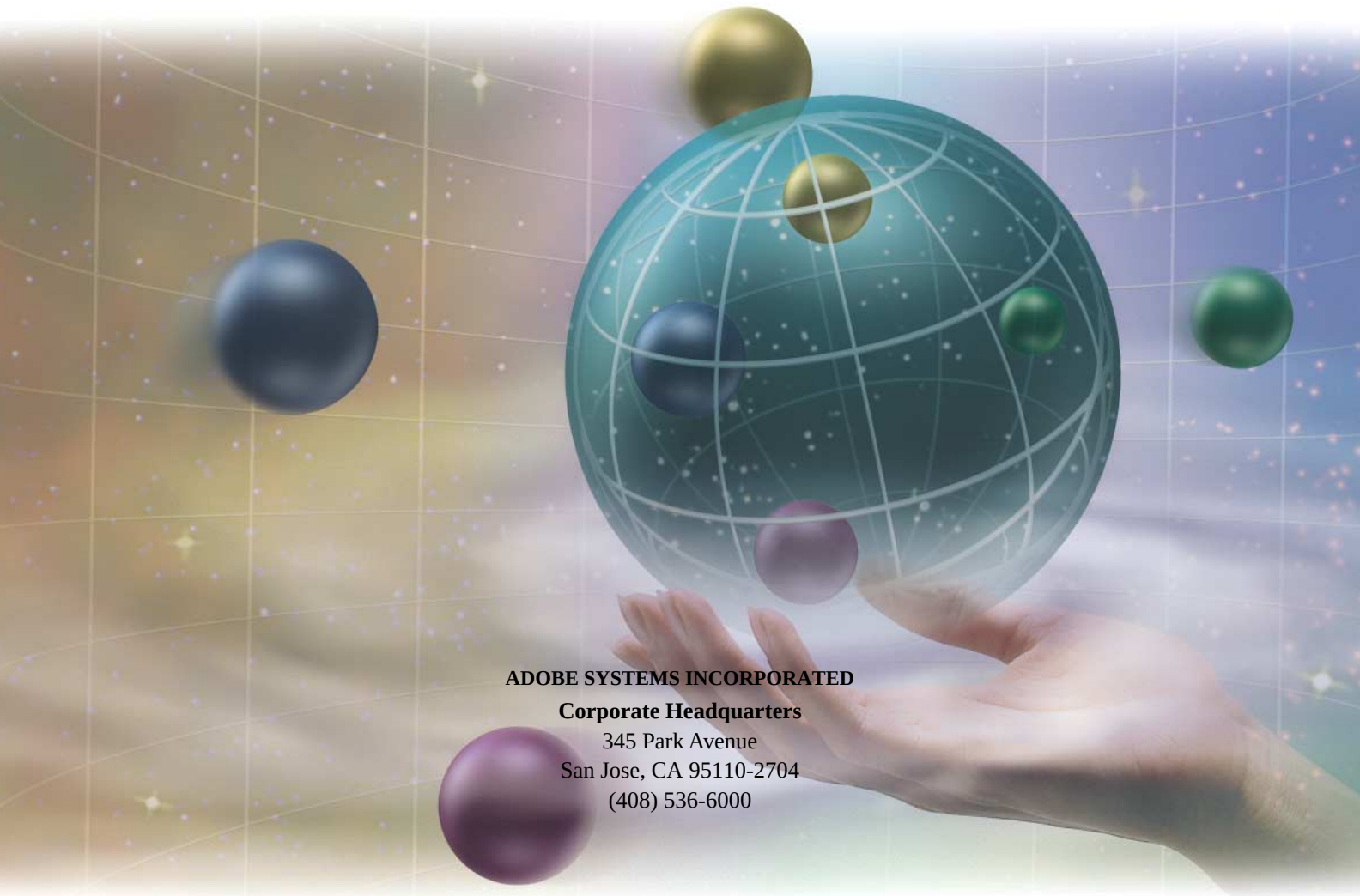
GoLive® 6.0 Extend Script SDK Reference

SDK Release 6.0r1
For Adobe® GoLive Version 6.0



ADOBE SYSTEMS INCORPORATED
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

Adobe® GoLive® 6.0 Extend Script SDK Reference for Windows and Mac OS.

Contents

Contents

Figures and Tables

Welcome

Welcome to the Adobe *GoLive® 6.0 Extend Script SDK Reference*. This preface describes how to use this book and its companion volume, the *GoLive 6.0 Extend Script SDK Programmer's Guide*, to extend the GoLive website development environment. The initial version of this book is provided on the GoLive 6.0 CD. The *GoLive 6.0 Extend Script SDK Programmer's Guide* and updates to this book are available at `www.adobe.com`.

Finding Information

To facilitate access to its contents, this book provides:

- a table of Contents
- an Index
- an Acrobat® Catalog index file (`index.pdx`), which enhances searches of this PDF document

NOTE: You must enable the Catalog index file before you can use it; for more information, choose the **Help>Acrobat Guide** menu item in Adobe Acrobat, then click the **Searching Catalog Indexes** bookmark. The index file and its associated folder are in the **GoLive SDK Documentation** folder with the PDF file you are now reading.

The Glossary in the *GoLive 6.0 Extend Script SDK Programmer's Guide* gathers in one place all specialized terms this book defines.

About The GoLive Extend Script SDK

GoLive is an application that helps you create HTML files. The GoLive Extend Script SDK enables you to extend the behavior and user interface of Adobe GoLive.

Using the GoLive Extend Script SDK, you can create tools tailored to your specific GoLive tasks. To make these new behaviors and tools available to the user, the SDK can

use, customize, and extend most aspects of the GoLive user interface, such as

- Menus and menu items
- Floating palettes
- Task-specific dialogs that include text, graphics, and controls
- Custom HTML elements , such as `<mytag>`, that can be edited in the Inspector
- Customized content as drag-and-drop items in the Objects palette
- Site Window
- Site Reports
- Doument parsing options for encoding, translation, localization, and non-HTML tags.

Because the files that define extensions use HTML syntax, you can use GoLive itself — including your own custom extensions — to create additional extensions to the GoLive design environment. In the same way that you can create JavaScript scripts to generate and manipulate HTML files, you can even write JavaScript scripts to generate and manipulate GoLive extensions. Using this technique you could, for example, use JavaScript scripts to customize menu items in GoLive according to the contents of a database.

The range of tasks an extension might perform in GoLive is almost unlimited — virtually all of the user commands in GoLive are made available in JavaScript. For example, using JavaScript to automate repetitive tasks, you can edit all the documents on your site programmatically.

Creating an Extend Script extension is similar to creating a Web page or a Web application: to design the appearance of your extension you use the GoLive user interface to add SDK-provided tags to an HTML document. These tags let you create menus, dialogs, and other user interface items. The SDK installs, displays, and manages these user interface items in the GoLive environment.

When the user interacts with one of your extension's user interface items, the SDK calls one or more JavaScript functions that you have created to provide the extension item's behavior.

The SDK enables even inexperienced JavaScript users to create simple extensions with custom menus and dialogs easily. Yet it is comprehensive and powerful: The SDK provides numerous JavaScript objects and methods to perform tasks on a document, on a site, in the GoLive environment, on local and remote file systems, and on DAV servers.

Optionally, Extend Script extensions can call custom libraries written in the C, C++ or JavaScript programming languages. You can even use XML to define entirely new structured markup languages and documents to GoLive—in effect, GoLive's extensibility is unlimited.

However, you don't need to know C languages or XML to use this SDK. If you've used HTML to create Web page content, and perhaps added some interactivity to that page with JavaScript, you're already familiar with the concepts behind Extend Script extensions.

About This Book

The *GoLive 6.0 Extend Script SDK Reference* provides reference descriptions of the markup tags, JavaScript objects, JavaScript methods, and C-language data structures and functions that are provided by the GoLive Extend Script SDK.

This book is a companion to the *GoLive 6.0 Extend Script SDK Programmer's Guide*, which describes how to use the GoLive Extend Script SDK to add functionality and custom user interface elements to version 5.0 of Adobe GoLive.

This book does not document the JavaScript language nor how to use GoLive. For a listing of some helpful publications, see Appendix F, "Additional Resources," on page 401.

Who Should Read This Book

This book is for anyone who wants to extend the capabilities of Adobe GoLive using JavaScript and the special markup tags that the GoLive Extend Script SDK provides.

What is in This Book

This book provides detailed reference information about the markup tags and JavaScript objects the GoLive Extend Script SDK provides. It does not present detailed code analyses or lengthy explanations of conceptual information; for this material, see the *GoLive 6.0 Extend Script SDK Programmer's Guide*.

Chapter Summaries

Here is a brief, chapter-by-chapter overview of this book.

- "Welcome" is this preface. It describes the content of this book.
- Chapter 1, "What is an Extension?" is a two-page description of what an extension is and what extensions can do.
- Chapter 2, "How To Create An Extension," is a one-page graphic showing four steps to creating an extension.

