`parameters`* property is a JavaScript object that stores all the parameter/value pairs.
- The *`matchRangeMin`* property defines the character offset from the participant node of the document to the beginning of the participant content.
- The *`matchRangeMax`* property is an integer of the participant that defines the offset from the beginning of the participant node to the last character of the participant content.

dreamweaver.serverBehaviorInspector.getServerBehaviors()

Availability

Dreamweaver UltraDev 1.

Description

Gets a list of all the behaviors on the page. When Dreamweaver determines that the internal list of server behaviors might be out of date, it calls the `findServerBehaviors()` function for each currently installed behavior. Each function returns an array. Dreamweaver merges all the arrays into a single array and sorts it, based on the order that each behavior's `selectedNode` object appears in the document. Dreamweaver stores the merged array internally. The `getServerBehaviors()` function returns a pointer to that merged array.

Arguments

None.

Returns

An array of JavaScript objects. The `findServerBehaviors()` call returns the objects in the array. The objects are sorted in the order that they appear in the Server Behaviors panel.

dreamweaver.popupServerBehavior()

Availability

Dreamweaver UltraDev 1.

Description

Applies a new server behavior to the document or modifies an existing behavior. If the user must specify parameters for the behavior, a dialog box appears.

Arguments

`{behaviorName` or *`behaviorObject}`*

- The *`behaviorName`* argument, which is optional, is a string that represents the behavior's name, the title tag of a file, or a filename.
- The *`behaviorObject`* argument, which is optional, is a behavior object.

If you omit the argument, Dreamweaver runs the currently selected server behavior. If the argument is the name of a server behavior, Dreamweaver adds the behavior to the page. If the argument is one of the objects in the array that the `getServerBehaviors()` function returns, a dialog box appears so the user can modify the parameters for the behavior.

Returns

Nothing.

Server model functions

In Macromedia Dreamweaver, each document has an associated document type. For dynamic document types, Dreamweaver also associates a server model (such as ASP-JS, ColdFusion, or PHP-MySQL).

Server models are used to group functionality that is specific to a server technology. Different server behaviors, data sources, and so forth, appear based on the server model that is associated with the document.

Using the server model functions, you can determine the set of server models that are currently defined; the name, language, and version of the current server model; and whether the current server model supports a named character set (such as UTF-8).

Note: Dreamweaver reads all the information in the server model HTML file and stores this information when it first loads the server model. So, when an extension calls functions such as `dom.serverModel.getServerName()`, `dom.serverModel.getServerLanguage()`, and `dom.serverModel.getServerVersion()`, these functions return the stored values.

dom.serverModel.getAppURLPrefix()

Availability

Dreamweaver MX.

Description

Returns the URL for the site's root folder on the testing server. This URL is the same as that specified for the Testing Server on the Advanced tab in the Site Definition dialog box.

When Dreamweaver communicates with your testing server, it uses HTTP (the same way as a browser). When doing so, it uses this URL to access your site's root folder.

Arguments

None.

Returns

A string, which holds the URL to the application server that is used for live data and debugging purposes.

Example

If the user creates a site and specifies that the testing server is on the local computer and that the root folder is named `"employeeapp"`, a call to the `dom.serverModel.getAppURLPrefix()` function returns the following string:

```
http://localhost/employeeapp/
```

dom.serverModel.getDelimiters()

Availability

Dreamweaver MX.

Description

Lets JavaScript code get the script delimiter for each server model, so managing the server model code can be separated from managing the user-scripted code.

Arguments

None.

Returns

An array of objects where each object contains the following three properties:

- The *`startPattern`* property is a regular expression that matches the opening script delimiter.
- The *`endPattern`* property is a regular expression that matches the closing script delimiter.
- The *`participateInMerge`* pattern is a Boolean value that specifies whether the content that is enclosed in the listed delimiters should (`true`) or should not (`false`) participate in block merging.

dom.serverModel.getDisplayName()

Availability

Dreamweaver MX.

Description

Gets the name of the server model that appears in the user interface (UI).

Arguments

None.

Returns

A string, the value of which is the name of the server model.

dom.serverModel.getFolderName()

Availability

Dreamweaver MX.

Description

Gets the name of the folder that is used for this server model in the Configuration folder (such as in the ServerModels subfolder).

Arguments

None.

Returns

A string, the value of which is the name of the folder.

dom.serverModel.getServerExtension() (deprecated)

Availability

Dreamweaver UltraDev 4; deprecated in Dreamweaver MX.

Description

Returns the default file extension of files that use the current server model. (The default file extension is the first in the list.) If no user document is currently selected, the `serverModel` object is set to the server model of the currently selected site.

Arguments

None.

Returns

A string that represents the supported file extensions.

dom.serverModel.getServerIncludeUrlPatterns()

Availability

Dreamweaver MX.

Description

Returns the following list of properties, which let you access:

- Translator URL patterns
- File references
- Type

Arguments

None.

Returns

A list of objects, one for each `searchPattern`. Each object has the following three properties:

Property	Description
`pattern`	A JavaScript regular expression that is specified in the `searchPattern` field of an EDML file. (A regular expression is delimited by a pair of forward slashes (`//`).)
`fileRef`	The 1-based index of the regular expression submatch that corresponds to the included file reference.
`type`	The portion of the `paramName` value that remains after removing the _includeUrl suffix. This type is assigned to the type attribute of the `<MM:BeginLock>` tag. For an example, see Server Model SSI.htm in the Configuration/Translators folder.

Example

The following code snippet from a participant file shows a translator `searchPatterns` tag:

```
<searchPatterns whereToSearch="comment">
   <searchPattern paramNames=",ssi_comment_includeUrl">
      <![CDATA[/<!--\s*#include\s+(file|virtual)\s*=\s*"([^"]*)"\s*-->/i]]>
   </searchPattern>
</searchPatterns>
```

The search pattern contains a JavaScript regular expression that specifies two submatches (both of which are contained within parentheses). The first submatch is for the text string `file` or `virtual`. The second submatch is a file reference.

To access the translator URL pattern, your code should look like the following example:

```
var serverModel = dw.getDocumentDOM().serverModel;
var includeArray = new Array();
includeArray = serverModel.getServerIncludeUrlPatterns();
```

The call to `serverModel.getServerIncludeUrlPatterns()` returns the following three properties:

Property	Return value
`pattern`	`/<!--\s*#include\s+(file|virtual)\s*=\s*"([^"]*)"\s*-->/i`
`fileRef`	`2`
`type`	`ssi_comment`

dom.serverModel.getServerInfo()

Availability

Dreamweaver MX.

Description

Returns information that is specific to the current server model. This information is defined in the HTML definition file for the server model, which is located in the Configuration/ ServerModels folder.

You can modify the information in the HTML definition file or place additional variable values or functions in the file. For example, you can modify the `serverName`, `serverLanguage`, and `serverVersion` properties. The `dom.serverModel.getServerInfo()` function returns the information that the server model author adds to the definition file.

Note: The other values that are defined in the default server model files are for internal use only.

The `serverName`, `serverLanguage`, and `serverVersion` properties are special because the developer can access them directly by using the following corresponding functions:

- `dom.serverModel.getServerName()`
- `dom.serverModel.getServerLanguage()`
- `dom.serverModel.getServerVersion()`

Arguments

None.

Returns

A JavaScript object that contains a variety of information that is specific to the current server model.

dom.serverModel.getServerLanguage() (deprecated)

Availability

UltraDev 1; deprecated in Dreamweaver MX.

Description

Determines the server model that is associated with the document and returns that value. The server language for a site is the value that comes from the Default Scripting Language setting on the App Server Info tab in the Site Definition dialog box. To get the return value, this function calls the `getServerLanguage()` function in the Server Models API.

Note: The Default Scripting Language list exists only in Dreamweaver 4 and earlier versions. For Dreamweaver MX or later, the Site Definition dialog box does not list supported scripting languages. Also, for Dreamweaver MX or later, the `dom.serverModel.getServerLanguage()` function reads the `serverLanguage` property of the object that is returned by a call to the `getServerInfo()` function in the Server Models API.

Arguments

None.

Returns

A string that contains the supported scripting languages.

dom.serverModel.getServerName()

Availability

Dreamweaver 1; enhanced in Dreamweaver MX.

Description

Retrieves the server name that is associated with the document and returns that value. The server name differentiates between server technologies (such as ASP.NET and JSP), but does not differentiate between languages on the same server technology (such as ASP.NET VB and ASP.NET C#). Possible values include `ASP`, `ASP.NET`, `ColdFusion`, `JSP`, and `PHP`.

To retrieve the server model name associated with the document, see "dom.serverModel.getDisplayName()" on page 339 or "dom.serverModel.getFolderName()" on page 339.

Note: For Dreamweaver MX, or later, `dom.serverModel.getServerName()` reads the `serverName` property of the object that is returned by a call to the `getServerInfo()` function in the Server Models API.

Arguments

None.

Returns

A string that contains the server name.

dom.serverModel.getServerSupportsCharset()

Availability

Dreamweaver MX.

Description

Determines whether the server model that is associated with the document supports the named character set.

Note: In addition to letting you call this function from the JavaScript layer, Dreamweaver calls this function when the user changes the encoding in the page Properties dialog box. If the server model does not support the new character encoding, this function returns `false` and Dreamweaver pops up a warning dialog box that asks if the user wants to do the conversion. An example of this situation is when a user attempts to convert a ColdFusion 4.5 document to UTF-8 because ColdFusion does not support UTF-8 encoding.

Arguments

`metaCharSetString`

- The *`metaCharSetString`* argument is a string value that names a particular character set. This value is the same as that of the `"charset="` attribute of a `meta` tag that is associated with a document. Supported values for a given server model are defined in the HTML definition file for the server model, which is located in the Configuration/ServerModels folder.

Returns

A Boolean value: `true` if the server model supports the named character set; `false` otherwise.

dom.serverModel.getServerVersion()

Availability

UltraDev 1; enhanced in Dreamweaver MX.

Description

Determines the server model that is associated with the document and returns that value. Each server model has a `getVersionArray()` function, as defined in the Server Models API, which returns a table of name-version pairs.

Note: For Dreamweaver, `dom.serverModel.getServerVersion()` first reads the `serverVersion` property of the object that is returned by a call to `getServerInfo()` in the Server Models API. If that property does not exist, `dom.serverModel.getServerVersion()` reads it from the `getVersionArray()` function.

Arguments

`name`

- The *`name`* argument is a string that represents the name of a server model.

Returns

A string that contains the version of the named server model.

dom.serverModel.testAppServer()

Availability

Dreamweaver MX.

Description

Tests whether a connection to the application server can be made.

Arguments

None.

Returns

A Boolean value that indicates whether the request to connect to the application server was successful.

dreamweaver.getServerModels()

Availability

Dreamweaver MX.

Description

Gets the names for all the currently defined server models. The set of names is the same as the ones that appear in the Server Model text field in the Site Definition dialog box.

Arguments

None.

Returns

An array of strings. Each string element holds the name of a currently defined server model.

CHAPTER 17
Design

The Design functions in Macromedia Dreamweaver MX 2004 perform operations related to designing the appearance of a document. These operations include functions that apply a specified cascading style sheet (CSS) style, split a selected frame vertically or horizontally, align selected layers or hotspots, play a selected plug-in item, create a layout cell, or manipulate table rows or columns.

CSS functions

CSS functions handle applying, removing, creating, and deleting CSS styles. Methods of the `dreamweaver.cssRuleTracker` object either control or act on the selection in the CSS rule tracker panel of the Selection inspector. Methods of the `dreamweaver.cssStylePalette` object either control or act on the selection in the CSS Styles panel, not in the current document.

dom.applyCSSStyle()

Availability

Dreamweaver 4.

Description

Applies the specified style to the specified element. This function is valid only for the active document.

Arguments

`elementNode, styleName, {classOrID}, {bForceNesting}`

- The *`elementNode`* argument is an element node in the DOM. If the *`elementNode`* argument is a `null` value or an empty string (`""`), the function acts on the current selection.
- The *`styleName`* argument is the name of a CSS style.

- The `classOrID` argument, which is optional, is the attribute with which the style should be applied (either `"class"` or `"id"`). If the `elementNode` argument is a `null` value or an empty string and no tag exactly surrounds the selection, the style is applied using `SPAN` tags. If the selection is an insertion point, Dreamweaver uses heuristics to determine to which tag the style should be applied.
- The `bForceNesting` argument, which is optional, is a Boolean value, which indicates whether nesting is allowed. If the `bForceNesting` flag is specified, Dreamweaver inserts a new `SPAN` tag instead of trying to modify the existing tags in the document. This argument defaults to a `false` value if it is not specified.

Returns

Nothing.

Example

The following code applies the `red` style to the selection, either by surrounding the selection with `SPAN` tags or by applying a `CLASS` attribute to the tag that surrounds the selection:

```
var theDOM = dreamweaver.getDocumentDOM('document');
theDOM.applyCSSStyle('','red');
```

dom.removeCSSStyle()

Availability

Dreamweaver 3.

Description

Removes the `CLASS` or `ID` attribute from the specified element, or removes the `SPAN` tag that completely surrounds the specified element. This function is valid only for the active document.

Arguments

`elementNode, {classOrID}`

- The `elementNode` argument is an element node in the DOM. If the `elementNode` argument is specified as an empty string (" "), the function acts on the current selection.
- The `classOrID` argument, which is optional, is the attribute that should be removed (either `"class"` or `"id"`). If the `classOrID` argument is not specified, it defaults to `"class"`. If no `CLASS` attribute is defined for the `elementNode` argument, the `SPAN` tag that surrounds the `elementNode` argument is removed.

Returns

Nothing.

dreamweaver.cssRuleTracker.editSelectedRule()

Availability

Dreamweaver MX 2004.

Description

Lets the user edit the currently selected rule in the rule tracker. This function displays the selected rule in the CSS property grid, and if necessary, will show the property grid and its containing floater.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.cssRuleTracker.canEditSelectedRule()" on page 449.

dreamweaver.cssRuleTracker.newRule()

Availability

Dreamweaver MX 2004.

Description

Opens the New CSS Style dialog box, so the user can create a new rule.

Arguments

None.

Returns

Nothing.

dreamweaver.cssStylePalette.applySelectedStyle()

Availability

Dreamweaver MX.

Description

Applies the selected style to the current active document or to its attached style sheet, depending on the selection in the CSS Styles panel.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.cssStylePalette.canApplySelectedStyle()" on page 449.

dreamweaver.cssStylePalette.attachStyleSheet()

Availability

Dreamweaver 4.

Description

Displays a dialog box that lets users attach a style sheet to the current active document or to one of its attached style sheets, depending on the selection in the CSS Styles panel.

Arguments

None.

Returns

Nothing.

dreamweaver.cssStylePalette.deleteSelectedStyle()

Availability

Dreamweaver 3.

Description

Deletes the style that is currently selected in the CSS Styles panel from the document.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.cssStylePalette.canDeleteSelectedStyle()" on page 450.

dreamweaver.cssStylePalette.duplicateSelectedStyle()

Availability

Dreamweaver 3.

Description

Duplicates the style that is currently selected in the CSS Styles panel and displays the Duplicate Style dialog box to let the user assign a name or selector to the new style.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.cssStylePalette.canDuplicateSelectedStyle()" on page 450.

dreamweaver.cssStylePalette.editSelectedStyle()

Availability

Dreamweaver 3.

Description

Opens the Style Definition dialog box for the style that is currently selected in the CSS Styles panel.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.cssStyle.canEditSelectedStyle()" on page 450.

dreamweaver.cssStylePalette.editStyleSheet()

Availability

Dreamweaver 3.

Description

Opens the Edit Style Sheet dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.cssStylePalette.canEditStyleSheet()" on page 451.

dreamweaver.cssStylePalette.getMediaType()

Availability

Dreamweaver MX 2004.

Description

Gets target media type for rendering. The default media type is `"screen"`.

Arguments

None.

Returns

A string value that specifies the target media type.

Example

```
var mediaType = dw.cssStylePalette.getMediaType();
```

dreamweaver.cssStylePalette.getSelectedStyle()

Availability

Dreamweaver 3; `fullSelector` available in Dreamweaver MX.

Description

Gets the name of the style that is currently selected in the CSS Styles panel.

Arguments

`fullSelector`

- The *`fullSelector`* argument is a Boolean value that indicates whether the full selector or only the class should return. If nothing is specified, only the class name returns. For instance, `p.class1` is a selector that means the style is applied to any `p` tag of `class1`, but it does not apply, for instance, to a `div` tag of `class1`. Without the *`fullSelector`* argument, the `dreamweaver.cssStylePalette.getSelectedStyle()` function returns only the class name, `class1`, for the selector. The *`fullSelector`* argument tells the function to return `p.class1` instead of `class1`.

Returns

When the *`fullSelector`* argument is a `true` value, the function returns either the full selector or an empty string when the stylesheet node is selected.

When the *`fullSelector`* argument is a `false` value or it is omitted, a string that represents the class name of the selected style returns. If the selected style does not have a class or a stylesheet node is selected, an empty string returns.

Example

If the style `red` is selected, a call to the `dw.cssStylePalette.getSelectedStyle()` function returns `"red"`.

dreamweaver.cssStylePalette.getSelectedTarget() (deprecated*)*

Availability

Dreamweaver 3; deprecated in Dreamweaver MX because there is no longer an Apply To Menu in the CSS Styles panel.

Description

This function gets the selected element in the Apply To pop-up menu at the top of the CSS Styles panel.

Arguments

None.

Returns

A deprecated function; always returns a `null` value.

dreamweaver.cssStylePalette.getStyles()

Availability

Dreamweaver 3.

Description

Gets a list of all the class styles in the active document.

Arguments

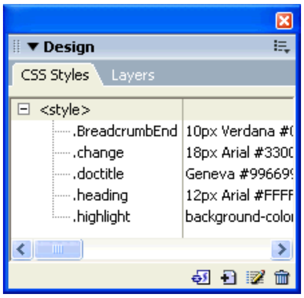
None.

Returns

An array of strings that represent the names of all the class styles in the document.

Example

If the CSS Styles panel is set up as shown in the following figure, a call to the `dreamweaver.cssStylePalette.getStyles()` function returns an array that contains these strings: `"BreadcrumbEnd"`, `"change"`, `"doctitle"`, `"heading"`, and `"highlight"`:



```
<style>
  .BreadcrumbEnd  10px Verdana #(
  .change         18px Arial #330(
  .doctitle       Geneva #99669(
  .heading        12px Arial #FFFF
  .highlight      background-colo
```

dreamweaver.cssStylePalette.newStyle()

Availability

Dreamweaver 3.

Description

Opens the New Style dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.cssStylePalette.setMediaType()

Availability

Dreamweaver MX 2004.

Description

Sets the target media type for rendering. Refreshes the rendering of all open documents.

Arguments

`mediaType`

- The *`mediaType`* argument specifies the new target media type.

Returns

Nothing.

Example

```
dw.cssStylePalette.setMediaType("print");
```

Frame and frameset functions

Frame and frameset functions handle two tasks: getting the names of the frames in a frameset and splitting a frame in two.

dom.getFrameNames()

Availability

Dreamweaver 3.

Description

Gets a list of all the named frames in the frameset.

Arguments

None.

Returns

An array of strings where each string is the name of a frame in the current frameset. Any unnamed frames are skipped. If none of the frames in the frameset is named, an empty array returns.

Example

For a document that contains four frames (two of which are named), a call to the `dom.getFrameNames()` function might return an array that contains the following strings:

- `"navframe"`
- `"main_content"`

dom.isDocumentInFrame()

Availability

Dreamweaver 4.

Description

Identifies whether the current document is being viewed inside a frameset.

Arguments

None.

Returns

A Boolean value: `true` if the document is in a frameset; `false` otherwise.

dom.saveAllFrames()

Availability

Dreamweaver 4.

Description

If a document is a frameset or is inside a frameset, this function saves all the frames and framesets from the Document window. If the specified document is not in a frameset, this function saves the document. This function opens the Save As dialog box for any documents that have not been previously saved.

Arguments

None.

Returns

Nothing.

dom.splitFrame()

Availability

Dreamweaver 3.

Description

Splits the selected frame vertically or horizontally.

Arguments

splitDirection

- The *splitDirection* argument is a string that must specify one of the following directions: `"up"`, `"down"`, `"left"`, or `"right"`.

Returns

Nothing.

Enabler

"dom.canSplitFrame()" on page 440.

Layer and image map functions

Layer and image map functions handle aligning, resizing, and moving layers and image map hotspots. The function description indicates if it applies to layers or to hotspots.

dom.align()

Availability

Dreamweaver 3.

Description

Aligns the selected layers or hotspots left, right, top, or bottom.

Arguments

alignDirection

- The *alignDirection* argument is a string that specifies the edge to align with the layers or hotspots (`"left"`, `"right"`, `"top"`, or `"bottom"`).

Returns

Nothing.

Enabler

"dom.canAlign()" on page 431.

dom.arrange()

Availability

Dreamweaver 3.

Description

Moves the selected hotspots in the specified direction.

Arguments

`toBackOrFront`

- The `toBackOrFront` argument is the direction in which the hotspots must move, either front or back.

Returns

Nothing.

Enabler

"dom.canArrange()" on page 432.

dom.makeSizesEqual()

Availability

Dreamweaver 3.

Description

Makes the selected layers or hotspots equal in height, width, or both. The last layer or hotspot selected is the guide.

Arguments

`bHoriz`, `bVert`

- The `bHoriz` argument is a Boolean value that indicates whether to resize the layers or hotspots horizontally.
- The `bVert` argument is a Boolean value that indicates whether to resize the layers or hotspots vertically.

Returns

Nothing.

dom.moveSelectionBy()

Availability

Dreamweaver 3.

Description

Moves the selected layers or hotspots by the specified number of pixels horizontally and vertically.

Arguments

x, y

- The *x* argument is the number of pixels that the selection must move horizontally.
- The *y* argument is the number of pixels that the selection must move vertically.

Returns

Nothing.

dom.resizeSelectionBy()

Availability

Dreamweaver 3.

Description

Resizes the currently selected layer or hotspot.

Arguments

`left`, *`top`*, *`bottom`*, *`right`*

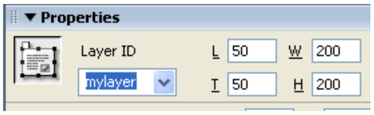
- The *`left`* argument is the new position of the left boundary of the layer or hotspot.
- The *`top`* argument is the new position of the top boundary of the layer or hotspot.
- The *`bottom`* argument is the new position of the bottom boundary of the layer or hotspot.
- The *`right`* argument is the new position of the right boundary of the layer or hotspot.

Returns

Nothing.

Example

If the selected layer has the Left, Top, Width, and Height properties shown, calling `dw.getDocumentDOM().resizeSelectionBy(-10,-30,30,10)` is equivalent to resetting Left to 40, Top to 20, Width to 240, and Height to 240.


Properties
Layer ID: mylayer
L 50 W 200
T 50 H 200

dom.setLayerTag()

Availability

Dreamweaver 3.

Description

Specifies the HTML tag that defines the selected layer or layers.

Arguments

`tagName`

- The `tagName` argument must be `"layer"`, `"ilayer"`, `"div"`, or `"span"`.

Returns

Nothing.

Layout environment functions

Layout environment functions handle operations that are related to the settings for working on a document. They affect the source, position, and opacity of the tracing image; get and set the ruler origin and units; turn the grid on and off and change its settings; and start or stop playing plug-ins.

dom.getRulerOrigin()

Availability

Dreamweaver 3.

Description

Gets the origin of the ruler.

Arguments

None.

Returns

An array of two integers. The first array item is the *x* coordinate of the origin, and the second array item is the *y* coordinate of the origin. Both values are in pixels.

dom.getRulerUnits()

Availability

Dreamweaver 3.

Description

Gets the current ruler units.

Arguments

None.

Returns

A string that contains one of the following values:

- `"in"`
- `"cm"`
- `"px"`

dom.getTracingImageOpacity()

Availability

Dreamweaver 3.

Description

Gets the opacity setting for the document's tracing image.

Arguments

None.

Returns

A value between 0 and 100, or nothing if no opacity is set.

Enabler

"dom.hasTracingImage()" on page 441.

dom.loadTracingImage()

Availability

Dreamweaver 3.

Description

Opens the Select Image Source dialog box. If the user selects an image and clicks OK, the Page Properties dialog box opens with the Tracing Image field filled in.

Arguments

None.

Returns

Nothing.

dom.playAllPlugins()

Availability

Dreamweaver 3.

Description

Plays all plug-in content in the document.

Arguments

None.

Returns

Nothing.

dom.playPlugin()

Availability

Dreamweaver 3.

Description

Plays the selected plug-in item.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canPlayPlugin()" on page 438.

dom.setRulerOrigin()

Availability

Dreamweaver 3.

Description

Sets the origin of the ruler.

Arguments

`xCoordinate`, *`yCoordinate`*

- The *`xCoordinate`* argument is a value, expressed in pixels, on the horizontal axis.
- The *`yCoordinate`* argument is a value, expressed in pixels, on the vertical axis.

Returns

Nothing.

dom.setRulerUnits()

Availability

Dreamweaver 3.

Description

Sets the current ruler units.

Arguments

`units`

- The *`units`* argument must be `"px"`, `"in"`, or `"cm"`.

Returns

Nothing.

dom.setTracingImagePosition()

Availability

Dreamweaver 3.

Description

Moves the upper left corner of the tracing image to the specified coordinates. If the arguments are omitted, the Adjust Tracing Image Position dialog box appears.

Arguments

`x`, *`y`*

- The *`x`* argument is the number of pixels that specify the horizontal coordinate.
- The *`y`* argument is the number of pixels that specify the vertical coordinate.

Returns

Nothing.

Enabler

"dom.hasTracingImage()" on page 441.

dom.setTracingImageOpacity()

Availability

Dreamweaver 3.

Description

Sets the opacity of the tracing image.

Arguments

`opacityPercentage`

- The *`opacityPercentage`* argument must be a number between 0 and 100.

Returns

Nothing.

Enabler

"dom.hasTracingImage()" on page 441.

Example

The following code sets the opacity of the tracing image to 30%:

```
dw.getDocumentDOM().setTracingOpacity('30');
```

dom.snapTracingImageToSelection()

Availability

Dreamweaver 3.

Description

Aligns the upper left corner of the tracing image with the upper left corner of the current selection.

Arguments

None.

Returns

Nothing.

Enabler

"dom.hasTracingImage()" on page 441.

dom.stopAllPlugins()

Availability

Dreamweaver 3.

Description

Stops all plug-in content that is currently playing in the document.

Arguments

None.

Returns

Nothing.

dom.stopPlugin()

Availability

Dreamweaver 3.

Description

Stops the selected plug-in item.

Arguments

None.

Returns

A Boolean value that indicates whether the selection is currently being played with a plug-in.

Enabler

"dom.canStopPlugin()" on page 441.

dreamweaver.arrangeFloatingPalettes()

Availability

Dreamweaver 3.

Description

Moves the visible floating panels to their default positions.

Arguments

None.

Returns

Nothing.

dreamweaver.showGridSettingsDialog()

Availability

Dreamweaver 3.

Description

Opens the Grid Settings dialog box.

Arguments

None.

Returns

Nothing.

Layout view functions

Layout view functions handle operations that change the layout elements within a document. They affect table, column, and cell settings, including position, properties, and appearance.

dom.addSpacerToColumn()

Availability

Dreamweaver 4.

Description

Creates a 1-pixel-high transparent spacer image at the bottom of a specified column in the currently selected table. This function fails if the current selection is not a table or if the operation is not successful.

Arguments

`colNum`

- The *`colNum`* argument is the column at the bottom of which the spacer image is created.

Returns

Nothing.

dom.createLayoutCell()

Availability

Dreamweaver 4.

Description

Creates a layout cell in the current document at the specified position and dimensions, either within an existing layout table or in the area below the existing content on the page. If the cell is created in an existing layout table, it must not overlap or contain any other layout cells or nested layout tables. If the rectangle is not inside an existing layout table, Dreamweaver tries to create a layout table to house the new cell. This function does not force the document into Layout view. This function fails if the cell cannot be created.

Arguments

`left`, *`top`*, *`width`*, *`height`*

- The *`left`* argument is the *x* position of the left border of the cell.
- The *`top`* argument is the *y* position of the top border of the cell.
- The *`width`* argument is the width of the cell in pixels.
- The *`height`* argument is the height of the cell in pixels.

Returns

Nothing.

dom.createLayoutTable()

Availability

Dreamweaver 4.

Description

Creates a layout table in the current document at the specified position and dimensions, either within an existing table or in the area below the existing content on the page. If the table is created in an existing layout table, it cannot overlap other layout cells or nested layout tables, but it can contain other layout cells or nested layout tables. This function does not force the document into Layout view. This function fails if the table cannot be created.

Arguments

`left`, *`top`*, *`width`*, *`height`*

- The *`left`* argument is the *x* position of the left border of the table.
- The *`top`* argument is the *y* position of the top border of the table.
- The *`width`* argument is the width of the table in pixels.
- The *`height`* argument is the height of the table in pixels.

Returns

Nothing.

dom.doesColumnHaveSpacer()

Availability

Dreamweaver 4.

Description

Determines whether a column contains a spacer image that Dreamweaver generated. It fails if the current selection is not a table.

Arguments

`colNum`

- The *`colNum`* argument is the column to check for a spacer image.

Returns

Returns `true` if the specified column in the currently selected table contains a spacer image that Dreamweaver generated; `false` otherwise.

dom.doesGroupHaveSpacers()

Availability

Dreamweaver 4.

Description

Determines whether the currently selected table contains a row of spacer images that Dreamweaver generated. It fails if the current selection is not a table.