- Chapter 3, “Using JavaScript In GoLive,” describes how GoLive makes JavaScript objects available to extensions.
- Chapter 4, “Objects,” describes the JavaScript objects, properties, and functions in the GoLive environment. This chapter does not describe objects that are standard parts of the JavaScript language or its Document Object Model.
- Chapter 5, “Tags,” describes markup tags that the GoLive Extend Script SDK supplies. You use these tags to define dialogs, palettes, controls, and custom elements your extension adds to the GoLive environment.
- Chapter 6, “Event-Handling Functions,” describes optional functions and methods your extension can implement to respond to events such as those generated by the user’s interaction with your extension’s controls.
- Appendix A, “Sort Order Tables,” provides the numeric values GoLive uses to sort icons and menu items.
- Appendix B, “Supported Layout Tags,”contains reference descriptions of markup tags that the Layout objects in GoLive can manage.
- Appendix C, “DAV Objects,” describes additional JavaScript objects that extensions can use to work with WebDAV servers.
- Appendix D, “C and C++ APIs For Use In External Binary Libraries,” is an advanced section that describes data types and utility functions for creating external binary libraries that GoLive extensions can use.
- Appendix E, “Additional Topics,” discusses advanced topics of interest to a limited audience, such as how to maintain compatibility with previous versions of GoLive, and a detailed discussion of the scope of `name` attributes extensions define.
- Appendix F, “Additional Resources,” lists additional sources of information on GoLive and JavaScript.

Document Conventions

This document uses the following typographic and terminology conventions.

Typographical conventions

Boldface font identifies the first use and definition of a term.

`Courier font` identifies code, such as JavaScript code, HTML code, filenames, and pathnames. In pathnames, the forward slash (/) is used as a directory separator; for example the `Samples/Main.html` notation refers to the `Main.html` file in the `Samples` folder.

Italic text identifies replaceable text in code; for example, the *myName* text in `name="`*myName*`"` represents a value you are expected to supply. Thus `name="`*myName*`"` represents code such as `name="Fred",` or `name="Homer",` and so on. When the replaceable text ends in "*Obj*", it denotes a an object; the rest of the replaceable text identifies the object's type. Thus "*pictureObj*" means "this gets replaced by a picture object".

Blue underlined text signifies a hyperlink you can click to display related information.

GoLive menus and menu items are listed in **sans-serif bold font**. The **>** symbol is used as shorthand notation for navigating to menu items; for example the **Edit>Cut** item refers to the **Cut** item in the **Edit** menu.

NOTE: Notes like this one highlight important points that deserve extra attention.

IMPORTANT: *Important notes like this one point out key requirements and common errors that may cause a GoLive extension to not work or to work incorrectly.*

JavaScript Conventions

Because most objects provided by the SDK provide a `name` property, the Objects chapter of the *GoLive 6.0 Extend Script SDK Reference* does not list `name` properties explicitly. Similarly, this reference does not list properties and methods provided by the JavaScript language itself. For example, it is common for JavaScript objects to provide a `toString` method, and many of the objects the SDK supplies implement this method; however, this book does not describe such methods unless they differ from the standard JavaScript implementation.

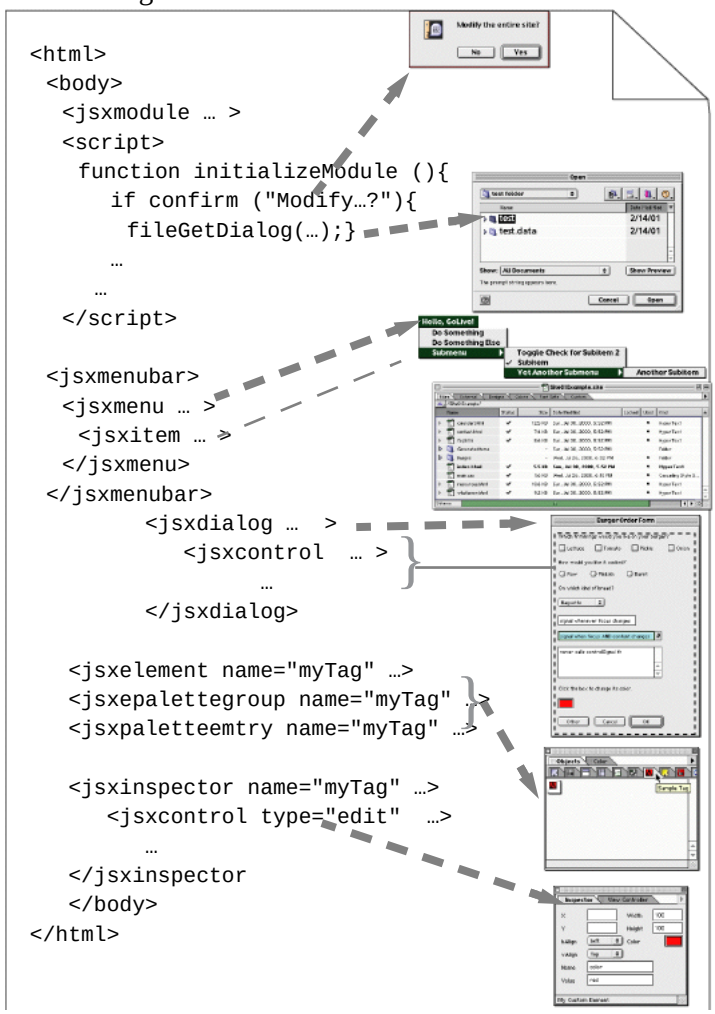
When a JavaScript function returns a value, we list it; when it does not return a value, it has no "Returns" section.

1 What is an Extension?

Like a plug-in, an extension provides new capabilities to its host environment. Extend Script is the Adobe technology that allows you to use JavaScript and HTML to extend and customize Adobe GoLive web design environment.

About Extend Script Extensions

An Extend Script extension is actually an HTML file that GoLive uses in a special way. Creating an extension is similar to creating a Web page: you add tags and scripts to this file to define content. Instead of defining content for a web page, however, you use special SDK-supplied <jsx … > tags that define menus, dialog, palettes, inspectors and custom tools in the GoLive design environment.



An extension can also incorporate normal HTML tags and other kinds of tags, like server-side tags. An extension can even define its own customized tags and content, such as a `<myTag>` tag.

But the content an extension creates is only half the story—an Extend Script extension can obtain content and services from the GoLive application, from other extensions, and from resources on local and remote file systems.

What Can Extensions Do?

One of the most popular uses of Extend Script is the programmatic editing of markup files written in HTML, XML, ASP, JSP, and other such languages.

To ease your learning curve, GoLive repurposes familiar tools for this task. The JavaScript DOM in GoLive works just like the one in a Web browser, providing programmatic access to the markup elements in an HTML file. This DOM enables Layout View in GoLive to edit HTML files just as a browser or other HTML editor would.

The SDK can operate directly on documents in Layout View, allowing you to create extensions that automate the creation or modification of HTML pages. The SDK can also operate on documents without displaying them in Layout View, enabling extensions to process batches of files rapidly.

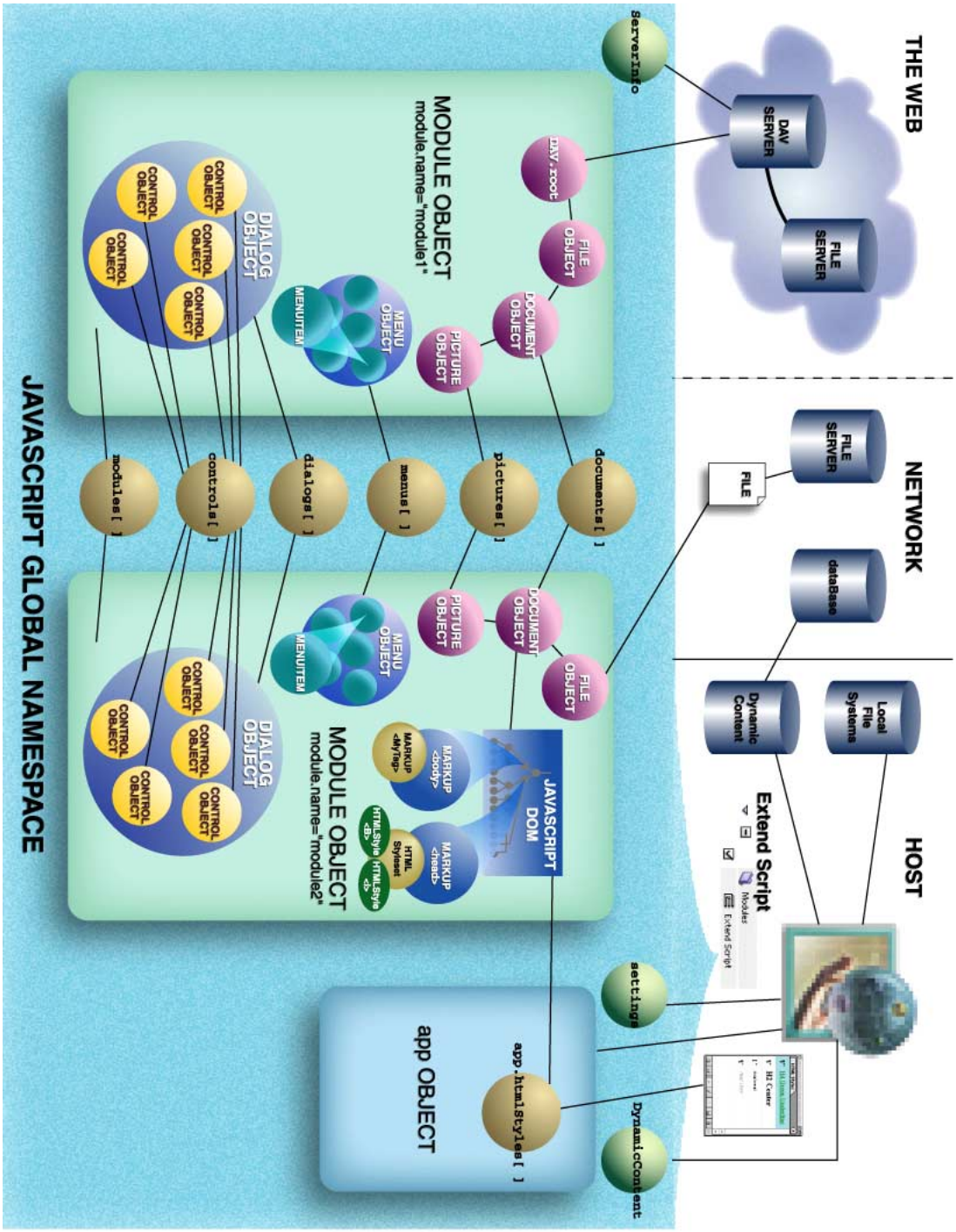
In addition to supporting standard JavaScript events, GoLive provides additional events that support the programmatic modification of documents and sites. Supporting an event in your extension is similar to supporting one in a Web page: you define an appropriately-named JavaScript function within the HTML file's `<script>` element. For example, to respond to the `docSignal` event, you define a `docSignal()` function that appears inside a `<script>` element in the `main.html` file that defines the extension.

The GoLive DOM supports not just HTML, but XML(Extensible Markup Language). This ability enables GoLive to recognize other markup languages, such as those which define server-side tags, as well as the special `<jsx... >` tags the SDK provides. As a result, you can edit an extension's `main.html` file (or any of the other file types noted) just like any other HTML file.

GoLive 6 provides enhanced support for site-level operations, including notification of changes in the content or organization of the Site window, and programmatic selection of site assets according to criteria you specify. For example, an extension can select all files in the site that would take longer than a minute to download at 56kbps.

In addition to its extended DOM and event model, the GoLive Object Model provides access to the GoLive application itself, which in turn makes available a host of other services and content.

FIGURE 1.1 Extensions get content and services from app, DOM, other extensions



The GoLive application provides access to

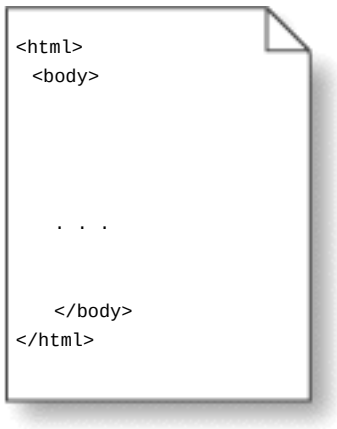
- The GoLive user interface (menus, dialogs, palettes, Inspectors, site window, site reports, built-in localization)
- User settings, global stylesheets, preferences, shared data, other extensions
- Custom/server tag support and special parsing behavior

  You can create extensions that provide your own customized tags and content as icons in the Objects palette. You can also preprocess documents before the GoLive parser reads them, to deal with encoding issues or to mimic the effects of server-side code in Layout view, which is the graphical editing view in GoLive.
- Automation and macros: apply automated edits to every file in a site, or generate entire sites programmatically.
- Dynamic Content database content
- Resources on network and WebDAV servers.

The typical extension can and does, of course, define things for itself in its own code, such as its own user interface items, custom functions, custom tags, SDK-provided tags, HTML tags, localization, source translation, and more. To accomplish this, you'll just add tags and functions to an extension's `main.html` file as you might add them to a Web page: tags go in the file's `<head>` and `<body>` elements, and JavaScript functions go in the file's `<script>` elements.

XML support enables even further extensibility, allowing GoLive to grow rapidly and conform easily to new standards that may emerge in the future.

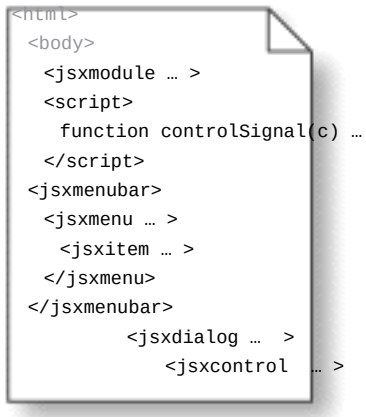
2 How To Create An Extension



Create

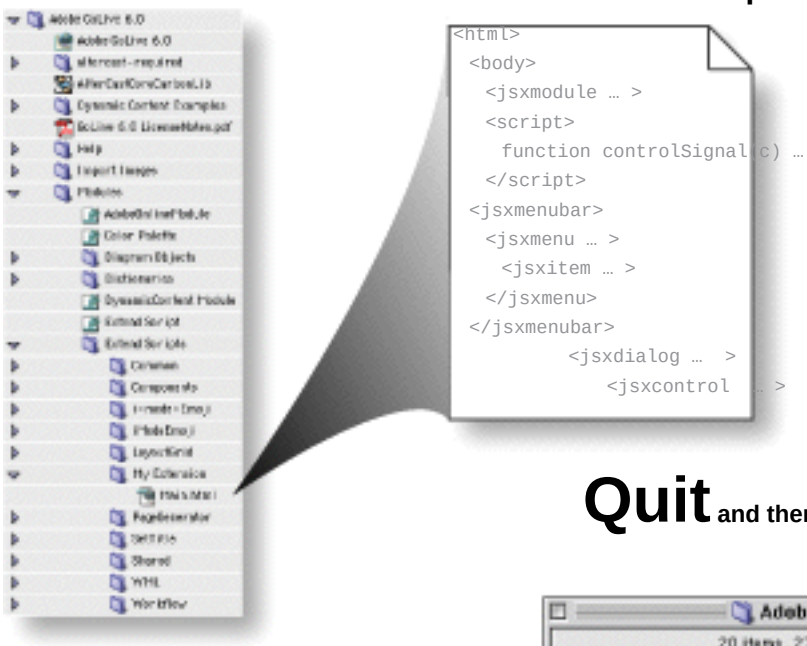
an HTML file
named `main.html`

Add SDK-provided tags and scripts that define your extension



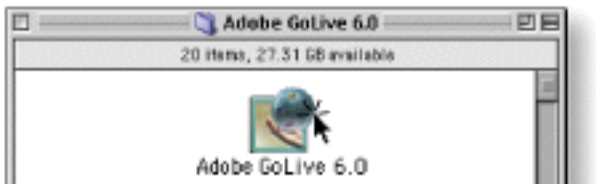
Place the `main.html` file

in its own subfolder of the
`Adobe GoLive 6.0/Modules/Extend Scripts/` folder



Quit and then Restart

Adobe GoLive



Where To Go Next...

- For a less-terse introduction to extension development, see Chapter 1, “Getting Started,” in the *Extend Script SDK Programmer’s Guide*.
- The SDK provides JavaScript objects that offer access to the GoLive application, documents, sites, and other resources. See Chapter 4, “Objects,” on page 29, for descriptions of JavaScript objects, properties, and methods the GoLive environment provides.
- For descriptions of the SDK-provided markup tags you use to define static data, such as the dialogs in your extension’s user interface, see Chapter 5, “Tags.”
- The SDK sends messages to announce events, such as a change in the state of a control or document. To respond to an event, your extension implements the corresponding JavaScript global function that Chapter 6, “Event-Handling Functions” describes. You need not implement event-handling functions that correspond to features your extension does not use.
- For additional details about the JavaScript environment in GoLive, see Chapter 3, “Using JavaScript In GoLive.”

3 Using JavaScript In GoLive

GoLive provides access to data and objects in a way that JavaScript programmers will find familiar. However, the GoLive Extend Script SDK is also intended to be useful to those who don't want to dive too deeply into the fine print of JavaScript. Thus, the GoLive environment usually provides multiple ways to access data and objects. This section describes various ways an extension's JavaScript code can access data and objects in the GoLive environment.

Attributes of Elements as Properties of JavaScript Objects

An **element** is defined by a **tag** and its attributes. The attributes of an element appear as the properties of the JavaScript object GoLive creates to represent the element. For example, the `name` attribute in the following control element appears as the `name` property of the control object GoLive creates when it interprets the markup element.

```
<jsxcontrol name="myControl" type=static value="good">
```

The value of an element's `name` attribute must follow JavaScript naming conventions. If more than one element or object uses the same name, the results of name-based object retrieval are unpredictable; thus, you should take care to ensure that your name attribute values are unique.