Arguments

None.

Returns

Returns `true` if the table contains a row of spacer images; `false` otherwise.

dom.getClickedHeaderColumn()

Availability

Dreamweaver 4.

Description

If the user clicks a menu button in the header of a table in Layout view and causes the table header menu to appear, this function returns the index of the column that the user clicked. The result is undefined if the table header menu is not visible.

Arguments

None.

Returns

An integer that represents the index of the column.

dom.getShowLayoutTableTabs()

Availability

Dreamweaver 4.

Description

Determines whether the current document shows tabs for layout tables in Layout view.

Arguments

None.

Returns

Returns `true` if the current document displays tabs for layout tables in Layout view; `false` otherwise.

dom.getShowLayoutView()

Availability

Dreamweaver 4.

Description

Determines the view for the current document; either Layout or Standard view.

Arguments

None.

Returns

Returns `true` if the current document is in Layout view; `false` if the document is in Standard view.

dom.isColumnAutostretch()

Availability

Dreamweaver 4.

Description

Determines whether a column is set to expand and contract automatically, depending on the document size. This function fails if the current selection is not a table.

Arguments

`colNum`

- The *`colNum`* argument is the column to be automatically sized or fixed width.

Returns

Returns `true` if the column at the given index in the currently selected table is set to autostretch; `false` otherwise.

dom.makeCellWidthsConsistent()

Availability

Dreamweaver 4.

Description

In the currently selected table, this function sets the width of each column in the HTML to match the currently rendered width of the column. This function fails if the current selection is not a table or if the operation is not successful.

Arguments

None.

Returns

Nothing.

dom.removeAllSpacers()

Availability

Dreamweaver 4.

Description

Removes all spacer images generated by Dreamweaver from the currently selected table. This function fails if the current selection is not a table or if the operation is not successful.

Arguments

None.

Returns

Nothing.

dom.removeSpacerFromColumn()

Availability

Dreamweaver 4.

Description

Removes the spacer image from a specified column and deletes the spacer row if there are no more spacer images that Dreamweaver generated. This function fails if the current selection is not a table or if the operation is not successful.

Arguments

`colNum`

- The *`colNum`* argument is the column from which to remove the spacer image.

Returns

Nothing.

dom.setColumnAutostretch()

Availability

Dreamweaver 4.

Description

Switches a column between automatically sized or fixed width. If *`bAutostretch`* is `true`, the column at the given index in the currently selected table is set to autostretch; otherwise it's set to a fixed width at its current rendered width. This function fails if the current selection isn't a table or if the operation isn't successful.

Arguments

`colNum`, *`bAutostretch`*

- The *`colNum`* argument is the column to be automatically sized or set to a fixed width.
- The *`bAutostretch`* argument specifies whether to set the column to autostretch (`true`) or to a fixed width (`false`).

Returns

Nothing.

dom.setShowLayoutTableTabs()

Availability

Dreamweaver 4.

Description

Sets the current document to display tabs for layout tables whenever it's in Layout view. This function does not force the document into Layout view.

Arguments

`bShow`

- The *`bShow`* argument indicates whether to display tabs for layout tables when the current document is in Layout view. If *`bShow`* is `true`, Dreamweaver displays tabs; if *`bShow`* is `false`, Dreamweaver does not display tabs.

Returns

Nothing.

dom.setShowLayoutView()

Availability

Dreamweaver 4.

Description

Places the current document in Layout view if *`bShow`* is `true`.

Arguments

`bShow`

- The *`bShow`* argument is a Boolean value that toggles the current document between Layout view and Standard view. If *`bShow`* is `true`, the current document is in Layout view; if *`bShow`* is `false`, the current document is in Standard view.

Returns

Nothing.

Table editing functions

Table functions add and remove table rows and columns, change column widths and row heights, convert measurements from pixels to percents and back, and perform other standard table-editing tasks.

dom.convertWidthsToPercent()

Availability

Dreamweaver 3.

Description

This function converts all `WIDTH` attributes in the current table from pixels to percentages.

Arguments

None.

Returns

Nothing.

dom.convertWidthsToPixels()

Availability

Dreamweaver 4.

Description

This function converts all `WIDTH` attributes in the current table from percentages to pixels.

Arguments

None.

Returns

Nothing.

dom.decreaseColspan()

Availability

Dreamweaver 3.

Description

This function decreases the column span by one.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canDecreaseColspan()" on page 434.

dom.decreaseRowspan()

Availability

Dreamweaver 3.

Description

This function decreases the row span by one.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canDecreaseRowspan()" on page 434.

dom.deleteTableColumn()

Availability

Dreamweaver 3.

Description

This function removes the selected table column or columns.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canDeleteTableColumn()" on page 435.

dom.deleteTableRow()

Availability

Dreamweaver 3.

Description

This function removes the selected table row or rows.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canDeleteTableRow()" on page 435 .

dom.doDeferredTableUpdate()

Availability

Dreamweaver 3.

Description

If the Faster Table Editing option is selected in the General preferences, this function forces the table layout to reflect recent changes without moving the selection outside the table. This function has no effect if the Faster Table Editing option is not selected.

Arguments

None.

Returns

Nothing.

dom.getShowTableWidths()

Availability

Dreamweaver MX 2004.

Description

Returns whether table widths appear in standard or expanded tables mode (non-Layout mode). For information on whether Dreamweaver displays table tabs in Layout mode, see "dom.getShowLayoutTableTabs()" on page 365.

Arguments

None.

Returns

A Boolean value: `true` if Dreamweaver shows table widths in standard or expanded tables mode; `false` otherwise.

dom.getTableExtent()

Availability

Dreamweaver 3.

Description

This function gets the number of columns and rows in the selected table.

Arguments

None.

Returns

An array that contains two whole numbers. The first array item is the number of columns, and the second array item is the number of rows. If no table is selected, nothing returns.

dom.increaseColspan()

Availability

Dreamweaver 3.

Description

This function increases the column span by one.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canIncreaseColspan()" on page 436.

dom.increaseRowspan()

Availability

Dreamweaver 3.

Description

This function increases the row span by one.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canDecreaseRowspan()" on page 434.

dom.insertTableColumns()

Availability

Dreamweaver 3.

Description

This function inserts the specified number of table columns into the current table.

Arguments

numberOfCols, *bBeforeSelection*

- The *numberOfCols* argument is the number of columns to insert.
- The *bBeforeSelection* argument is a Boolean value: `true` indicates that the columns should be inserted before the column that contains the selection; `false` otherwise.

Returns

Nothing.

Enabler

"dom.canInsertTableColumns()" on page 436.

dom.insertTableRows()

Availability

Dreamweaver 3.

Description

This function inserts the specified number of table rows into the current table.

Arguments

numberOfRows, *bBeforeSelection*

- The *numberOfRows* argument is the number of rows to insert.
- The *bBeforeSelection* argument is a Boolean value: `true` indicates that the rows should be inserted above the row that contains the selection; `false` otherwise.

Returns

Nothing.

Enabler

"dom.canInsertTableRows()" on page 437.

dom.mergeTableCells()

Availability

Dreamweaver 3.

Description

This function merges the selected table cells.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canMergeTableCells()" on page 438.

dom.removeAllTableHeights()

Availability

Dreamweaver 3.

Description

This function removes all `HEIGHT` attributes from the selected table.

Arguments

None.

Returns

Nothing.

dom.removeAllTableWidths()

Availability

Dreamweaver 3.

Description

This function removes all `WIDTH` attributes from the selected table.

Arguments

None.

Returns

Nothing.

dom.removeColumnWidth()

Availability

Dreamweaver MX 2004.

Description

This function removes all `WIDTH` attributes from a single, selected column.

Arguments

None.

Returns

Nothing.

dom.selectTable()

Availability

Dreamweaver 3.

Description

Selects an entire table.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canSelectTable()" on page 439.

dom.setShowTableWidths()

Availability

Dreamweaver MX 2004.

Description

Toggles the display of table widths on and off in standard or Expanded Tables mode (non-Layout mode). This function sets the value for the current document and *any future* document unless otherwise specified. For information on setting the display of table tabs in Layout mode, see "dom.setShowLayoutTableTabs()" on page 368.

Arguments

`bShow`

- The *`bShow`* is a Boolean argument that indicates whether to display table widths for tables when the current document is in standard or Expanded Tables mode (non-Layout mode). If *`bShow`* is `true`, Dreamweaver displays the widths. If *`bShow`* is `false`, Dreamweaver does not display the widths.

Returns

Nothing.

dom.setTableCellTag()

Availability

Dreamweaver 3.

Description

This function specifies the tag for the selected cell.

Arguments

`tdOrTh`

- The *`tdOrTh`* argument must be either `"td"` or `"th"`.

Returns

Nothing.

dom.setTableColumns()

Availability

Dreamweaver 3.

Description

This function sets the number of columns in the selected table.

Arguments

`numberOfCols`

- The *`numberOfCols`* argument specifies the number of columns to set in the table.

Returns

Nothing.

dom.setTableRows()

Availability

Dreamweaver 3.

Description

This function sets the number of rows in the selected table.

Arguments

`numberOfCols`

- The *`numberOfRows`* argument specifies the number of rows to set in the selected table.

Returns

Nothing.

dom.showInsertTableRowsOrColumnsDialog()

Availability

Dreamweaver 3.

Description

This function opens the Insert Rows or Columns dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canInsertTableColumns()" on page 436 or "dom.canInsertTableRows()" on page 437.

dom.splitTableCell()

Availability

Dreamweaver 3.

Description

This function splits the current table cell into the specified number of rows or columns. If one or both of the arguments is omitted, the Split Cells dialog box appears.

Arguments

`{colsOrRows}`, `{numberToSplitInto}`

- The `colsOrRows` argument, which is optional, must be either `"columns"` or `"rows"`.
- The `numberToSplitInto` argument, which is optional, is the number of rows or columns into which the cell will be split.

Returns

Nothing.

Enabler

"dom.canSplitTableCell()" on page 440.

CHAPTER 18
Code

The Code functions let you perform operations on a document that appears in Code view. These operations include adding new menu or function tags to a Code Hints menu, finding and replacing string patterns, deleting the current selection from a document, printing all or selected code, editing tags, or applying syntax formatting to selected code.

Code functions

Code Hints are menus that Macromedia Dreamweaver MX 2004 opens when you type certain character patterns in Code view. Code Hints provide a typing shortcut by offering a list of strings that potentially complete the string you are typing. If the string you are typing appears in the menu, you can scroll to it and press Enter or Return to complete your entry. For example, when you type <, a pop-up menu shows a list of tag names. Instead of typing the rest of the tag name, you can select the tag from the menu to include it in your text.

You can add Code Hints menus to Dreamweaver by defining them in the CodeHints.xml file. For information on the CodeHints.xml file, see *Extending Dreamweaver.*

You can also add new Code Hints menus dynamically through JavaScript after Dreamweaver loads the contents of the CodeHints.xml file. For example, JavaScript code populates the list of session variables in the Bindings panel. You can use the same code to add a Code Hints menu, so when a user types **Session** in Code view, Dreamweaver displays a menu of session variables.

The CodeHints.xml file and the JavaScript API expose a useful subset of the Code Hints engine, but some Dreamweaver functionality is not accessible. For example, there is no JavaScript hook to open a color picker, so Dreamweaver cannot express the Attribute Values menu using JavaScript. You can only open a menu of text items from which you can insert text.

Code Coloring lets you specify code color styles and to modify existing code coloring schemes or create new ones. You can specify code coloring styles and schemes by modifying the Colors.xml and code coloring scheme files. For more information on these files, see *Extending Dreamweaver.*

The JavaScript API for Code Hints and Code Coloring consists of the following functions.

dreamweaver.codeHints.addMenu()

Availability

Dreamweaver MX.

Description

Dynamically defines a new `menu` tag in the CodeHints.xml file. If there is an existing menu tag that has the same pattern and document type, this function adds items to the existing menu.

Arguments

`menuGroupId, pattern, labelArray, {valueArray}, {iconArray}, {doctypes}, {casesensitive}`

- The *`menuGroupId`* argument is the ID attribute for one of the `menugroup` tags.
- The *`pattern`* argument is the pattern attribute for the new `menu` tag.
- The *`labelArray`* argument is an array of strings. Each string is the text for a single menu item in the pop-up menu.
- The *`valueArray`* argument, which is optional, is an array of strings, which should be the same length as the *`labelArray`* argument. When a user selects an item from the pop-up menu, the string in this array is inserted in the user's document. If the string to be inserted is always the same as the menu label, this argument might have a `null` value.
- The *`iconArray`* argument, which is optional, is either a string or an array of strings. If it is a string, it specifies the URL for a single image file that Dreamweaver uses for all items in the menu. If it is an array of strings, it must be the same length as the *`labelArray`* argument. Each string is a URL, relative to the Dreamweaver Configuration folder, for an image file that Dreamweaver uses as an icon for the corresponding menu item. If this argument is a `null` value, Dreamweaver displays the menu without icons.
- The *`doctypes`* argument, which is optional, specifies that this menu is active for only certain document types. You can specify the `doctypes` argument as a comma-separated list of document type IDs. For a list of Dreamweaver document types, see the Dreamweaver Configuration/Documenttypes/MMDocumentTypes.xml file.
- The *`casesensitive`* argument, which is optional, specifies whether the pattern is case-sensitive. The possible values for the *`casesensitive`* argument are the Boolean values `true` or `false`. The value defaults to `false` if you omit this argument. If the *`casesensitive`* argument is a `true` value, the Code Hints menu appears only if the text that the user types exactly matches the pattern that the pattern attribute specifies. If the *`casesensitive`* argument is a `false` value, the menu appears even if the pattern is lowercase and the text is uppercase.

Returns

Nothing.

Example

If the user creates a record set called "`myRs`", the following code would create a menu for `myRS`:

```
dw.codeHints.addMenu(
  "CodeHints_object_methods",  // menu is enabled if object methods are enabled
  "myRS.",                        // pop up menu if user types "myRS."
  new Array("firstName", "lastName"),  // items in drop-down menu for myRS
  new Array("firstName", "lastName"),  // text to actually insert in document
  null,             // no icons for this menu
  "ASP_VB, ASP_JS"); // specific to the ASP doc types
```

dreamweaver.codeHints.addFunction()

Availability

Dreamweaver MX.

Description

Dynamically defines a new `function` tag. If there is an existing `function` tag with the same pattern and document type, this function replaces the existing `function` tag.

Arguments

`menuGroupId, pattern, {doctypes}, {casesensitive}`

- The *`menuGroupId`* argument is the ID string attribute of a `menugroup` tag.
- The *`pattern`* argument is a string that specifies the pattern attribute for the new `function` tag.
- The *`doctypes`* argument, which is optional, specifies that this function is active for only certain document types. You can specify the `doctypes` argument as a comma-separated list of document type IDs. For a list of Dreamweaver document types, see the Dreamweaver Configuration/Documenttypes/MMDocumentTypes.xml file.
- The *`casesensitive`* argument, which is optional, specifies whether the pattern is case-sensitive. The possible values for the `casesensitive` argument are the Boolean values `true` or `false`. The value defaults to `false` if you omit this argument. If the *`casesensitive`* argument is a `true` value, the Code Hints menu appears only if the text that the user types exactly matches the pattern that the pattern attribute specifies. If *`casesensitive`* is a `false` value, the menu appears even if the pattern is lowercase and the text is uppercase.