Use of a unique prefix or postfix in all of your extension's names can help to ensure their uniqueness; for example, the following element definitions begin the value of each element's `name` attribute with the letters `ADBE`.

```
<jsxmenubar> // opens definition of all menus
   <jsxmenu name="ADBEHello" title="Hello, GoLive!"> // Hello menu
      <jsxitem name="ADBEThis" title="Do Something"> // menu item
      <jsxitem name="ADBEThat" title="Do Something Else" > menu item
   </jsxmenu> // closes definition of Hello menu
</jsxmenubar> // closes definition of all menus
```

When the SDK interprets these markup elements, it creates a menu object that appears in the JavaScript global `menus` collection. To retrieve this menu from the array, an extension's JavaScript code can use the value of the menu object's `name` attribute as an index into the array. You can retrieve the `ADBEHello` menu as the following line of JavaScript does.

```
var myMenu = menus["ADBEHello"];
```

The precise means by which you obtain a reference to a JavaScript object depends on the object's type. You can retrieve most of an extension's objects by name from global collections such as the `menus` and `dialogs` collections. Objects that represent HTML page content are available from the markup tree object the page's document provides. For information on retrieving a particular object, see that object's description in this Reference.

Access to JavaScript properties is case-sensitive; that is, the `Thing` attribute creates the `Thing` property, not the `thing` property; when writing JavaScript code, observe case accordingly.

Although HTML allows dashes or hyphens in attribute names, JavaScript does not allow hyphens in identifiers. The JavaScript properties such attributes create cannot be accessed by means of the usual syntax.

```
anObject.hyphenated-name // normal dot-operator syntax won't work
```

To retrieve property names that include hyphens (dashes), such as `background-color`, use the following syntax.

```
anObject["hyphenated-name"]// retrieving a hyphenated property name
```

Name-Based Access to Objects

Most objects in the GoLive environment can be retrieved by name. Commonly, the `name` property of a JavaScript object is specified by the `name` attribute of the markup element GoLive interpreted to create it.

IMPORTANT: *To ensure reliable name-based access to JavaScript objects, the* `name` *attribute of each element your extension defines must be unique within the JavaScript namespace. If the value of an element's* `name` *attribute or an object's* `name` *property is not unique within this namespace, your code may not run reliably, it may not run at all, or you may not be able to retrieve the non-uniquely named items reliably from collections such as those provided by the* `dialogs` *or* `controls` *global properties.*

You should try to name all of your JavaScript objects uniquely. Most SDK objects are hindered by non-unique names; however a few of them actually use non-unique names for special purposes:

- You can use a non-unique JavaScript name to add menu items to an existing menu.
- Control names used in different dialogs can be non-unique; for example, all of your extension's dialogs could reuse the same `myOK` button.

Although the SDK tolerates and in these cases makes use of non-unique names, such items cannot be retrieved by name reliably. For a more detailed discussion, see "JavaScript Scope of Name Attribute Values" on page 396.

If you omit the `name` attribute from an element you define, GoLive usually generates a unique `name` attribute for you; however, it is recommended that you choose your own names, If you need to retrieve an object later, name it yourself so you'll know exactly which JavaScript name is associated with it.

JavaScript Object Collections

Upon creation, the SDK makes commonly-used objects available as the elements of array-like structures that all extensions can access. GoLive updates the contents of these structures dynamically as these objects are created and deleted.

Each of these structures is known as a Collection Object. You can think of a collection object as an array that provides access to its elements by name. It is important to note, however, that Collection objects are not arrays. Not every collection provides numeric access to its elements, as an Array object does.

Table 3.1 summarizes the characteristics of the collection objects GoLive provides.

TABLE 3.1 Collection Object Access

Object	JavaScript Access	Indexes		Contents
		Number	Name	
BoxCollection	`boxes` global variable	√	√	Read-only array of all boxes in the current document.
ControlCollection	`controls` global variable	AGL5 only	√	The current document's controls that have run at least once.
DialogCollection	`dialogs` global variable	√	√	The current document's dialogs that have run at least once. Read-only.
DocumentCollection	`documents` global variable	√	√	Documents open in GoLive.
History	`document.history` property	√	√	Undo actions involving `document`
HTMLStyleSetCollection	`app.htmlStyles` property	√	√	Every HTMLStyleSet Object in the **Window>Styles** palette. AGL6 only.
LinkCollection	*boxObj*`.links` property	√	√	Links to all files that reference this box (in links) and all files this box references (out links).
MarkupCollection	*markupObj*`.subElements` `document.element.subElements`	√	√	Immediate subelements of the *markup* object.
MenuCollection	`menus` global variable	√	√	All menus currently available in GoLive.
MenuItemCollection	`menus[`*value*`].items` property *menuObj*`.items.`*propertyName*	√	√	All the menu items belonging to the `menus[`*value*`]` menu object.
ModulesCollection	`modules` global variable	√	√	All currently-running GoLive Extend Script extensions.
PictureCollection	`pictures` global variable	√	√	All pictures accessible to this module.
WebsiteCollection	`websites` global variable	√	√	All websites currently open in GoLive.

Each of these global properties contains a Collection Object that GoLive updates dynamically as objects are created and destroyed.

Retrieving Objects

This section provides examples that show various ways extensions can retrieve objects by JavaScript name in GoLive. These examples are based on the use of the `menus` array, which provides access to all of the menus and menu items added to GoLive by **Extend Script** extensions. Except as noted, all of the other arrays in Table 3.1 work the same way.

To use a collection, access it as Table 3.1 indicates; for example, the following line of code retrieves the **Sample** menu from the `MenuCollection` object the `menus` global variable provides. It then stores the retrieved Menu object in the `sampMenu` variable.

```
var sampMenu = menus ["sample"]
```

Theoretically, you could also retrieve the **Sample** menu by numeric index; however, doing so poses challenges—extensions other than your own can also add menus to this array, so you can't easily determine the numeric index that provides a particular object. You'll encounter the same problem when working with the other object arrays, too—other extensions can add items to these arrays as well. Most of the time, you'll find that an object's unique `name` property provides the most expedient and reliable way to retrieve it.

Sub-elements are generally made available as the properties of a parent element; for example, the SDK makes menu item objects available as properties of the menu in which they appear. The parent menu may be accessed by means of its unique `name` attribute, or by means of the collection the `menus` global property provides.

Many collections the SDK provides can be addressed by numeric index as well as by name; for example, the following lines of JavaScript are equivalent.

```
menus ["sample"].items ["item1"]
menus ["sample"].items [0] // assume item1 is the first menu item
menus ["sample"].item1
sample.item1
```

The first line of JavaScript retrieves the `item1` menu item by name from the `sample` menu, which it also retrieves by name.

```
menus ["sample"].items ["item1"]
```

The `menus ["sample"]` expression gets the menu named `sample` by name from the global `menus` array. The `sample` menu's `items` property holds the array of menu items. To retrieve the `item1` element from this array by name, we use the same technique we used to get the `sample` element from the global `menus` array.

Alternatively, you can retrieve the `item1` menu item as a property of the `sample` menu, as in the next line of JavaScript.

```
menus ["sample"].item1
```

GoLive also makes each menu available as a JavaScript object in the global namespace; thus, the following simple line of JavaScript provides yet another way to retrieve the first item in the `sample` menu.

```
sample.item1
```

IMPORTANT: *To avoid unpredictable results, the JavaScript name of each element your extension defines must be unique within the global namespace. If the value of a tag's* `name` *attribute or an object's* `name` *property is not unique within the JavaScript namespace, your code may not run reliably or it may not run at all.*

Comparing Objects

To ascertain an object's identity, you can compare the value of its `name` property to a known value, such as a string or a reference to a global object. For example, you could test the name of a menu item in any of the following ways:

```
if (item.name == "item1") // compare object name to known string value
if (item == menus ["sample"].items ["item1"]) // compare objects
if (item == sample.item1) // another object comparison example
```

The `name` property holds a string; thus, the first line of code performs a simple string comparison. The second and third lines perform object comparisons, and demonstrate different ways to retrieve the menu item to compare.

Errors in Drawing Functions

If the SDK detects an error in a drawing function, such as in the `drawControl` or `drawBox` function, it does not call this function again until you close and reopen the document that defines the function. Without this feature, an error in a drawing function would cause GoLive to throw an endless stream of errors, because drawing functions are called whenever a JavaScript object must be redrawn.

Saving References To Objects

If you save a reference to an object that GoLive generated as the result of interpreting a markup tag, you must update that reference any time the document containing the tag changes. For details, see Chapter 5, "Documents."

For optimum performance, release unused memory as soon as possible. Set unneeded variables to `null` to make them available for garbage collection.

Before unloading your extension, GoLive calls its `terminateModule` function. Your implementation of this function must release all of your extension's saved JavaScript references by setting to `null` the values of all global variables your extension creates.

User Preference Data

Extension modules can use a prefs Object to store persistent user preference data that is available to all extensions; for more information, see the description of the prefs Object in this chapter.

The app.prefs Object provides read-only access to a subset of GoLive's global preferences. This object is in the `prefs` property of the application object, which is available in the `app` global property; thus, the `app.prefs` JavaScript expression returns the global preferences object. For more information, see "app.prefs Object" on page 56.

4 Objects

This chapter describes the JavaScript objects that the SDK supplies. Some of these objects are created when GoLive interprets a `Main.html` file, while others, such as the application object, are always available.

Global Properties and Functions

This section describes properties and methods available within the global JavaScript namespace.

Global Properties

Global properties provide access to the application object, the module object, the common object, and all currently-available dialogs, palettes, boxes, and menus.

Various global properties provide access to collections of dialogs, palettes, boxes, and menus. A collection is like an array that enables you retrieve objects by name as well as by integer index.

NOTE: When two or more objects in a collection have the same `name` value, those objects cannot be retrieved by name reliably.

app	*Application*	The application object. Read-only.
boxes	*BoxCollection*	A read-only array of all custom boxes running in the current document. You can retrieve boxes from this array by name or by numeric index.
common	*Object*	The common object allows modules to share and exchange data. All properties belonging to the common object are shared among all modules. Unlike the Prefs object, properties of the common object are not persistent. Properties of the common object can hold anything except objects, which are not permitted because objects cannot be shared among extension modules.
controls	*ControlCollection*	A read-only array of all controls that have run at least once in the current user session. You can retrieve controls from this array by name; in AGL 5 only, you can also use a numeric index to retrieve controls from this collection.
dialogs	*DialogCollection*	A read-only array of all dialogs that have run at least once in the current user session. You can retrieve dialogs from this array by name or by numeric index.

Global properties (*continued from preceding page*)

`document`	*Document*	The current document. Read-only
`documents`	*DocumentCollection*	A read-only array of all open documents. You can retrieve documents from this array by name (as reported by the `document.name` property) or by numeric index.
`DynamicContent`	*DynamicContent*	Provides access to Dynamic Content features. GoLive 6 only.
`menus`	*MenuCollection*	A read-only array of every menu and submenu defined by every SDK module, including the Special menu. You can retrieve menus from this array by name or by numeric index. Menu items are available as properties of the menus that provide them.
`module`	*Module*	The extension itself. Read-only
`pictures`	*PictureCollection*	A read-only array of picture objects created by this module. You can retrieve pictures from this array by name or by numeric index. Picture objects created by interpreting `<img>` elements in this module's `main.html` are available from this array immediately. A picture object created by the `createPicture` function is not added to this collection until the first time it draws itself.
`prefs`	*Prefs*	The Preferences object gives all extension modules access to all preference data it holds. All properties of this object are persistent and have the same value for every extension module.
`ServerInfo`	*ServerInfo*	The `ServerInfo` object enables Extend Script extensions to request data from server information files on a Dynamic Content server. Available only in GoLive 6.
`settings`	*Settings*	The Settings object provides access to GoLive settings. Available only in GoLive 6.
`website`	*Website*	The active site. The active site's document does not have to be the frontmost document. The active site is the site that also shows its objects (colors, font sets, site extras) in the various palettes and menus. If you assign a Website object to this property, GoLive makes that site's document frontmost.
`websites`	*WebsiteCollection*	A collection of Website objects. Each object in the collection represents a currently-open website.

Global Functions

Any JavaScript object or function can call the functions this section describes.

absoluteURL

`absoluteURL (`*relURL*`,` *baseURL*`,` *separator*`)`

Using the specified base URL, converts a relative URL to an absolute URL.

Availability

5.0, 6.0

Parameters

`relURL`	*String*	Relative URL to convert
`baseURL`	*String*	URL to which this method appends the `relURL` value.
`separator`	*String*	Optional. When this value is supplied, it specifies the separator character used in the URL this method constructs. When this value is not supplied, this method uses the forward slash (`/`) character as the separator in the new URL it constructs.

Returns

`String`

Description

Using the specified base URL, converts a relative URL to an absolute URL. Optionally, constructs the new URL using a specified separator character. If no separator is specified, this method uses the forward slash (`/`) separator character.

alert

`alert (`*text*`)`

Displays the specified string in a user alert box that provides an OK button.

Availability

5.0, 6.0

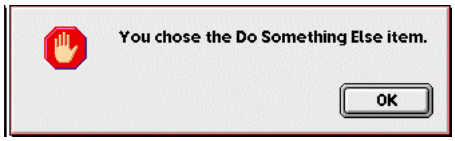
Parameter

`text`	*String*	A brief message to display to the user.

Description

This method displays a dialog similar to the one Figure 4.1 depicts.

FIGURE 4.1 Example of Alert Dialog



The alert dialog is not intended for lengthy messages. When the *string* argument to the `alert` method is too long, the `alert` dialog truncates it.

clearOutput

`clearOutput()`

Erases the contents of the **JavaScript Shell** window's output view.

Availability

5.0, 6.0

confirm

`confirm (`*text*`)`

Displays the specified string in a self-sizing modal dialog box that provides **Yes** (default) and **No** buttons.

Availability

5.0, 6.0

Parameter

text	*String*	Message to display to the user.

Returns

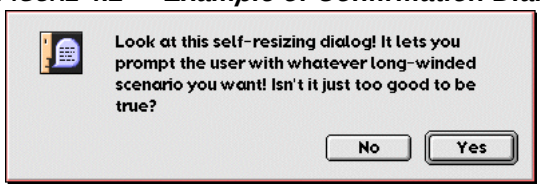
Boolean	`true`	The user clicked the **Yes** button to dismiss the dialog.
	`false`	The user clicked the **No** button to dismiss the dialog.

Description

This method displays a dialog similar to the one Figure 4.2 depicts. When this user clicks one of this dialog's buttons, this method hides the dialog and returns a value that indicates the button the user clicked to dismiss the dialog.

Although this dialog displays more text than the `alert` and `prompt` methods do, it still truncates *text* strings that are too long to fit in the dialog.

FIGURE 4.2 Example of Confirmation Dialog



createPicture

`createPicture (`*`url`*`)`

Reads the specified file to create a picture for use in an extension's drawing operation.

Availability

5.0, 6.0

Parameter

`url`	*String*	Relative or absolute URL of the file from which this method creates a picture.

Returns

Picture	The new picture object.

disposePicture

`disposePicture (`*`pic`*`)`

Deletes the `pic` picture created by the `createPicture` method.

Availability

5.0, 6.0

Parameter

`pic`	*Picture*	The Picture object to delete.

Description

When this method returns, the *pic* picture is no longer usable.

fileGetDialog

`fileGetDialog ([`*prompt*`, `*typeList*`])`

Presents the file-opening dialog that is standard for the platform on which GoLive is running.

Availability

5.0, 6.0

Parameters

prompt	*String*	Brief message to display in the file open dialog. If this string is too long, the dialog truncates it.
typeList	*String*	A platform-specific value indicating a list of file types to display.

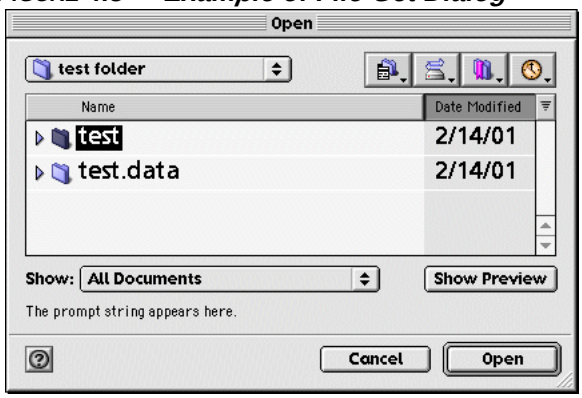
Returns

File Object	The user selected a file.
null	The user dismissed the dialog without selecting a file.

Description

This method displays a dialog similar to the one Figure 4.2 depicts. Optionally, the dialog can display a user prompt or it can display only specified file types.

FIGURE 4.3 Example of File Get Dialog



On Mac OS platforms, the *typeList* argument is a comma-separated list of four-character Mac OS file types; for example, passing `"TEXT,APPL"` as this argument specifies that the file open dialog is to display text files and applications only.

On Windows platforms, the *typeList* argument is a semicolon-separated list of search masks. For example, the value `"*.jpg;*.jpeg,*.html"` specifies that the file open dialog is to show only files having names that end with one of the `.jpg`, `.jpeg`, or `.html` extensions.

filePutDialog

`filePutDialog (`*prompt*`, `*default*`[, `*typeList*`])`

Presents the file-saving dialog that is standard for the platform on which GoLive is running.