Returns

Nothing.

Example

The following example of the `dw.codeHints.addFunction()` function adds the function name pattern `out.newLine()` to the Code Hints menu group `CodeHints_Object_Methods` and makes it active only for `JSP` document types:

```
dw.codeHints.addFunction(
  "CodeHints_Object_Methods",
  "out.newLine()",
  "JSP")
```

dreamweaver.codeHints.resetMenu()

Availability

Dreamweaver MX.

Description

Resets the specified menu tag or function tag to its state immediately after Dreamweaver reads the CodeHints.xml file. In other words, a call to this function erases the effect of previous calls to the `addMenu()` and `addFunction()` functions.

Arguments

`menuGroupId, pattern, {doctypes}`

- The *`menuGroupId`* argument is the ID string attribute of a `menugroup` tag.
- The *`pattern`* argument is a string that specifies the pattern attribute for the new `menu` or `function` tag to be reset.
- The *`doctypes`* argument, which is optional, specifies that this menu is active for only certain document types. You can specify the `doctypes` argument as a comma-separated list of document type IDs. For a list of Dreamweaver document types, see the Dreamweaver Configuration/Documenttypes/MMDocumentTypes.xml file.

Returns

Nothing.

Example

Your JavaScript code might build a Code Hints menu that contains user-defined session variables. Each time the list of session variables changes, that code needs to update the menu. Before the code can load the new list of session variables into the menu, it needs to remove the old list. Calling this function removes the old session variables.

dreamweaver.codeHints.showCodeHints()

Availability

Dreamweaver MX.

Description

Dreamweaver calls this function when the user opens the Edit > Show Code Hints menu item. The function opens the Code Hints menu at the current selection location in Code view.

Arguments

None.

Returns

Nothing.

Example

The following example opens the Code Hints menu at the current insertion point in the document when it is in Code view.

```
dw.codeHints.showCodeHints()
```

dreamweaver.reloadCodeColoring()

Description

Reloads code coloring files from the Dreamweaver Configuration/Code Coloring folder.

Arguments

None.

Returns

Nothing.

Example

```
dreamweaver.reloadCodeColoring()
```

Find/replace functions

Find/replace functions handle find and replace operations. They cover basic functionality, such as finding the next instance of a search pattern, and complex replacement operations that require no user interaction.

dreamweaver.findNext()

Availability

Dreamweaver 3; modified in Dreamweaver MX 2004.

Description

Finds the next instance of the search string that was specified previously by dreamweaver.setUpFind(), by dreamweaver.setUpComplexFind(), or by the user in the Find dialog box, and selects the instance in the document.

Arguments

{bUseLastSetupSearch}

- The *bUseLastSetupSearch* argument, which is optional, is a Boolean value. If *bUseLastSetupSearch* is the value `true`, which is the default if no argument is given, the function does a find-next operation using the parameters specified by a previous call to either the `dreamweaver.setupComplexFind()` function or the `dreamweaver.setupComplexFindReplace()` function. If you set *bUseLastSetupSearch* to the value `false`, the function ignores the previously set up search and performs a search for the next instance of the text that is currently selected in the document.

Returns

Nothing.

Enabler

"dreamweaver.canFindNext()" on page 444.

dreamweaver.replace()

Availability

Dreamweaver 3.

Description

Verifies that the current selection matches the search criteria that was specified by dreamweaver.setUpFindReplace(), by dreamweaver.setUpComplexFindReplace(), or by the user in the Replace dialog box; the function then replaces the selection with the replacement text that is specified by the search request.

Arguments

None.

Returns

Nothing.

dreamweaver.replaceAll()

Availability

Dreamweaver 3.

Description

Replaces each section of the current document that matches the search criteria that was specified by dreamweaver.setUpFindReplace(), by dreamweaver.setUpComplexFindReplace(), or by the user in the Replace dialog box, with the specified replacement content.

Arguments

None.

Returns

Nothing.

dreamweaver.setUpComplexFind()

Availability

Dreamweaver 3.

Description

Prepares for an advanced text or tag search by loading the specified XML query.

Arguments

`xmlQueryString`

- The *`xmlQueryString`* argument is a string of XML code that begins with `dwquery` and ends with `/dwquery`. (To get a string of the proper format, set up the query in the Find dialog box, click the Save Query button, open the query file in a text editor, and copy everything from the opening of the `dwquery` tag to the closing of the `/dwquery` tag.)

Returns

Nothing.

Example

The first line of the following example sets up a tag search and specifies that the scope of the search should be the current document. The second line performs the search operation.

```
dreamweaver.setUpComplexFind('<dwquery><queryparams matchcase="false" ¬
ignorewhitespace="true" useregexp="false"/><find>¬
<qtag qname="a"><qattribute qname="href" qcompare="="
qvalue="#">¬
  </qattribute><qattribute qname="onMouseOut" qcompare="=" qvalue="" ¬
qnegate="true"></qattribute></qtag></find></dwquery>');
dw.findNext();
```

dreamweaver.setUpComplexFindReplace()

Availability

Dreamweaver 3.

Description

Prepares for an advanced text or tag search by loading the specified XML query.

Arguments

`xmlQueryString`

- The *`xmlQueryString`* argument is a string of XML code that begins with the `dwquery` tag and ends with the `/dwquery` tag. (To get a string of the proper format, set up the query in the Find dialog box, click the Save Query button, open the query file in a text editor, and copy everything from the beginning of the `dwquery` tag to the end of the `/dwquery` tag.)

Returns

Nothing.

Example

The first statement in the following example sets up a tag search and specifies that the scope of the search should be four files. The second statement performs the search and replace operation.

```
dreamweaver.setUpComplexFindReplace('<dwquery><queryparams ¬
matchcase="false" ignorewhitespace="true" useregexp="false"/>¬
<find><qtag qname="a"><qattribute qname="href" qcompare="=" ¬
  qvalue="#"></qattribute><qattribute qname="onMouseOut" ¬
  qcompare="=" qvalue="" qnegate="true"></qattribute></qtag>¬
  </find><replace action="setAttribute" param1="onMouseOut" ¬
  param2="this.style.color='#000000';this.style.¬
fontWeight='normal'"/></dwquery>');
dw.replaceAll();
```

dreamweaver.setUpFind()

Availability

Dreamweaver 3.

Description

Prepares for a text or HTML source search by defining the search parameters for a subsequent `dreamweaver.findNext()` operation.

Arguments

`searchObject`

The *`searchObject`* argument is an object for which the following properties can be defined:

- The *`searchString`* is the text for which to search.
- The *`searchSource`* property is a Boolean value that indicates whether to search the HTML source.
- The *`{matchCase}`* property, which is optional, is a Boolean value that indicates whether the search is case-sensitive. If this property is not explicitly set, it defaults to `false`.
- The *`{ignoreWhitespace}`* property, which is optional, is a Boolean value that indicates whether white space differences should be ignored. The *`ignoreWhitespace`* property defaults to `false` if the value of the *`useRegularExpressions`* property is `true`, and `true` if the *`useRegularExpressions`* property is `false`.
- The *`{useRegularExpressions}`* property is a Boolean value that indicates whether the `searchString` property uses regular expressions. If this property is not explicitly set, it defaults to a value of `false`.

Returns

Nothing.

Example

The following code demonstrates three ways to create a *`searchObject`* object:

```
var searchParams;
searchParams.searchString = 'bgcolor="#FFCCFF"';
searchParams.searchSource = true;
dreamweaver.setUpFind(searchParams);

var searchParams = {searchString: 'bgcolor="#FFCCFF"', searchSource: true};
dreamweaver.setUpFind(searchParams);

dreamweaver.setUpFind({searchString: 'bgcolor="#FFCCFF"', searchSource: ¬
true});
```

dreamweaver.setUpFindReplace()

Availability

Dreamweaver 3.

Description

Prepares for a text or HTML source search by defining the search parameters and the scope for a subsequent `dreamweaver.replace()` or `dreamweaver.replaceAll()` operation.

Arguments

`searchObject`

The *`searchObject`* argument is an object for which the following properties can be defined:

- The *`searchString`* property is the text for which to search.
- The *`replaceString`* property is the text with which to replace the selection.
- The *`searchSource`* property is a Boolean value that indicates whether to search the HTML source.
- The *`{matchCase}`* property, which is optional, is a Boolean value that indicates whether the search is case-sensitive. If this property is not explicitly set, it defaults to a `false` value.
- The *`{ignoreWhitespace}`* property, which is optional, is a Boolean value that indicates whether white space differences should be ignored. The *`ignoreWhitespace`* property defaults to `false` if the *`useRegularExpressions`* property has a value of `true`, and defaults to a value of `true` if the *`useRegularExpressions`* property has a value of `false`.
- The *`{useRegularExpressions}`* property is a Boolean value that indicates whether the `searchString` property uses regular expressions. If this property is not explicitly set, it defaults to a value of `false`.

Returns

Nothing.

Example

The following code demonstrates three ways to create a *`searchObject`* object:

```
var searchParams;
searchParams.searchString = 'bgcolor="#FFCCFF"';
searchParams.replaceString = 'bgcolor="#CCFFCC"';
searchParams.searchSource = true;
dreamweaver.setUpFindReplace(searchParams);

var searchParams = {searchString: 'bgcolor="#FFCCFF"', replaceString:
  'bgcolor="#CCFFCC"', searchSource: true};
dreamweaver.setUpFindReplace(searchParams);

dreamweaver.setUpFindReplace({searchString: 'bgcolor="#FFCCFF"', ¬
replaceString: 'bgcolor="#CCFFCC"', searchSource: true});
```

dreamweaver.showFindDialog()

Availability

Dreamweaver 3.

Description

Opens the Find dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canShowFindDialog()" on page 448.

dreamweaver.showFindReplaceDialog()

Availability

Dreamweaver 3.

Description

Opens the Replace dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canShowFindDialog()" on page 448.

General editing functions

You handle general editing functions in the Document window. These functions insert text, HTML, and objects; apply, change, and remove font and character markup; modify tags and attributes; and more.

dom.applyCharacterMarkup()

Availability

Dreamweaver 3.

Description

Applies the specified type of character markup to the selection. If the selection is an insertion point, it applies the specified character markup to any subsequently typed text.

Arguments

`tagName`

- The *`tagName`* argument is the tag name that is associated with the character markup. It must be one of the following strings: `"b"`, `"cite"`, `"code"`, `"dfn"`, `"em"`, `"i"`, `"kbd"`, `"samp"`, `"s"`, `"strong"`, `"tt"`, `"u"`, or `"var"`.

Returns

Nothing.

dom.applyFontMarkup()

Availability

Dreamweaver 3.

Description

Applies the `FONT` tag and the specified attribute and value to the current selection.

Arguments

`attribute`, *`value`*

- The *`attribute`* argument must be `"face"`, `"size"`, or `"color"`.
- The *`value`* argument is the value to be assigned to the attribute; for example, `"Arial, Helvetica, sans-serif"`, `"5"`, or `"#FF0000"`.

Returns

Nothing.

dom.deleteSelection()

Availability

Dreamweaver 3.

Description

Deletes the selection in the document.

Arguments

None.

Returns

Nothing.

dom.editAttribute()

Availability

Dreamweaver 3.

Description

Displays the appropriate interface for editing the specified Document attribute. In most cases, this interface is a dialog box. This function is valid only for the active document.

Arguments

`attribute`

- The *`attribute`* is a string that specifies the tag attribute that you want to edit.

Returns

Nothing.

dom.exitBlock()

Availability

Dreamweaver 3.

Description

Exits the current paragraph or heading block, leaving the insertion point outside all block elements.

Arguments

None.

Returns

Nothing.

dom.getCharSet()

Availability

Dreamweaver 4.

Description

Returns the `charset` attribute in the meta tag of the document.

Arguments

None.

Returns

The encoding identity of the document. For example, for a Latin1 document, the function returns `iso-8859-1`.

dom.getFontMarkup()

Availability

Dreamweaver 3.

Description

Gets the value of the specified attribute of the `FONT` tag for the current selection.

Arguments

`attribute`

- The *`attribute`* argument must be `"face"`, `"size"`, or `"color"`.

Returns

A string that contains the value of the specified attribute or an empty string if the attribute is not set.

dom.getLineFromOffset()

Availability

Dreamweaver MX.

Description

Finds the line number of a specific character offset in the text (the HTML or JavaScript code) of the file.

Arguments

`offset`

- The *`offset`* argument is an integer that represents the character location from the beginning of the file.

Returns

An integer that represents the line number in the document.

dom.getLinkHref()

Availability

Dreamweaver 3.

Description

Gets the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found and then calling the `getAttribute('HREF')` on the link.

Arguments

None.

Returns

A string that contains the name of the linked file, which is expressed as a file:// URL.

dom.getLinkTarget()

Availability

Dreamweaver 3.

Description

Gets the target of the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found and then calling the `getAttribute('TARGET')` function on the link.

Arguments

None.

Returns

A string that contains the value of the `TARGET` attribute for the link or an empty string if no target is specified.

dom.getListTag()

Availability

Dreamweaver 3.

Description

Gets the style of the selected list.

Arguments

None.

Returns

A string that contains the tag that is associated with the list (`"ul"`, `"ol"`, or `"dl"`) or an empty string if no tag is associated with the list. This value always returns in lowercase letters.

dom.getTextAlignment()

Availability

Dreamweaver 3.

Description

Gets the alignment of the block that contains the selection.

Arguments

None.

Returns

A string that contains the value of the `ALIGN` attribute for the tag that is associated with the block or an empty string if the `ALIGN` attribute is not set for the tag. This value always returns in lowercase letters.

dom.getTextFormat()

Availability

Dreamweaver 3.

Description

Gets the block format of the selected text.

Arguments

None.

Returns

A string that contains the block tag that is associated with the text (for example, `"p"`, `"h1"`, `"pre"`, and so on) or an empty string if no block tag is associated with the selection. This value always returns in lowercase letters.

dom.hasCharacterMarkup()

Availability

Dreamweaver 3.

Description

Checks whether the selection already has the specified character markup.

Arguments

`markupTagName`

- The *`markupTagName`* argument is the name of the tag that you're checking. It must be one of the following strings: `"b"`, `"cite"`, `"code"`, `"dfn"`, `"em"`, `"i"`, `"kbd"`, `"samp"`, `"s"`, `"strong"`, `"tt"`, `"u"`, or `"var"`.

Returns

A Boolean value that indicates whether the entire selection has the specified character markup. The function returns a value of `false` if only part of the selection has the specified markup.

dom.indent()

Availability

Dreamweaver 3.