Availability

5.0, 6.0

Parameters

`prompt`	*String*	Brief message to display in the file-saving dialog. If this string is too long, the dialog truncates it.
`default`	*String*	Default file name to display in the file-saving dialog. This value must observe the file-naming conventions of the platform on which GoLive is running.
`typeList`	*String*	Optional. A platform-specific value indicating a list of file types to display.

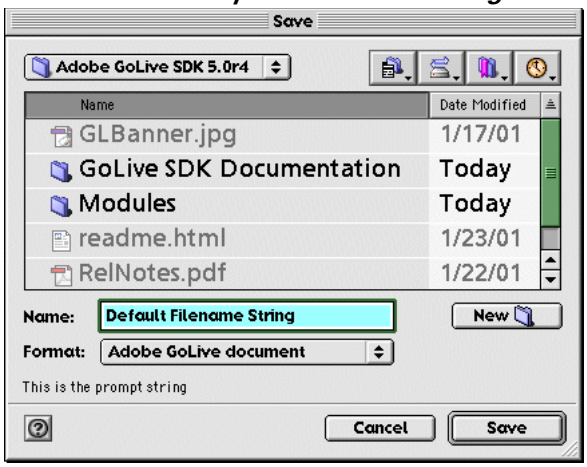
Returns

File Object	File object initialized to the path the user specified in the dialog.
null	The user dismissed the dialog without saving the file.

Description

This method displays a dialog similar to the one Figure 4.4 depicts. The `prompt` message and the `default` file name to present to the user are required.

FIGURE 4.4 Example of File Put Dialog



On Mac OS platforms, the optional *typeList* argument is a comma-separated list of four-character Mac OS file types; for example, passing `"TEXT,APPL"` as this argument specifies that the file open dialog is to display text files and applications only. On Windows platforms, the

optional *typeList* argument is a semicolon-separated list of search masks. For example, passing `"*.jpg;*.jpeg,*.html"` as this argument specifies that the file open dialog is to show only files having names that end with one of the `.jpg`, `.jpeg`, or `.html` extensions.

NOTE: In GoLive 5, the *typeList* argument is meaningful on Windows platforms only.

folderGetDialog

`folderGetDialog([`*prompt*`])`

Displays a dialog in which the user can select a folder.

Availability

5.0, 6.0

Parameter

`prompt`	*String*	Optional. Brief message to display to Windows users. The SDK ignores this argument when running on Mac OS platforms.

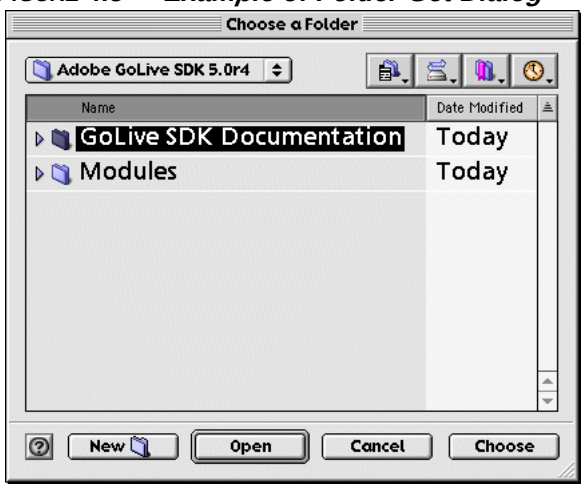
Returns

File Object representing the selected folder or `null` if the user cancelled.

Description

Displays a dialog similar to the one Figure 4.5 depicts. Optionally, this dialog displays a user prompt string on Windows platforms only.

FIGURE 4.5 Example of Folder Get Dialog



prompt

`prompt (`*prompt*`,` *default*`)`

Displays a modal dialog that returns the user's text input.

Availability

5.0, 6.0

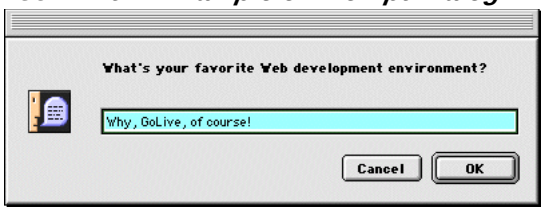
Returns

String	Text the user entered in the input field before clicking the OK button.
`undefined`	The user clicked the Cancel button.

Description

Displays a modal dialog like the one Figure 4.6 depicts. When the dialog opens, the editable text field displays the specified *default* text. When the user clicks OK to dismiss the dialog, this method returns the text the user entered. If the user clicks the Cancel button in this dialog, this method's result is undefined.

FIGURE 4.6 Example of Prompt Dialog



registerCallback

`registerCallback (`*evtName*`,` *fnName*`)`

Causes GoLive to call the *fnName*() callback function when the *evtName* event occurs.

Availability

6.0

Parameters

evtName	*Markup*	Name of event that is to cause a call to the *fnName* function.
fnName	*Markup*	Name of function to execute in response to the *evtName* event.

Returns

Boolean	`true` indicates success.

Description

Your extension implements the *fnToCall* function as one that accepts a single argument. GoLive supplies this value when it calls your function.

```
function fnToCall (data) { // respond to evtName event ... }
```

Example

This example causes GoLive to call the `onNow()` custom callback function when the `Now!` event occurs.

```
function onNow (data)
{
     writeln ("I've been called with " + data * " as data!");
}
function initializeModule()
{
     registerCallback ("Now!", "onNow");
}
```

relativeURL

`relativeURL (`*absURL*`, `*baseURL*`, `*separator*`)`

Converts an absolute URL to a URL that is relative to the specified base URL.

Availability

5.0, 6.0

Parameters

`absURL`	*String*	Absolute URL to convert
`baseURL`	*String*	URL used as the working directory value.
`separator`	*String*	Optional. When this value is supplied, it specifies the separator character used in the URL this method constructs. When this value is not supplied, this method uses the forward slash (`/`) character as the separator in the new URL it constructs.

Returns

`String`

Description

Optionally, constructs the new relative URL using a specified separator character. If no separator is specified, this method uses the forward slash (`/`) separator character.

startTimer

`startTimer (`*scriptlet*`, `*timeout*` [`, *repeat*`])`

Executes the *scriptlet* JavaScript code after *timeout* seconds have elapsed.

Availability

5.0, 6.0

Parameters

`scriptlet`	*String*	JavaScript expression to execute.
`timeout`	*Number*	The number of milliseconds that must elapse before the SDK executes the scriptlet.
`repeat`	*Boolean*	Optional. Pass `true` to execute the scriptlet repeatedly.

Returns

Number	Pass this value to the `stopTimer` method to stop the repeated execution of this scriptlet.

stopTimer

`stopTimer (`*id*`)`

Stops the delayed or repeated execution of the specified scriptlet.

Availability

5.0, 6.0

Parameter

`id`	*Number*	Value returned by the startTimer method when the execution of this scriptlet was scheduled.

write

`write (`*text*`)`

Writes its arguments to the output view of the **JavaScript Shell** palette.

Availability

5.0, 6.0

writeln

`writeln (`*text*`)`

Writes its arguments and a Linefeed character to the output view of the **JavaScript Shell** palette.

Availability

5.0, 6.0

---

app Object

Availability

5.0, 6.0

The Application object enables programmatic access to data and functionality that the GoLive application provides. Properties of the Application object provide access to information about the environment in which the extension is running. Methods of the application object can create documents, open existing documents, display progress bars, broadcast a message string to all running extensions, and quit the GoLive application. When the GoLive application quits, it closes all open documents, prompting the user to save or discard each document's changes as necessary, and creating disk files as necessary.

Acquiring the Application Object

The app global variable makes this object available.

```
app
```

Application Object Properties

Properties of the Application object provide access to information about the environment in which the extension is running, such as

- Current HTML parser preferences.
- Information about the currently-running GoLive application, such as its version and whether it is currently displaying a modal dialog.
- Information about the currently-running Extend Script SDK, such as its version.
- Paths to commonly-used folders such as the current site's folder, the system trash, the folder that holds the GoLive application, the default HTML browser application, and so on.
- Shared data and user preference data provided by extensions

Application object properties (*continued from preceding page*)

`asymmetricTokens`	*String*	This property is a string containing all delimiters which the GoLive scanner uses together with the < character to recognize asymmetric tags. By default, the string is set to „!@“, which means that SGML <!tags> and <@tags> are recognized. This value reflects the setting inside the Web database. The value of this property is not persistent unless the Web database is saved manually.
`currentFolder`	*File*	A *File* object referring to the current working directory. The **current working directory** is the folder in which GoLive executes operations that do not specify another path. Normally, this directory is the root folder of the currently-active site, but the user or JavaScript code can change this default value. May be assigned a path name or a *File* object describing a folder.
`defaultBrowser`	*File*	A *File* object referring to the current default HTML browser application. The value of this property is null when no default browser preference is specified. Read-only.
`encodingAlert`	*Boolean*	`true` when encoding alerts are enabled. To disable these alerts temporarily, set this property to `false`, restoring its previous value once the problematic document is opened. For more information, see “Disabling Encoding Alerts,” later in this section.
`folder`	*File*	A *File* object referring to the folder that holds the GoLive application. Read-only.
`htmlStyles`	*HTMLStyleSetCollection*	All currently available HTML styles. These styles appear in the **Window>HTML Styles** palette.
`isModal`	*Boolean*	`true` when GoLive is displaying a modal dialog. Read-only.
`osVersion`	*String*	The version of the operating system. Read-only.
`prefs`	*GlobalPrefs*	This object provides limited access to GoLive's preferences. Read-only
`scanBrackets`	*Boolean*	This property controls whether the GoLive scanner recognizes bracketed tags like `[hello]`. This value reflects the setting inside the Web database. The value of this property is not persistent unless the Web database is saved manually.

Application object properties (*continued from preceding page*)

`sdkVersion`	*number*	The version of the currently-running Extend Script SDK. Read-only. This floating-point number provides the version numbers of the GoLive application and the **ExtendScript** module when you interpret the decimal point as a separator character. The portion of this value appearing to the left of the decimal point (the integer part) is a two-digit representation of the currently running GoLive application's version number; currently, this value is 60, representing version 6.0 of the GoLive application. The portion of this value appearing to the right of the decimal point (the fractional part) is a three-digit representation of the ExtendScript module version. Currently, this value is 001, representing release 1 of the **ExtendScript** module. Thus, the nominal value of the `sdkVersion` property is `60.001` in the current release.

NOTE: The SDK release number does not necessarily match the module's version number. For example, SDK Release 4.1 supplies version 004 of the **ExtendScript** module. For details of the configuration of a particular SDK release, see the Release Notes and Readme file that accompany the distribution. For SDK updates and new sample extensions as they become available, visit `http://partners.adobe.com/asn/developer/gapsdk/GoLiveSDK.html`

`settingsFolder`	*File*	A *File* object referring to the `Modules` folder that holds GoLive extension modules and other settings. Read-only.
`symmetricTokens`	*String*	This property is a string containing all delimiters which the GoLive scanner uses together with the `<` character to recognize symmetric tags. By default, the string is set to `?%`, which means that `<%tags%>` and `<?tags?>` are recognized. It reflects the setting inside the Web database. The value of this property is not persistent unless the Web database is saved manually.
`systemFolder`	*File*	A *File* object referring to the folder that holds the operating system. Usually, this is the `C:\WINDOWS` or `C:\WINNT` folder on Windows platforms; on Mac OS platforms, this folder can have any name and it can reside on any local disk.

Application object properties (*continued from preceding page*)

`tableStyles`	*TableStyleCollection*	Table style list maintained by the GoLive application. These styles appear in the **Style** tab of the **Window**>**Table** palette.
`tempFolder`	*File*	A *File* object that represents the folder GoLive uses to store temporary files. Read-only.
`trashFolder`	*File*	A *File* object referring to the system Trash folder. On Windows platforms, this object refers to the local `\RECYCLED` directory. On Mac OS platforms, this object refers to the *startupDisk*`:Trash` folder. Read-only.
`version`	*String*	The version of the currently-running GoLive application. Read only.

Disabling Encoding Alerts

When GoLive is configured to retrieve HTML document encoding information from the document's `<META>` element, it displays a dialog when it opens an HTML document that:

- Has no `<META>` element
- Contains an invalid `<META>` element
- Contains a `<META>` element that specifies an unsupported character set.

To disable these alerts temporarily, set the Application object's `encodingAlerts` property to `false`. Once the problematic document is opened, restored this property to its previous value.

For additional information, see "Using a Specific Encoding to Read a Document" on page 44

Application Object Functions

Functions of the application object provide a programmatic means of performing application-level tasks, such as passing keycodes and action codes to GoLive, creating new documents or opening existing ones, opening a URL in the current browser, accessing user preferences and configuration information, broadcasting string data to all Extend Script extensions, or quitting GoLive.

Using a Specific Encoding to Read a Document

In GoLive 6, you can specify the document encoding that the `openDocument` and `openMarkup` methods use to open a document that has no `<meta>` tag. For more information, see the individual listings for the `app.openDocument`, `app.openMarkup`, and *FileObj*`.openMarkup` methods.

broadcast

`app.broadcast(`*argument* `[,` *targets*`])`

Calls the `broadcastSignal` function of other extension modules.

Availability

5.0, 6.0

Parameters

`argument`	*String*	Argument string this method passes to the `broadcastSignal` function. If you pass an object as this argument, it is converted to a string, and the string value is passed. Objects cannot be shared among modules.
`targets`	*String*	Optional. Comma-separated list of names of modules to call, in the order they are to be called. When this argument is supplied, this method calls only the modules this list names. When this argument is not supplied, this method calls all modules.

Returns

This method's result is the return value supplied by any called module's `broadcastSignal` function. If none of the called modules supply a return value, the `broadcast` method's return value is undefined.

Description

This method sends an extension-specific message to the specified extension modules and returns the response as its result; to do so, this method

- Converts the `argument` value to a string.
- Passes the name of the calling module and the converted `argument` string to the `broadcastSignal` function of another module.

  The optional `targets` parameter specifies a list of modules to call in the order they are to be called; when this argument is not supplied, the `broadcast` method calls every extension module's `broadcastSignal` function.

Each called module's `broadcastSignal` function returns a string value or `undefined`. Broadcasting terminates when any module's `broadcastSignal` method returns.

A called module's `broadcastSignal` function can set the `broadcast` method's return value to `undefined` by returning the `undefined` string, as the following example does.

```
function broadcastSignal(...) {
     var result = 0;
     if (...)
           result = 5;
     else
           result = undefined;
     return result;
}
```

createProgressLog

`app.createProgressLog()`

Creates a new ProgressLog object.

Availability

6.0

Returns

ProgressLog	New ProgressLog object.

Description

Before attempting to add messages to the progress log this method returns, call *progressLogObj*`.begin()` once.

isModulePresent

`app.isModulePresent(`*name*`)`

Returns `true` if the specified GoLive module is enabled.

Availability

5.0, 6.0

Parameter

name	*String*	`Name` property of the module to test.

Returns

Boolean

Description

This method does not test for the presence of Extend Script modules. This method always returns `false` when passed the name of an Extend Script module.

NOTE: Deprecated. Use the `isOn` method of the settings.aglmodules Object instead of this method. This method is available for GoLive 5.0 compatibility purposes only, and may not be supported at some future date.

launchURL

`app.launchURL (`*url*`)`

Opens the specified URL in the current browser.

Availability

5.0, 6.0

Parameter

`url`	*String*	URL to open.

Returns

Boolean

Description

Opens the specified URL in the current browser. If the current browser is not defined, this method does nothing and returns `false`.

newDocument

`app.newDocument()`

Opens a new, empty document and returns a value indicating success or failure.

Availability

5.0, 6.0

Returns

Document	The newly-created document, which has been added to the `documents` array.
`null`	Document creation failed.

Description

Opens a new, empty document and returns a value indicating success or failure. Same as choosing the **File>New** menu item.

openDocument

`app.openDocument (`*pathOrURL* `[,` *encoding*`])`

Open the specified document.

Availability

5.0, 6.0

Parameters

pathOrURL	*String*	Full or partial path to the file to open, specified as a URL or platform-specific path name. If you do not specify this value as a URL, this value must follow the file system conventions of the platform on which GoLive is currently running.
encoding (6.0 only)	*String*	Optional. The name of an encoding character set. This character set overrides the default encoding that GoLive would use if the document did not contain a META tag. A META tag causes the supplied encoding to be ignored. When this argument specifies an unknown or unsupported encoding, GoLive generates an "Unsupported encoding" runtime error.

Returns

Document	The newly-opened document, which has been added to the `documents` array.
`null`	Document creation failed.

Description

Open the specified document. If the `docName` argument is not supplied, this method displays the file-open dialog that is standard for the platform on which GoLive is running.

NOTE: In GoLive 5, this method displays an alert when the file being opened does not provide a META tag defining the encoding of the document. In GoLive 5, the `app.encodingAlert` property can be used to suppress this alert. In GoLive 6, the `openDocument` method does not display encoding alerts. Use of the `encodingAlert` property is deprecated in GoLive 6.

openMarkup

`app.openMarkup (`*pathOrURL* `[,` *encoding*`])`

Opens a document without displaying it in a window.

Availability

5.0, 6.0

Parameters

pathOrURL	*String*	Full or partial path to the document to open, specified as a URL or platform-specific path name. If you do not specify this value as a URL, this value must follow the file system conventions of the platform on which GoLive is currently running.
encoding (6.0 only)	*String*	Optional. The name of an encoding character set. This character set overrides the default encoding that GoLive would use if the document did not contain a META tag. A META tag causes the supplied encoding to be ignored. If the encoding name is unknown or unsupported, GoLive generates an “Unsupported encoding” runtime error.

Returns

Document	The newly-created document, which has been added to the `documents` array.
`null`	Document creation failed.

Description

Use this method to work with a document’s markup tree without displaying the document in Layout view. Not displaying a document in Layout view

- Saves time and memory.