Description

Indents the selection using `BLOCKQUOTE` tags. If the selection is a list item, this function indents the selection by converting the selected item into a nested list. This nested list is of the same type as the outer list and contains one item, the original selection.

Arguments

None.

Returns

Nothing.

dom.insertHTML()

Availability

Dreamweaver 3.

Description

Inserts HTML content into the document at the current insertion point.

Arguments

`contentToInsert`, *`{bReplaceCurrentSelection}`*

- The *`contentToInsert`* argument is the content you want to insert.
- The *`bReplaceCurrentSelection`* argument, which is optional, is a Boolean value that indicates whether the content should replace the current selection. If the *`bReplaceCurrentSelection`* argument is a value of `true`, the content replaces the current selection. If the value is `false`, the content is inserted after the current selection.

Returns

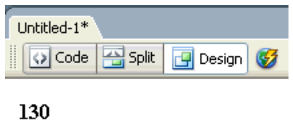
Nothing.

Example

The following code inserts the HTML string `<b>130</b>` into the current document:

```
var theDOM = dw.getDocumentDOM();
theDOM.insertHTML('<b>130</b>');
```

The result appears in the Document window, as shown in the following figure:



dom.insertObject()

Availability

Dreamweaver 3.

Description

Inserts the specified object, prompting the user for parameters if necessary.

Arguments

`objectName`

- The *`objectName`* argument is the name of an object in the Configuration/Objects folder.

Returns

Nothing.

Example

A call to the `dom.insertObject('Button')` function inserts a form button into the active document after the current selection. If nothing is selected, this function inserts the button at the current insertion point.

Note: Although object files can be stored in separate folders, it's important that these files have unique names. If a file called Button.htm exists in the Forms folder and also in the MyObjects folder, Dreamweaver cannot distinguish between them.

dom.insertText()

Availability

Dreamweaver 3.

Description

Inserts text content into the document at the current insertion point.

Arguments

`contentToInsert`, *`{bReplaceCurrentSelection}`*

- The *`contentToInsert`* argument is the content that you want to insert.
- The *`bReplaceCurrentSelection`* argument, which is optional, is a Boolean value that indicates whether the content should replace the current selection. If the *`bReplaceCurrentSelection`* argument is a value of `true`, the content replaces the current selection. If the value is `false`, the content is inserted after the current selection.

Returns

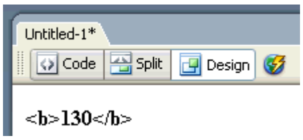
Nothing.

Example

The following code inserts the text: `&lt;b&gt;130&lt;/b&gt;` into the current document:

```
var theDOM = dreamweaver.getDocumentDOM();
theDOM.insertText('<b>130</b>');
```

The results appear in the Document window, as shown in the following figure:



<b>130</b>

dom.newBlock()

Availability

Dreamweaver 3.

Description

Creates a new block with the same tag and attributes as the block that contains the current selection or creates a new paragraph if the pointer is outside all blocks.

Arguments

None.

Returns

Nothing.

Example

If the current selection is inside a center-aligned paragraph, a call to the `dom.newBlock()` function inserts `<p align="center">` after the current paragraph.

dom.notifyFlashObjectChanged()

Availability

Dreamweaver 4.

Description

Tells Dreamweaver that the current Flash object file has changed. Dreamweaver updates the Preview display, resizing it as necessary, preserving the width-height ratio from the original size. For example, Flash Text uses this feature to update the text in the Layout view as the user changes its properties in the Command dialog box.

Arguments

None.

Returns

Nothing.

dom.outdent()

Availability

Dreamweaver 3.

Description

Outdents the selection.

Arguments

None.

Returns

Nothing.

dom.removeCharacterMarkup()

Availability

Dreamweaver 3.

Description

Removes the specified type of character markup from the selection.

Arguments

`tagName`

- The *`tagName`* argument is the tag name that is associated with the character markup. It must be one of the following strings: `"b"`, `"cite"`, `"code"`, `"dfn"`, `"em"`, `"i"`, `"kbd"`, `"samp"`, `"s"`, `"strong"`, `"tt"`, `"u"`, or `"var"`.

Returns

Nothing.

dom.removeFontMarkup()

Availability

Dreamweaver 3.

Description

Removes the specified attribute and its value from a `FONT` tag. If removing the attribute leaves only the `FONT` tag, the `FONT` tag is also removed.

Arguments

`attribute`

- The *`attribute`* argument must be `"face"`, `"size"`, or `"color"`.

Returns

Nothing.

dom.removeLink()

Availability

Dreamweaver 3.

Description

Removes the hypertext link from the selection.

Arguments

None.

Returns

Nothing.

dom.resizeSelection()

Availability

Dreamweaver 3.

Description

Resizes the selected object to the specified dimensions.

Arguments

`newWidth`, `newHeight`

- The `newWidth` argument specifies the new width to which the function will set the selected object.
- The `newHeight` argument specifies the new height to which the function will set the selected object.

Returns

Nothing.

dom.setAttributeWithErrorChecking()

Availability

Dreamweaver 3.

Description

Sets the specified attribute to the specified value for the current selection, prompting the user if the value is the wrong type or if it is out of range. This function is valid only for the active document.

Arguments

`attribute`, `value`

- The `attribute` argument specifies the attribute to set for the current selection.
- The `value` argument specifies the value to set for the attribute.

Returns

Nothing.

dom.setLinkHref()

Availability

Dreamweaver 3.

Description

Makes the selection a hypertext link or changes the URL value of the HREF tag that encloses the current selection.

Arguments

`linkHREF`

- The *`linkHREF`* argument is the URL (document-relative path, root-relative path, or absolute URL) comprising the link. If this argument is omitted, the Select HTML File dialog box appears.

Returns

Nothing.

Enabler

"dom.canSetLinkHref()" on page 439.

dom.setLinkTarget()

Availability

Dreamweaver 3.

Description

Sets the target of the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found and then calling the `setAttribute('TARGET')` function on the link.

Arguments

`{linkTarget}`

- The *`linkTarget`* argument, which is optional, is a string that represents a frame name, window name, or one of the reserved targets (`"_self"`, `"_parent"`, `"_top"`, or `"_blank"`). If the argument is omitted, the Set Target dialog box appears.

Returns

Nothing.

dom.setListBoxKind()

Availability

Dreamweaver 3.

Description

Changes the kind of the selected `SELECT` menu.

Arguments

`kind`

- The *`kind`* argument must be either `"menu"` or `"list box"`.

Returns

Nothing.

dom.showListPropertiesDialog()

Availability

Dreamweaver 3.

Description

Opens the List Properties dialog box.

Arguments

None.

Returns

Nothing.

Enabler

"dom.canShowListPropertiesDialog()" on page 440.

dom.setListTag()

Availability

Dreamweaver 3.

Description

Sets the style of the selected list.

Arguments

`listTag`

- The *`listTag`* argument is the tag that is associated with the list. It must be `"ol"`, `"ul"`, `"dl"`, or an empty string.

Returns

Nothing.

dom.setTextAlignment()

Availability

Dreamweaver 3.

Description

Sets the `ALIGN` attribute of the block that contains the selection to the specified value.

Arguments

`alignValue`

- The *`alignValue`* argument must be `"left"`, `"center"`, or `"right"`.

Returns

Nothing.

dom.setTextFieldKind()

Availability

Dreamweaver 3.

Description

Sets the format of the selected text field.

Arguments

`fieldType`

- The *`fieldType`* argument must be `"input"`, `"textarea"`, or `"password"`.

Returns

Nothing.

dom.setTextFormat()

Availability

Dreamweaver 4.

Description

Sets the block format of the selected text.

Arguments

`blockFormat`

- The *`blockFormat`* argument is a string that specifies one of the following formats: `""` (for no format), `"p"`, `"h1"`, `"h2"`, `"h3"`, `"h4"`, `"h5"`, `"h6"`, or `"pre"`.

Returns

Nothing.

dom.showFontColorDialog()

Availability

Dreamweaver 3.

Description

Opens the Color Picker dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.deleteSelection()

Availability

Dreamweaver 3.

Description

Deletes the selection in the active document or the Site panel; on the Macintosh, it deletes the text box that has focus in a dialog box or floating panel.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canDeleteSelection()" on page 443.

dreamweaver.editFontList()

Availability

Dreamweaver 3.

Description

Opens the Edit Font List dialog box.

Arguments

None.

Returns

Nothing.

dreamweaver.getFontList()

Availability

Dreamweaver 3.

Description

Gets a list of all the font groups that appear in the text Property inspector and in the Style Definition dialog box.

Arguments

None.

Returns

An array of strings that represent each item in the font list.

Example

For the default installation of Dreamweaver, a call to the `dreamweaver.getFontList()` function returns an array that contains the following items:

- `"Arial, Helvetica, sans-serif"`
- `"Times New Roman, Times, serif"`
- `"Courier New, Courier, mono"`
- `"Georgia, Times New Roman, Times, serif"`
- `"Verdana, Arial, Helvetica, sans-serif"`

dreamweaver.getFontStyles()

Availability

Dreamweaver 4.

Description

Returns the styles that a specified TrueType font supports.

Arguments

`fontName`

- The *`fontName`* argument is a string that contains the name of the font.

Returns

An array of three Boolean values that indicates what the font supports. The first value indicates whether the font supports *Bold*, the second indicates whether the font supports *Italic*, and the third indicates whether the font supports both *Bold* and *Italic*.

dreamweaver.getKeyState()

Availability

Dreamweaver 3.

Description

Determines whether the specified modifier key is depressed.

Arguments

key

- The *key* argument must be one of the following values: `"Cmd"`, `"Ctrl"`, `"Alt"`, or `"Shift"`. In Windows, `"Cmd"` and `"Ctrl"` refer to the Control key; on the Macintosh, `"Alt"` refers to the Option key.

Returns

A Boolean value that indicates whether the key is pressed.

Example

The following code checks that both the Shift and Control keys (Windows) or Shift and Command keys (Macintosh) are pressed before performing an operation:

```
if (dw.getKeyState("Shift") && dw.getKeyState("Cmd")){
  // execute code
}
```

dreamweaver.getNaturalSize()

Availability

Dreamweaver 4.

Description

Returns the width and height of a graphical object.

Arguments

url

- The *url* argument points to a graphical object for which the dimensions are wanted. Dreamweaver must support this object (GIF, JPEG, PNG, Flash, and Shockwave). The URL that is provided as the argument to the `getNaturalSize()` function must be an absolute URL that points to a local file; it cannot be a relative URL.

Returns

An array of two integers where the first integer defines the width of the object, and the second defines the height.

dreamweaver.getSystemFontList()

Availability

Dreamweaver 4.

Description

Returns a list of fonts for the system. This function can get either all fonts or only TrueType fonts. These fonts are needed for the Flash Text object.

Arguments

`fontTypes`

- The *`fontTypes`* argument is a string that contains either `"all"` or `"TrueType"`.

Returns

An array of strings that contain all the font names; returns a value of `null` if no fonts are found.

Print function

The print function lets the user print code from Code view.

dreamweaver.PrintCode()

Availability

Dreamweaver MX.

Description

In Windows, this function prints all or selected portions of code from the Code view. On the Macintosh, it prints all code or a page range of code.

Arguments

`showPrintDialog, document`

- The *`showPrintDialog`* argument is `true` or `false`. If this argument is set to `true`, in Windows, the `dreamweaver.PrintCode()` function displays the Print dialog box to ask if the user wants to print all text or selected text. On the Macintosh, the `dreamweaver.PrintCode()` function displays the Print dialog box to ask if the user wants to print all text or a page range.

  If the argument is set to `false`, `dreamweaver.PrintCode()` uses the user's previous selection. The default value is `true`.
- The *`document`* argument is the DOM of the document to print. For information on how to obtain the DOM for a document, see "dreamweaver.getDocumentDOM()" on page 259.

Returns

A Boolean value: `true` if the code can print; `false` otherwise.

Example

The following example calls `dw.PrintCode()` to invoke the Print dialog box for the user's document. If the function returns the value `false`, the code displays an alert to inform the user that it cannot execute the print request.

```
var theDOM = dreamweaver.getDocumentDOM("document");
if(!dreamweaver.PrintCode(true, theDOM))
{
  alert("Unable to execute your print request!");
}
```

Quick Tag Editor functions

Quick Tag Editor functions navigate through the tags within and surrounding the current selection. They remove any tag in the hierarchy, wrap the selection inside a new tag, and show the Quick Tag Editor to let the user edit specific attributes for the tag.

dom.selectChild()

Availability

Dreamweaver 3.

Description

Selects a child of the current selection. Calling this function is equivalent to selecting the next tag to the right in the tag selector at the bottom of the Document window.

Arguments

None.

Returns

Nothing.

dom.selectParent()

Availability

Dreamweaver 3.

Description

Selects the parent of the current selection. Calling this function is equivalent to selecting the next tag to the left in the tag selector at the bottom of the Document window.

Arguments

None.

Returns

Nothing.

dom.stripTag()

Availability

Dreamweaver 3.

Description

Removes the tag from around the current selection, leaving any contents. If the selection has no tags or contains more than one tag, Dreamweaver reports an error.

Arguments

None.

Returns

Nothing.

dom.wrapTag()

Availability

Dreamweaver 3.

Description

Wraps the specified tag around the current selection. If the selection is unbalanced, Dreamweaver reports an error.

Arguments

startTag

- The *startTag* argument is the source that is associated with the opening tag.

Returns

Nothing.

Example

The following code wraps a link around the current selection:

```
var theDOM = dw.getDocumentDOM();
var theSel = theDOM.getSelectedNode();
if (theSel.nodeType == Node.TEXT_NODE){
  theDOM.wrapTag('<a href="foo.html">');
}
```

dreamweaver.showQuickTagEditor()

Availability

Dreamweaver 3.

Description

Displays the Quick Tag Editor for the current selection.

Arguments

{nearWhat}, *{mode}*

- The optional *nearWhat* argument, if specified, must be either `"selection"` or `"tag selector"`. If this argument is omitted, the default value is `"selection"`.
- The optional *mode* argument, if specified, must be `"default"`, `"wrap"`, `"insert"`, or `"edit"`. If *mode* is `"default"` or omitted, Dreamweaver uses heuristics to determine the mode to use for the current selection. The *mode* argument is ignored if *nearWhat* is `"tag selector"`.

Returns

Nothing.

Code view functions

Code view functions include operations that are related to editing document source code (and that have subsequent impact on the Design view). The functions in this section let you add navigational controls to Code views within a split document view or the Code inspector window.

dom.formatRange()

Availability

Dreamweaver MX.

Description

Applies Dreamweaver automatic syntax formatting to a specified range of characters in the Code view, according to the settings in the Preferences > Code Format dialog box.

Arguments

startOffset, *endOffset*

- The *startOffset* argument is an integer that represents the beginning of the specified range as the offset from the beginning of the document.
- The *endOffset* argument is an integer that represents the end of the specified range as the offset from the beginning of the document.

Returns

Nothing.

dom.formatSelection()

Availability

Dreamweaver MX.

Description

Applies Dreamweaver automatic syntax formatting to the selected content (the same as selecting the Commands > Apply Source Formatting to Selection option) according to the settings in the Preferences > Code Format dialog box.

Arguments

None.

Returns

Nothing.

dom.getShowNoscript()

Availability

Dreamweaver MX.

Description

Gets the current state of the `noscript` content option (from the View > Noscript Content menu option). On by default, the `noscript` tag identifies page script content that can be rendered, or not (by choice), in the browser.

Arguments

None.

Returns

A Boolean value: `true` if the `noscript` tag content is currently rendered; `false` otherwise.

dom.getAutoValidationCount()

Availability

Dreamweaver MX 2004.

Description

Gets the number of errors, warnings, and information messages for the last auto-validation (also known as an inline validation) of the document. Currently only a target-browser check is performed during auto-validation (see "dom.runValidation()" on page 268).

Note: This function returns only the results that are currently in the results window for the document. If you want to make sure that the counts are up-to-date, you can call dom.runValidation() before calling this function.

Arguments

None.

Returns

An object with the following properties:

- The `numError` property, which is the number of errors
- The `numWarning` property, which is the number of warnings
- The `numInfo` property, which is the number of information messages

Example

```
theDom = dw.getDocumentDOM();
theDom.runValidation();
theDom.getAutoValidationCount();
```

dom.isDesignviewUpdated()

Availability

Dreamweaver 4.

Description

Determines whether the Design view and Text view content is synchronized for those Dreamweaver operations that require a valid document state.

Arguments

None.

Returns

A Boolean value: `true` if the Design view (WYSIWYG) is synchronized with the text in the Text view; `false` otherwise.

dom.isSelectionValid()

Availability

Dreamweaver 4.

Description

Determines whether a selection is valid, meaning it is currently synchronized with the Design view, or if it needs to be moved before an operation occurs.

Arguments

None.

Returns

A Boolean value: `true` if the current selection is in a valid piece of code; `false` if the document has not been synchronized, because the selection is not updated.

dom.setShowNoscript

Availability

Dreamweaver MX.

Description

Sets the `noscript` content option on or off (the same as selecting the View > Noscript Content option). On by default, the `noscript` tag identifies page script content that can be rendered, or not (by choice), in the browser.

Arguments

`{bShowNoscript}`

- The *`bShowNoscript`* argument, which is optional, is a Boolean value that indicates whether the `noscript` tag content should be rendered; `true` if the `noscript` tag content should be rendered, `false` otherwise.

Returns

Nothing.

dom.source.arrowDown()

Availability

Dreamweaver 4.

Description

Moves the insertion point down the Code view document, line by line. If content is already selected, this function extends the selection line by line.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of lines that the insertion point must move. If *nTimes* is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

dom.source.arrowLeft()

Availability

Dreamweaver 4.

Description

Moves the insertion point to the left in the current line of the Code view. If content is already selected, this function extends the selection to the left.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of characters that the insertion point must move. If *nTimes* is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

dom.source.arrowRight()

Availability

Dreamweaver 4.

Description

Moves the insertion point to the right in the current line of the Code view. If content is already selected, this function extends the selection to the right.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument, which is optional, is the number of characters that the insertion point must move. If *nTimes* is omitted, the default is 1.
- The *bShiftIsDown* argument, which is optional, is a Boolean value that indicates whether content is being selected. If *bShiftIsDown* is `true`, the content is selected; otherwise it is not.

Returns

Nothing.

dom.source.arrowUp()

Availability

Dreamweaver 4.

Description

Moves the insertion point up the Code view document, line by line. If content is already selected, this function extends the selection line by line.

Arguments

{nTimes}, *{bShiftIsDown}*

- The *nTimes* argument is the number of lines that the insertion point must move. If *nTimes* is omitted, the default is 1.
- The *bShiftIsDown* argument is a Boolean value that indicates whether content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

dom.source.balanceBracesTextview()

Availability

Dreamweaver 4.

Description

This function is a Code view extension that enables parentheses balancing. You can call `dom.source.balanceBracesTextview()` to extend a currently highlighted selection or insertion point from the opening of the surrounding parenthetical statement to the end of the statement to balance the following characters: `[]`, `{}` and `()`. Subsequent calls expand the selection through further levels of punctuation nesting.

Arguments

None.

Returns

Nothing.

dom.source.endOfDocument()

Availability

Dreamweaver 4.

Description

Places the insertion point at the end of the current Code view document. If content is already selected, this function extends the selection to the end of the document.

Arguments

`bShiftIsDown`

- The *`bShiftIsDown`* argument is a Boolean value that indicates whether content is being selected. If *`bShiftIsDown`* is `true`, the content is selected.

Returns

Nothing.

dom.source.endOfLine()

Availability

Dreamweaver 4.

Description

Places the insertion point at the end of the current line. If content is already selected, this function extends the selection to the end of the current line.

Arguments

`bShiftIsDown`

- The *`bShiftIsDown`* argument is a Boolean value that indicates whether content is being selected. If *`bShiftIsDown`* is `true`, the content is selected.

Returns

Nothing.

dom.source.endPage()

Availability

Dreamweaver 4.

Description

Moves the insertion point to the end of the current page or to the end of the next page if the insertion point is already at the end of a page. If content is already selected, this function extends the selection page by page.

Arguments

`{nTimes}`, *`{bShiftIsDown}`*

- The *`nTimes`* argument, which is optional, is the number of pages that the insertion point must move. If *`nTimes`* is omitted, the default is 1.
- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether content is being selected. If `bShiftIsDown` is `true`, the content is selected.

Returns

Nothing.

dom.source.getCurrentLines()

Availability

Dreamweaver 4.

Description

Returns the line numbers for the specified offset locations from the beginning of the document.

Arguments

None.

Returns

The line numbers for the current selection.

dom.source.getSelection()

Description

Gets the selection in the current document, which is expressed as character offsets into the document's Code view.

Arguments

None.

Returns

A pair of integers that represent offsets from the beginning of the source document. The first integer is the opening of the selection; the second is the closing of the selection. If the two numbers are equal, the selection is an insertion point. If there is no selection in the source, both numbers are -1.

dom.source.getLineFromOffset()

Availability

Dreamweaver MX.

Description

Takes an offset into the source document.

Arguments

None.

Returns

The associated line number, or -1 if the offset is negative or past the end of the file.

dom.source.getText()

Availability

Dreamweaver 4.

Description

Returns the text string in the source between the designated offsets.

Arguments

`startOffset`, *`endOffset`*

- The *`startOffset`* argument is an integer that represents the offset from the beginning of the document.
- The *`endOffset`* argument is an integer that represents the end of the document.

Returns

A string that represents the text in the source code between the offsets *`start`* and *`end`*.

dom.source.getValidationErrorsForOffset()

Availability

Dreamweaver MX 2004.

Description

Returns a list of validation errors at the specified offset, or it searches from the offset for the next error. If none are found the function, it returns `null`.

Arguments

`offset, {searchDirection}`

- The *`offset`* argument is a number that specifies the offset in the code for which the function will return any errors.
- The *`searchDirection`* argument, which is optional, is a string that specifies `"empty"`, `"forward"` or `"back"`. If specified, the function searches forward or back from the given offset to the next characters with errors and returns them. If not specified, the function simply checks for errors at the given offset.

Returns

An array of objects or the value `null`. Each object in the array has the following properties:

- The `message` object is a string that contains the error message.
- The `floaterName` object is a string that contains the name of the results window. You can pass this value to the `showResults()` or `setFloaterVisibility()` functions.
- The `floaterIndex` object is an index of items in the floater results list.
- The `start` object is the opening index of underlined code.
- The `end` object is the closing index of underlined code.

Note: The returned floater indexes should not be stored because they can change frequently, such as when documents are opened or closed.

Example

The following example calls `getValidationErrorsForOffset()` to check for any errors at the offset of the current selection. If the function returns an error, the code calls the `alert()` function to display the error message to the user.

```
var offset = dw.getDocumentDOM().source.getSelection()[0];
var errors = dw.getDocumentDOM().source.getValidationErrorsForOffset(offset);
if ( errors && errors.length > 0 )
  alert( errors[0].message );
```

dom.source.indentTextview()

Availability

Dreamweaver 4.

Description

Moves selected Code view text one tab stop to the right.

Arguments

None.

Returns

Nothing.

dom.source.insert()

Availability

Dreamweaver 4.

Description

Inserts the specified string into the source code at the specified offset from the beginning of the source file. If the offset is not greater than or equal to zero, the insertion fails and the function returns `false`.

Arguments

`offset`, *`string`*

- The *`offset`* argument is the offset from the beginning of the file where the string must be inserted.
- The *`string`* argument is the string to insert.

Returns

A Boolean value: `true` if successful; `false` otherwise.

dom.source.nextWord()

Availability

Dreamweaver 4.

Description

Moves the insertion point to the beginning of the next word (or words, if specified) in the Code view. If content is already selected, this function extends the selection to the right.

Arguments

`{nTimes}`, *`{bShiftIsDown}`*

- The *`nTimes`* argument, which is optional, is the number of words that the insertion point must move. If *`nTimes`* is omitted, the default is 1.
- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether content is being selected. If *`bShiftIsDown`* is `true`, the content is selected.

Returns

Nothing.

dom.source.outdentTextview()

Availability

Dreamweaver 4.

Description

Moves selected Code view text one tab stop to the left.

Arguments

None.

Returns

Nothing.

dom.source.pageDown()

Availability

Dreamweaver 4.

Description

Moves the insertion point down the Code view document, page by page. If content is already selected, this function extends the selection page by page.

Arguments

`{nTimes}`, `{bShiftIsDown}`

- The `nTimes` argument, which is optional, is the number of pages that the insertion point must move. If `nTimes` is omitted, the default is 1.
- The `bShiftIsDown` argument, which is optional, is a Boolean value that indicates whether content is being selected. If `bShiftIsDown` is `true`, the content is selected.

Returns

Nothing.

dom.source.pageUp()

Availability

Dreamweaver 4.

Description

Moves the insertion point up the Code view document, page by page. If content is already selected, this function extends the selection page by page.

Arguments

`{nTimes}`, `{bShiftIsDown}`

- The `nTimes` argument, which is optional, is the number of pages that the insertion point must move. If `nTimes` is omitted, the default is 1.
- The `bShiftIsDown` argument, which is optional, is a Boolean value that indicates whether content is being selected. If `bShiftIsDown` is `true`, the content is selected.

Returns

Nothing.

dom.source.previousWord()

Availability

Dreamweaver 4.

Description

Moves the insertion point to the beginning of the previous word (or words, if specified) in Code view. If content is already selected, this function extends the selection to the left.

Arguments

`{nTimes}`, `{bShiftIsDown}`

- The `nTimes` argument, which is optional, is the number of words that the insertion point must move. If `nTimes` is omitted, the default is 1.
- The `bShiftIsDown` argument, which is optional, is a Boolean value that indicates whether content is being selected. If `bShiftIsDown` is `true`, the content is selected.

Returns

Nothing.

dom.source.replaceRange()

Availability

Dreamweaver 4.

Description

Replaces the range of source text between `startOffset` and `endOffset` with `string`. If `startOffset` is greater than `endOffset` or if either offset is not a positive integer, it does nothing and returns `false`. If `endOffset` is greater than the number of characters in the file, it replaces the range between `startOffset` and the end of the file. If both `startOffset` and `endOffset` are greater than the number of characters in the file, it inserts the text at the end of the file.

Arguments

`startOffset`, `endOffset`, `string`

- The `startOffset` argument is the offset that indicates the beginning of the block to replace.
- The `endOffset` argument is the offset that indicates the end of the block to replace.
- The `string` argument is the string to insert.

Returns

A Boolean value: `true` if successful; `false` otherwise.

dom.source.scrollEndFile()

Availability

Dreamweaver 4.

Description

Scrolls the Code view to the bottom of the document file without moving the insertion point.

Arguments

None.

Returns

Nothing.

dom.source.scrollLineDown()

Availability

Dreamweaver 4.

Description

Scrolls the Code view down line by line without moving the insertion point.

Arguments

`nTimes`

- The *`nTimes`* argument is the number of lines to scroll. If *`nTimes`* is omitted, the default is 1.

Returns

Nothing.

dom.source.scrollLineUp()

Availability

Dreamweaver 4.

Description

Scrolls the Code view up line by line without moving the insertion point.

Arguments

`nTimes`

- The *`nTimes`* argument is the number of lines to scroll. If *`nTimes`* is omitted, the default is 1.

Returns

Nothing.

dom.source.scrollPageDown()

Availability

Dreamweaver 4.

Description

Scrolls the Code view down page by page without moving the insertion point.

Arguments

`nTimes`

- The `nTimes` argument is the number of pages to scroll. If `nTimes` is omitted, the default is 1.

Returns

Nothing.

dom.source.scrollPageUp()

Availability

Dreamweaver 4.

Description

Scrolls the Code view up page by page without moving the insertion point.

Arguments

`nTimes`

- The `nTimes` argument is the number of pages to scroll. If `nTimes` is omitted, the default is 1.

Returns

Nothing.

dom.source.scrollTopFile()

Availability

Dreamweaver 4.

Description

Scrolls the Code view to the top of the document file without moving the insertion point.

Arguments

None.

Returns

Nothing.

dom.source.selectParentTag()

Availability

Dreamweaver 4.

Description

This function is a Code view extension that enables tag balancing. You can call `dom.source.selectParentTag()` to extend a currently highlighted selection or insertion point from the surrounding open tag to the closing tag. Subsequent calls extend the selection to additional surrounding tags until there are no more enclosing tags.

Arguments

None.

Returns

Nothing.

dom.source.setCurrentLine()

Availability

Dreamweaver 4.

Description

Puts the insertion point at the beginning of the specified line. If the *`lineNumber`* argument is not a positive integer, the function does nothing and returns `false`. It puts the insertion point at the beginning of the last line if *`lineNumber`* is larger than the number of lines in the source.

Arguments

`lineNumber`

- The *`lineNumber`* argument is the line at the beginning of which the insertion point is placed.

Returns

A Boolean value: `true` if successful; `false` otherwise.

dom.source.startOfDocument()

Availability

Dreamweaver 4.

Description

Places the insertion point at the beginning of the Code view document. If content is already selected, this function extends the selection to the beginning of the document.

Arguments

`bShiftIsDown`

- The *`bShiftIsDown`* argument is a Boolean value that indicates whether content is being selected. If *`bShiftIsDown`* is `true`, the content is selected.

Returns

Nothing.

dom.source.startOfLine()

Availability

Dreamweaver 4.

Description

Places the insertion point at the beginning of the current line. If content is already selected, this function extends the selection to the beginning of the current line.

Arguments

`bShiftIsDown`

- The *`bShiftIsDown`* argument is a Boolean value that indicates whether content is being selected. If *`bShiftIsDown`* is `true`, the content is selected.

Returns

Nothing.

dom.source.topPage()

Availability

Dreamweaver 4.

Description

Moves the insertion point to the top of the current page or to the top of the previous page if the insertion point is already at the top of a page. If content is already selected, this function extends the selection page by page.

Arguments

`{nTimes}`, *`{bShiftIsDown}`*

- The *`nTimes`* argument, which is optional, is the number of pages that the insertion point must move. If *`nTimes`* is omitted, the default is 1.
- The *`bShiftIsDown`* argument, which is optional, is a Boolean value that indicates whether content is being selected. If *`bShiftIsDown`* is `true`, the content is selected.

Returns

Nothing.

dom.source.wrapSelection()

Availability

Dreamweaver 4.

Description

Inserts the text of `startTag` before the current selection and the text of `endTag` after the current selection. The function then selects the entire range between, and including, the inserted tags. If the current selection was an insertion point, then the function places the insertion point between the `startTag` and `endTag`. (`startTag` and `endTag` don't have to be tags; they can be any arbitrary text.)

Arguments

`startTag`, `endTag`

- The `startTag` argument is the text to insert at the beginning of the selection.
- The `endTag` argument is the text to insert at the end of the selection.

Returns

Nothing.

dom.synchronizeDocument()

Availability

Dreamweaver 4.

Description

Synchronizes the Design and Code views.

Arguments

None.

Returns

Nothing.

Tag editor and tag library functions

You can use tag editors to insert new tags, edit existing tags, and access reference information about tags. The Tag Chooser lets users organize their tags so that they can easily select frequently used tags. The tag libraries that come with Dreamweaver store information about tags that are used in standards-based markup languages and most widely used tag-based scripting languages. You can use the JavaScript tag editor, Tag Chooser, and tag library functions when you need to access and work with tag editors and tag libraries in your extensions.

dom.getTagSelectorTag()

Availability

Dreamweaver MX.

Description

This function gets the DOM node for the tag that is currently selected in the Tag Selector bar at the bottom of the document window.

Arguments

None.

Returns

The DOM node for the currently selected tag; `null` if no tag is selected.

dreamweaver.popupInsertTagDialog()

Availability

Dreamweaver MX.

Description

This function checks the VTM files to see if a tag editor has been defined for the tag. If so, the editor for that tag pops up and accepts the start tag. If not, the start tag is inserted unmodified into the user's document.

Arguments

A start tag string that includes one of the following types of initial values:

- A tag, such as `<input>`
- A tag with attributes, such as `<input type='text'>`
- A directive, such as `<%= %>`

Returns

A Boolean value: `true` if anything is inserted into the document; `false` otherwise.

dreamweaver.popupEditTagDialog()

Availability

Dreamweaver MX.

Description

If a tag is selected, this function opens the tag editor for that tag, so you can edit the tag.

Arguments

None.

Returns

Nothing.

Enabler

"dreamweaver.canPopupEditTagDialog()" on page 445.

dreamweaver.showTagChooser()

Availability

Dreamweaver MX.

Description

This function displays the Tag Chooser dialog box, brings it to the front, and sets it in focus.

Arguments

None.

Returns

Nothing.

dreamweaver.showTagLibraryEditor()

Availability

Dreamweaver MX.

Description

This function opens the Tag Library editor.

Arguments

None.

Returns

None.

dreamweaver.tagLibrary.getTagLibraryDOM()

Availability

Dreamweaver MX.

Description

Given the URL of a *`filename`*`.vtm` file, this function returns the DOM for that file, so that its contents can be edited. This function should be called only when the Tag Library editor is active.

Arguments

`fileURL`

- The *`fileURL`* argument is the URL of a *`filename`*`.vtm` file, relative to the Configuration/Tag Libraries folder, as shown in the following example:

```
"HTML/img.vtm"
```

Returns

A DOM pointer to a new or previously existing file within the TagLibraries folder.

dreamweaver.tagLibrary.getSelectedLibrary()

Availability

Dreamweaver MX.

Description

If a library node is selected in the Tag Library editor, this function gets the library name.

Arguments

None.

Returns

A string, the name of the library that is currently selected in the Tag Library editor; returns an empty string if no library is selected.

dreamweaver.tagLibrary.getSelectedTag()

Availability

Dreamweaver MX.

Description

If an attribute node is currently selected, this function gets the name of the tag that contains the attribute.

Arguments

None.

Returns

A string, name of the tag that is currently selected in the Tag Library editor; returns an empty string if no tag is selected.

dreamweaver.tagLibrary.importDTDOrSchema()

Availability

Dreamweaver MX.

Description

This function imports a DTD or schema file from a remote server into the tag library.

Arguments

`fileURL, Prefix`

- The *`fileURL`* argument is the path to DTD or schema file, in local URL format.
- The *`Prefix`* argument is the prefix string that should be added to all tags in this tag library.

Returns

Name of the imported tag library.

dreamweaver.tagLibrary.getImportedTagList()

Availability

Dreamweaver MX.

Description

This function generates a list of `tagInfo` objects from an imported tag library.

Arguments

`libname`

- The *`libname`* argument is the name of the imported tag library.

Returns

Array of `tagInfo` objects.

A `taginfo` object contains information about a single tag that is included in the tag library. The following properties are defined in a `tagInfo` object:

- The `tagName` property, which is a string
- The `attributes` property, which is an array of strings. Each string is the name of an attribute that is defined for this tag.

Example:

The following example shows that using the `dw.tagLibrary.getImportedTagList()` function can get an array of tags from the `libName` library:

```
// "fileURL" and "prefix" have been entered by the user.
// tell the Tag Library to Import the DTD/Schema
var libName = dw.tagLibrary.importDTDOrSchema(fileURL, prefix);

// get the array of tags for this library
// this is the TagInfo object
var tagArray = dw.tagLibrary.getImportedTagList(libName);

// now I have an array of tagInfo objects.
// I can get info out of them. This gets info out of the first one.
// note: this assumes there is at least one TagInfo in the array.
var firstTagName = tagArray[0].name;
var firstTagAttributes = tagArray[0].attributes;
// note that firstTagAttributes is an array of attributes.
```

CHAPTER 19
Enablers

Macromedia Dreamweaver MX 2004 Enabler functions determine whether another function can perform a specific operation in the current context. The function specifications describe the general circumstances under which each function returns a `true` value. However, the descriptions are not intended to be comprehensive and might exclude some cases in which the function would return a `false` value.

Enablers

The enabler functions in the JavaScript API include the following functions.

dom.canAlign()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Align Left, Align Right, Align Top, or Align Bottom operation.

Arguments

None.

Returns

A Boolean value that indicates whether two or more layers or hotspots are selected.

dom.canApplyTemplate()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Apply To Page operation. This function is valid only for the active document.

Arguments

None.

Returns

A Boolean value that indicates whether the document is not a library item or a template, and that the selection is not within the `NOFRAMES` tag.

dom.canArrange()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Bring to Front or Move to Back operation.

Arguments

None.

Returns

A Boolean value that indicates whether a hotspot is selected.

dom.canClipCopyText()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Copy as Text operation.

Arguments

None.

Returns

A Boolean value: `true` if the opening and closing offsets of the selection are different; `false` otherwise, to indicate that nothing has been selected.

dom.canClipPaste()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Paste operation.

Arguments

None.

Returns

A Boolean value: `true` if the Clipboard contains any content that can be pasted into Dreamweaver; `false` otherwise.

dom.canClipPasteText()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Paste as Text operation.

Arguments

None.

Returns

A Boolean value: `true` if the Clipboard contains any content that can be pasted into Dreamweaver as text; `false` otherwise.

dom.canConvertLayersToTable()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Convert Layers to Table operation.

Arguments

None.

Returns

A Boolean value: `true` if all the content in the `BODY` section of the document is contained within layers; `false` otherwise.

dom.canConvertTablesToLayers()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Convert Tables to Layers operation.

Arguments

None.

Returns

A Boolean value: `true` if all the content in the `BODY` section of the document is contained within tables, and the document is not based on a template; `false` otherwise.

dom.canDecreaseColspan()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Decrease Colspan operation.

Arguments

None.

Returns

A Boolean value: `true` if the current cell has a `COLSPAN` attribute and that attribute's value is greater than or equal to 2; `false` otherwise.

dom.canDecreaseRowspan()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Decrease Rowspan operation.

Arguments

None.

Returns

A Boolean value: `true` if the current cell has a `ROWSPAN` attribute and that attribute's value is greater than or equal to 2; `false` otherwise.

dom.canDeleteTableColumn()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Delete Column operation.

Arguments

None.

Returns

A Boolean value: `true` if the insertion point is inside a cell or if a cell or column is selected; `false` otherwise.

dom.canDeleteTableRow()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Delete Row operation.

Arguments

None.

Returns

A Boolean value: `true` if the insertion point is inside a cell or if a cell or row is selected; `false` otherwise.

dom.canEditNoFramesContent()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Edit No Frames Content operation.

Arguments

None.

Returns

A Boolean value: `true` if the current document is a frameset or within a frameset; `false` otherwise.

dom.canIncreaseColspan()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Increase Colspan operation.

Arguments

None.

Returns

A Boolean value: `true` if there are any cells to the right of the current cell; `false` otherwise.

dom.canIncreaseRowspan()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Increase Rowspan operation.

Arguments

None.

Returns

A Boolean value: `true` if there are any cells below the current cell; `false` otherwise.

dom.canInsertTableColumns()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Insert Column(s) operation.

Arguments

None.

Returns

A Boolean value: `true` if the selection is inside a table; `false` if the selection is an entire table or is not inside a table.

dom.canInsertTableRows()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Insert Row(s) operation.

Arguments

None.

Returns

A Boolean value: `true` if the selection is inside a table; `false` if the selection is an entire table or is not inside a table.

dom.canMakeNewEditableRegion()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a New Editable Region operation.

Arguments

None.

Returns

A Boolean value: `true` if the current document is a template (DWT) file.

dom.canMarkSelectionAsEditable()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Mark Selection as Editable operation.

Arguments

None.

Returns

A Boolean value: `true` if there is a selection and the current document is a DWT file; `false` otherwise.

dom.canMergeTableCells()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Merge Cells operation.

Arguments

None.

Returns

A Boolean value: `true` if the selection is an adjacent grouping of table cells; `false` otherwise.

dom.canPlayPlugin()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Play operation. This function is valid only for the active document.

Arguments

None.

Returns

A Boolean value: `true` if the selection can be played with a plug-in.

dom.canRedo()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Redo operation.

Arguments

None.

Returns

A Boolean value: `true` if any steps remain to redo; `false` otherwise.

dom.canRemoveEditableRegion()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Unmark Editable Region operation.

Arguments

None.

Returns

A Boolean value: `true` if the current document is a template; `false` otherwise.

dom.canSelectTable()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Select Table operation.

Arguments

None.

Returns

A Boolean value: `true` if the insertion point or selection is within a table; `false` otherwise.

dom.canSetLinkHref()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can change the link around the current selection or create one if necessary.

Arguments

None.

Returns

A Boolean value: `true` if the selection is an image, text, or if the insertion point is inside a link; `false` otherwise. A text selection is defined as a selection for which the text Property inspector would appear.

dom.canShowListPropertiesDialog()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can show the List Properties dialog box.

Arguments

None.

Returns

A Boolean value: `true` if the selection is within an `LI` tag; `false` otherwise.

dom.canSplitFrame()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Split Frame [Left | Right | Up | Down] operation.

Arguments

None.

Returns

A Boolean value: `true` if the selection is within a frame; `false` otherwise.

dom.canSplitTableCell()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Split Cell operation.

Arguments

None.

Returns

A Boolean value: `true` if the insertion point is inside a table cell or the selection is a table cell; `false` otherwise.

dom.canStopPlugin()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Stop operation.

Arguments

None.

Returns

A Boolean value: `true` if the selection is currently being played with a plug-in; `false` otherwise.

dom.canUndo()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Undo operation.

Arguments

None.

Returns

A Boolean value: `true` if any steps remain to undo; `false` otherwise.

dom.hasTracingImage()

Availability

Dreamweaver 3.

Description

Checks whether the document has a tracing image.

Arguments

None.

Returns

A Boolean value: `true` if the document has a tracing image; `false` otherwise.

dreamweaver.assetPalette.canEdit()

Availability

Dreamweaver 4.

Description

Enables menu items in the Assets panel for editing.

Arguments

None.

Returns

Returns a Boolean value: `true` if the asset can be edited; `false` otherwise. Returns a `false` value for colors and URLs in the Site list, and returns a `false` value for a multiple selection of colors and URLs in the Favorites list.

dreamweaver.assetPalette.canInsertOrApply()

Availability

Dreamweaver 4.

Description

Checks if the selected elements can be inserted or applied. Returns either a `true` or `false` value so the menu items can be enabled or disabled for insertion or application.

Arguments

None.

Returns

Returns a Boolean value: `true` if the selected elements can be inserted or applied; `false` if the current page is a template and the current category is Templates. The function also returns a `false` value if no document is open or if a library item is selected in the document and the current category is Library.

dreamweaver.canClipCopy()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Copy operation.

Arguments

None.

Returns

A Boolean value: `true` if there is any content selected that can be copied to the Clipboard; `false` otherwise.

dreamweaver.canClipCut()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Cut operation.

Arguments

None.

Returns

A Boolean value: `true` if there is any selected content that can be cut to the Clipboard; `false` otherwise.

dreamweaver.canClipPaste()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Paste operation.

Arguments

None.

Returns

A Boolean value: `true` if the Clipboard contains any content that can be pasted into the current document or the active window in the Site panel (on the Macintosh, a text field in a floating panel or dialog box); `false` otherwise.

dreamweaver.canDeleteSelection()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can delete the current selection. Depending on the window that has focus, the deletion might occur in the Document window or the Site panel (on the Macintosh, in an text field in a dialog box or floating panel).

Arguments

None.

Returns

A Boolean value: `true` if the opening and closing offsets for the selection are different, which indicates that there is a selection; `false` if the offsets are the same, indicating that there is only an insertion point.

dreamweaver.canExportCSS()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Export CSS Styles operation.

Arguments

None.

Returns

A Boolean value: `true` if the document contains any class styles that are defined in the `HEAD` section; `false` otherwise.

dreamweaver.canExportTemplateDataAsXML()

Availability

Dreamweaver MX.

Description

Checks whether Dreamweaver can export the current document as XML.

Arguments

None.

Returns

A Boolean value: `true` if you can perform an export on the current document; `false` otherwise.

Example

The following example calls `dw.canExportTemplateDataAsXML()` to determine whether Dreamweaver can export the current document as XML and if it returns `true`, calls `dw.ExportTemplateDataAsXML()` to export it:

```
if(dreamweaver.canExportTemplateDataAsXML())
{
  dreamweaver.exportTemplateDataAsXML("file:///c|/dw_temps/mytemplate.txt")
}
```

dreamweaver.canFindNext()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Find Next operation.

Arguments

None.

Returns

A Boolean value: `true` if a search pattern has already been established; `false` otherwise.

dreamweaver.canOpenInFrame()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Open in Frame operation.

Arguments

None.

Returns

A Boolean value: `true` if the selection or insertion point is within a frame; `false` otherwise.

dreamweaver.canPlayRecordedCommand()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Play Recorded Command operation.

Arguments

None.

Returns

A Boolean value: `true` if there is an active document and a previously recorded command that can be played; `false` otherwise.

dreamweaver.canPopupEditTagDialog()

Availability

Dreamweaver MX.

Description

Checks whether the current selection is a tag and whether the Edit Tag menu item is active.

Arguments

None.

Returns

The name of the currently selected tag or a `null` value if no tag is selected.

dreamweaver.canRedo()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Redo operation in the current context.

Arguments

None.

Returns

A Boolean value that indicates whether any operations can be undone.

dreamweaver.canRevertDocument()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Revert (to the last-saved version) operation.

Arguments

`documentObject`

- The *`documentObject`* argument is the object at the root of a document's DOM tree (the value that the `dreamweaver.getDocumentDOM()` function returns).

Returns

A Boolean value that indicates whether the document is in an unsaved state and a saved version of the document exists on a local drive.

dreamweaver.canSaveAll()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Save All operation.

Arguments

None.

Returns

A Boolean value that indicates whether one or more unsaved documents are open.

dreamweaver.canSaveDocument()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Save operation on the specified document.

Arguments

`documentObject`

- The *`documentObject`* argument is the root of a document's DOM (the same value that the `dreamweaver.getDocumentDOM()` function returns).

Returns

A Boolean value that indicates whether the document has any unsaved changes.

dreamweaver.canSaveDocumentAsTemplate()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Save As Template operation on the specified document.

Arguments

`documentObject`

- The *`documentObject`* argument is the root of a document's DOM (the same value that the `dreamweaver.getDocumentDOM()` function returns).

Returns

A Boolean value that indicates whether the document can be saved as a template.

dreamweaver.canSaveFrameset()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Save Frameset operation on the specified document.

Arguments

`documentObject`

- The *`documentObject`* argument is the root of a document's DOM (the same value that the `dreamweaver.getDocumentDOM()` function returns).

Returns

A Boolean value that indicates whether the document is a frameset with unsaved changes.

dreamweaver.canSaveFramesetAs()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Save Frameset As operation on the specified document.

Arguments

`documentObject`

- The *`documentObject`* argument is the root of a document's DOM (the same value that the `dreamweaver.getDocumentDOM()` function returns).

Returns

A Boolean value that indicates whether the document is a frameset.

dreamweaver.canSelectAll()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Select All operation.

Arguments

None.

Returns

A Boolean value that indicates whether a Select All operation can be performed.

dreamweaver.canShowFindDialog()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Find operation.

Arguments

None.

Returns

A Boolean value that is `true` if a Site panel or a Document window is open. This function returns the value `false` when the selection is in the `HEAD` section.

dreamweaver.canUndo()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Undo operation in the current context.

Arguments

None.

Returns

A Boolean value that indicates whether any operations can be undone.

dreamweaver.cssRuleTracker.canEditSelectedRule()

Availability

Dreamweaver MX 2004.

Description

Checks whether the Property Grid editor can be applied to the selected rule. Because the Property Grid can display rules in locked files, a return value of `true` does not guarantee that the rule can be modified.

Arguments

None.

Returns

A Boolean value: `true` if the Property Grid editor can be applied to the selected rule; `false` otherwise.

Example

The following code checks whether the enabler function has been set to the value `true` before allowing edits to the selected rule:

```
if(dw.cssRuleTracker.canEditSelectedRule()){
  dw.cssRuleTracker.editSelectedRule();
}
```

dreamweaver.cssStylePalette.canApplySelectedStyle()

Availability

Dreamweaver MX.

Description

Checks the current active document to see whether the selected style can be applied.

Arguments

None.

Returns

A Boolean value: `true` if the selected style has a class selector; `false` otherwise.

dreamweaver.cssStylePalette.canDeleteSelectedStyle()

Availability

Dreamweaver MX.

Description

Checks the current selection to determine whether the selected style can be deleted.

Arguments

None.

Returns

A Boolean value: `true` if the selection can be deleted; `false` otherwise.

dreamweaver.cssStylePalette.canDuplicateSelectedStyle()

Availability

Dreamweaver MX.

Description

Checks the current active document to see whether the selected style can be duplicated.

Arguments

None.

Returns

A Boolean value: `true` if the selected style can be duplicated; `false` otherwise.

dreamweaver.cssStyle.canEditSelectedStyle()

Availability

Dreamweaver MX.

Description

Checks the current active document to see whether the selected style can be edited.

Arguments

None.

Returns

A Boolean value: `true` if the selected style is editable; `false` otherwise.

dreamweaver.cssStylePalette.canEditStyleSheet()

Availability

Dreamweaver MX.

Description

Checks the current selection to see whether it contains style sheet elements that can be edited.

Arguments

None.

Returns

A Boolean value: `true` if the selection is a style sheet node or a style definition within a style sheet node and the style sheet is neither hidden nor this document; `false` if the selection is hidden or in this document.

dreamweaver.isRecording()

Availability

Dreamweaver 3.

Description

Reports whether Dreamweaver is currently recording a command.

Arguments

None.

Returns

A Boolean value that indicates whether Dreamweaver is recording a command.

dreamweaver.htmlStylePalette.canEditSelection()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can edit, delete, or duplicate the selection in the HTML Styles panel.

Arguments

None.

Returns

A Boolean value: `true` if Dreamweaver can edit, delete, or duplicate the selection in the HTML Styles panel; `false` if no style is selected or if one of the clear styles is selected.

dreamweaver.resultsPalette.canClear()

Availability

Dreamweaver MX.

Description

Checks whether you can clear the contents of the Results panel that is currently in focus.

Arguments

None.

Returns

A Boolean value: `true` if the contents can clear; `false` otherwise.

dreamweaver.resultsPalette.canCopy()

Availability

Dreamweaver MX.

Description

Checks whether the current Results window can display a copied message in its contents.

Arguments

None.

Returns

A Boolean value: `true` if the contents can display; `false` otherwise.

dreamweaver.resultsPalette.canCut()

Availability

Dreamweaver MX.

Description

Checks whether the current Results window can display a Cut message in its contents.

Arguments

None.

Returns

A Boolean value: `true` if the contents can display; `false` otherwise.

dreamweaver.resultsPalette.canPaste()

Availability

Dreamweaver MX.

Description

Checks whether the current Results window can display a Paste message in its contents.

Arguments

None.

Returns

A Boolean value: `true` if the contents can display; `false` otherwise.

dreamweaver.resultsPalette.canOpenInBrowser()

Availability

Dreamweaver MX.

Description

Checks whether the current report can display in a browser.

Arguments

None.

Returns

A Boolean value: `true` if the contents can display; `false` otherwise.

dreamweaver.resultsPalette.canOpenInEditor()

Availability

Dreamweaver MX.

Description

Checks whether the current report can display in an editor.

Arguments

None.

Returns

A Boolean value: `true` if the contents can display; `false` otherwise.

dreamweaver.resultsPalette.canSave()

Availability

Dreamweaver MX.

Description

Checks whether the Save dialog box can open for the current panel. Currently, the Site Reports, Target Browser Check, Validation, and Link Checker panels support the Save dialog box.

Arguments

None.

Returns

A Boolean value: `true` if the Save dialog box can appear; `false` otherwise.

dreamweaver.resultsPalette.canSelectAll()

Availability

Dreamweaver MX.

Description

Checks whether a Select All message can be sent to the window that is currently in focus.

Arguments

None.

Returns

A Boolean value: `true` if the Select All message can be sent; `false` otherwise.

dreamweaver.snippetpalette.canEditSnippet()

Availability

Dreamweaver MX.

Description

Checks whether you can edit the currently selected item and returns either a `true` or `false` value so you can enable or disable menu items for editing.

Arguments

None.

Returns

A Boolean value: `true` if you can edit the currently selected item; `false` otherwise.

dreamweaver.snippetpalette.canInsert()

Availability

Dreamweaver MX.

Description

Checks whether you can insert or apply the selected element and returns either a `true` or `false` value so you can enable or disable menu items for inserting or applying.

Arguments

None.

Returns

A Boolean value: `true` if you can insert or apply the selected element; `false` otherwise.

site.browseDocument()

Availability

Dreamweaver 4.

Description

Opens all selected documents in a browser window. It is the same as using the Preview in Browser command.

Arguments

`browserName`

- The *`browserName`* argument is the name of a browser as defined in the Preview in Browser preferences. If omitted, this argument defaults to the user's primary browser.

Returns

Nothing.

site.canAddLink()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform an Add Link to [Existing File | New File] operation.

Arguments

None.

Returns

A Boolean value: `true` if the selected document in the site map is an HTML file; `false` otherwise.

site.canChangeLink()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Change Link operation.

Arguments

None.

Returns

A Boolean value: `true` if an HTML or Flash file links to the selected file in the site map; `false` otherwise.

site.canCheckIn()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Check In operation.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

A Boolean value: `true` if the following conditions are true; `false` otherwise:

- A remote site has been defined.
- If a document window has focus, the file has been saved in a local site; or, if the Site panel has focus, one or more files or folders are selected.
- The Check In/Check Out feature is turned on for the site.

site.canCheckOut()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Check Out operation on the specified file or files.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

A Boolean value: `true` if all the following conditions are true; `false` otherwise:

- A remote site has been defined.
- If a document window has focus, the file is part of a local site and is not already checked out; or, if the Site panel has focus, one or more files or folders are selected and at least one of the selected files is not already checked out.
- The Check In/Check Out feature is turned on for the site.

site.canCloak()

Availability

Dreamweaver MX.

Description

Determines whether Dreamweaver can perform a Cloaking operation.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the `canCloak()` function should act on the selection in the Site panel or the URL of a particular folder, which indicates that the `canCloak()` function should act on the specified folder and all its contents.

Returns

A Boolean value: `true` if Dreamweaver can perform the Cloaking operation on the current site or the specified folder; `false` otherwise.

site.canConnect()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can connect to the remote site.

Arguments

None.

Returns

A Boolean value: `true` if the current remote site is an FTP site; `false` otherwise.

site.canFindLinkSource()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Find Link Source operation.

Arguments

None.

Returns

A Boolean value that indicates that the selected link in the site map is not the home page.

site.canGet()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Get operation.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

If the argument is `site`, a Boolean value that indicates whether one or more files or folders is selected in the Site panel and a remote site has been defined. If the argument is a URL, a Boolean value that indicates whether the document belongs to a site for which a remote site has been defined.

site.canLocateInSite()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Locate in Local Site or Locate in Remote Site operation (depending on the argument).

Arguments

`localOrRemote`, *`siteOrURL`*

- The *`localOrRemote`* argument must be either `local` or `remote`.
- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

One of the following values:

- If the first argument is the keyword `local` and the second argument is a URL, a Boolean value that indicates whether the document belongs to a site
- If the first argument is the keyword `remote` and the second argument is a URL, a Boolean value that indicates whether the document belongs to a site for which a remote site has been defined, and, if the server type is Local/Network, whether the drive is mounted
- If the second argument is the keyword `site`, a Boolean value that indicates whether both windows contain site files (not the site map) and whether the selection is in the opposite pane from the argument

site.canMakeEditable()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Turn Off Read Only operation.

Arguments

None.

Returns

A Boolean value: `true` if Dreamweaver can perform a Turn Off Read Only operation; `false` if one or more of the selected files is locked.

site.canMakeNewFileOrFolder()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a New File or New Folder operation in the Site panel.

Arguments

None.

Returns

A Boolean value: `true` if any files are visible in the selected pane of the Site panel; `false` otherwise.

site.canOpen()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can open the files or folders that are currently selected in the Site panel.

Arguments

None.

Returns

A Boolean value: `true` if any files or folders are selected in the Site panel; `false` otherwise.

site.canPut()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Put operation.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the function should act on the selection in the Site panel, or the URL for a single file.

Returns

One of the following values:

- If the argument is the keyword `site`, returns the value `true` if any files or folders are selected in the Site panel and a remote site has been defined; otherwise `false`.
- If the argument is a URL, returns the value `true` if the document belongs to a site for which a remote site has been defined; otherwise `false`.

site.canRecreateCache()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Recreate Site Cache operation.

Arguments

None.

Returns

A Boolean value: `true` if the Use Cache To Speed Link Updates option is enabled for the current site.

site.canRefresh()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Refresh [Local | Remote] operation.

Arguments

`localOrRemote`

- The *`localOrRemote`* argument must be either the `local` or `remote` keyword.

Returns

A value of `true` if the *`localOrRemote`* argument is the `local` keyword; otherwise, a Boolean value that indicates whether a remote site has been defined.

site.canRemoveLink()

Availability

Dreamweaver 3.

Description

Checks whether Dreamweaver can perform a Remove Link operation.

Arguments

None.

Returns

A Boolean value that indicates that an HTML or Flash file links to the selected file in the site map.

site.canSetLayout()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Layout operation.

Arguments

None.

Returns

A Boolean value: `true` if the site map is visible; `false` otherwise.

site.canSelectAllCheckedOutFiles()

Availability

Dreamweaver 4.

Description

Determines whether the current working site has the Check In/Check Out feature enabled.

Arguments

None.

Returns

A Boolean value: `true` if the site allows Check In/Check Out; `false` otherwise.

site.canSelectNewer()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Select Newer [Remote | Local] operation.

Arguments

`localOrRemote`

- The *`localOrRemote`* argument must be either the `local` or `remote` keyword.

Returns

A Boolean value that indicates whether the document belongs to a site for which a remote site has been defined.

site.canShowPageTitles()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Show Page Titles operation.

Arguments

None.

Returns

A Boolean value: `true` the site map is visible; `false` otherwise.

site.canSynchronize()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a Synchronize operation.

Arguments

None.

Returns

A Boolean value that indicates whether a remote site has been defined.

site.canUncloak()

Availability

Dreamweaver MX.

Description

Determines whether Dreamweaver can perform an uncloaking operation.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the `canUncloak()` function should act on the selection in the Site panel or the URL of a particular folder, which indicates that the `canUncloak()` function should act on the specified folder and all its contents.

Returns

A Boolean value: `true` if Dreamweaver can perform the uncloaking operation on the current site or the specified folder; `false` otherwise.

site.canUndoCheckOut()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform an Undo Check Out operation.

Arguments

`siteOrURL`

- The *`siteOrURL`* argument must be the `site` keyword, which indicates that the function should act on the selection in the Site panel or the URL for a single file.

Returns

A Boolean value: `true` if the specified file or at least one of the selected files is checked out.

site.canViewAsRoot()

Availability

Dreamweaver 3.

Description

Determines whether Dreamweaver can perform a View as Root operation.

Arguments

None.

Returns

A Boolean value: `true` if the specified file is an HTML or Flash file; `false` otherwise.

INDEX