  To batch-process documents quickly, open them with this method rather than the openDocument() method.

- Avoids reparses.

  GoLive does not reparse when Layout view is not open.

openPrefs

`app.openPrefs()`

Opens the Preferences panel.

Availability

5.0, 6.0

Description

Same as choosing the **Edit>Preferences** menu item.

postKey

`app.postKey(`*key*`)`

Posts a keycode to the GoLive application's input queue.

Availability

5.0, 6.0

Parameter

key	*String*	The ASCII value of the key to post or a string that begins with the character to post.

quit

`app.quit()`

Terminates the GoLive application by posting a quit message to the operating system.

Availability

5.0, 6.0

Description

Terminates the GoLive application by posting a quit message to the operating system. Same as choosing the **File>Quit** menu item.

sendLMScript

`app.sendLMScript(`*lmscript* `)`

Sends the *lmscript* JavaScript code to Adobe LiveMotion" .

Parameter

lmscript	*String*	The JavaScript code to send to LiveMotion.

Description

Equivalent to running the *lmscript* Automation Script manually from within LiveMotion. For more information on scripting LiveMotion, see the Adobe LiveMotion user guide.

setProgress

`app.setProgress(`*value* `[,` *message*`])`

While the progress bar or busy bar is displayed, call this method regularly to update the progress user interface and to determine whether the user has clicked the Stop button.

Availability

5.0, 6.0

Parameter

`value`	*Number*	A value between 0 and 1 that sets the progress bar's position. Busy bars ignore this parameter. A value of 1 specifies that 100% of the bar is to be drawn, and decimal values less than 1 equate to a corresponding percentage of the progress bar.
`message`	*String*	Optional. Status message to display in the progress dialog. Replaces the current message.

Returns

Boolean	`false` indicates that the user clicked the progress dialog's **Stop** button.

Description

While the progress bar or busy bar is displayed, call this method regularly to update the progress user interface and to determine whether the user has clicked the Stop button.

startProgress

`app.startProgress(`*title* `[,` *message*`,` *doBusy*`,` *seconds*`])`

Displays a progress dialog with progress bar or busy bar.

Availability

5.0, 6.0

Parameters

`title`	*String*	String that appears in the progress dialog's title bar.
`message`	*String*	Optional. Status message to display in the progress dialog. Replaces the current message.
`doBusy`	*Boolean*	Optional. Set to `true` to display a busy bar instead of a progress bar.
`seconds`	*Number*	Optional. Number of seconds to wait before displaying progress dialog.

Description

Displays a progress dialog with progress bar or busy bar. All parameters except `title` are optional.

stopProgress

```
app.stopProgress()
```

Closes the progress window.

Availability

5.0, 6.0

---

app.htmlStyles Object

Availability

6.0

This object provides access to all named stylesets in the **Window>Styles** palette. Other stylesets, such as those read from a document, do not appear in this array unless added to it by this object's `addStyle` method.

Acquiring the app.htmlStyles Object

The `htmlStyles` property of the application object provides a collection of `HTMLStyleSet` objects that appear in the **Window>Styles** palette.

```
app.htmlStyles
```

app.htmlStyles Object Properties

`name`	*String*	Styleset name.
`paragraph`	*Boolean*	true if this styleset used at the total paragraph.
`replace`	*Boolean*	true if the style replaces the current style. (Paragraph styles replace existing paragraph styles and inline styles replace existing inline styles.)
`style`	*HTMLStyleSet*	Array of HTMLStyle objects. Read-only.

app.htmlStyles Object Functions

HTMLStyleSetObj.addStyle (*style*)
HTMLStyleSetObj.addStyle (*elementname*, *attributename*, *attributevalue*)
HTMLStyleSetObj.removeStyle (*indexNumber*)
HTMLStyleSetObj.removeStyle (*HTMLStyleObject*)
HTMLStyleSetObj.removeStyle (*elementname*, *attributename*, *attributevalue*)
HTMLStyleSetObj.setStyle (*indexNumber*)
HTMLStyleSetObj.setStyle (*newStyle*)
HTMLStyleSetObj.setStyle (*newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle (*newEltName*, *newAttrName*, *newAttrVal*, *newStyle*)
HTMLStyleSetObj.setStyle (*indexNumber*, *newStyle*)
HTMLStyleSetObj.setStyle (*indexNumber*, *newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle (*oldStyle*, *newStyle*)
HTMLStyleSetObj.setStyle (*oldStyle*, *newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle (*newEltName*, *newAttrName*, *newAttrVal*, *newStyle*)
HTMLStyleSetObj.setStyle (*oldEltName*, *oldAttrName*, *oldAttrVal*,
newEltName, *newAttrName*, *newAttrVal*)

addStyle

HTMLStyleSetObj.addStyle(*style*)

HTMLStyleSetObj.addStyle(*elementname*, *attributename*, *attributevalue*)

Add the specified style to the styleset.

Availability

6.0

Parameters

style	*HTMLStyle*	Style to add.
elementname	*String*	Name of element that gets the new style.
attributename	*String*	Name of style attribute to add.
attributevalue	*String*	Value of style attribute to add.

Returns

Boolean	`true` indicates success.

Description

You can describe the style to add by passing an existing **HTMLStyle** object or by passing string arguments that describe it.

removeStyle

HTMLStyleSetObj`.removeStyle(`*indexNumber*`)`

HTMLStyleSetObj`.removeStyle(`*HTMLStyleObject*`)`

HTMLStyleSetObj`.removeStyle(`*elementname*`,` *attributename*`,` *attributevalue*`)`

Remove the specified style from the called `HTMLStyleSet` object.

Availability

6.0

Parameters

indexNumber	*Number*	Index number of style to remove. Index number 0 is the first style in the collection.
HTMLStyleObject	*HTMLStyle*	A HTMLStyle object that is the same as the one to remove.
elementname	*String*	Name of element from which to remove the style.
attributename	*String*	Name of style attribute to remove.
attributevalue	*String*	Value of style attribute to remove.

Returns

Boolean	`true` indicates success.

Description

You can specify the styleset to remove by numeric index into the styleset; by passing an existing **HTMLStyle** object; or by passing string arguments that describe it.

setStyle

HTMLStyleSetObj.setStyle(*indexNumber*)

HTMLStyleSetObj.setStyle(*newStyle*)

HTMLStyleSetObj.setStyle(*newEltName*, *newAttrName*, *newAttrVal*)

HTMLStyleSetObj.setStyle(*newEltName*, *newAttrName*, *newAttrVal*, *newStyle*)

HTMLStyleSetObj.setStyle(*indexNumber*, *newStyle*)

HTMLStyleSetObj.setStyle(*indexNumber*, *newEltName*, *newAttrName*, *newAttrVal*)

HTMLStyleSetObj.setStyle(*replaceStyle*, *newStyle*)

HTMLStyleSetObj.setStyle(*replaceStyle*, *newEltName*, *newAttrName*, *newAttrVal*)

HTMLStyleSetObj.setStyle(*newEltName*, *newAttrName*, *newAttrVal*, *newStyle*)

HTMLStyleSetObj.setStyle(*replaceEltName*, *replaceAttrName*, *replaceAttrVal*, *newEltName*, *newAttrName*, *newAttrVal*)

Set attributes of the current style or those of a specified *HTMLStyleSetObj* object.

Availability

6.0

Parameters

indexNumber	*Number*	Index number of style to set. Index number 0 is the first style in the collection.
newStyle	*HTMLStyle*	The HTMLStyle object that provides the style to set.
replaceStyle	*HTMLStyle*	The HTMLStyle object to replace.
newEltName	*String*	Name of style element to use as current style.
newAttrName	*String*	Name of style attribute to use in current style.
replaceAttrName	*String*	Name of style attribute to replace.
newAttrVal	*String*	Value of style attribute to use in current style.
replaceAttrVal	*String*	Value of style attribute to replace.

Returns

Boolean	`true` indicates success.

app.prefs Object

Availability

5.0, 6.0

The `app.prefs` object enables read-only access to a subset of the GoLive global preferences that the **Edit>Preferences** panel makes available to the user.

Acquiring the app.prefs Object

```
app.prefs
```

Properties of the app.prefs Object

`absoluteURLs`	*Boolean*	**Edit>Preferences>General>URL Handling>Make new links absolute** check box.
`imageFolder`	*String*	**Edit>Preferences>General>Images>Picture Import>Import folder** input field.
`scriptLibFolder`	*String*	The name of the Script Library folder that holds `.js` files. In GoLive 5.0, the default Script Library folder is the site's `GeneratedItems` folder. The user can change this default value in the **Edit>Preferences>Script Library>Folder for Script Library** input field. This property is undefined when the **Smart Objects** module is disabled.
`scriptLibName`	*String*	The name of the file that the **Import GoLive Script Library** radio button imports. The user can change this default value in the **Edit>Preferences>Script Library>Name of Script Library** input field.This property is undefined when the **Smart Objects** module is disabled.
`writeGenerator`	*Boolean*	**Edit>Preferences>General>Write "Generator Adobe GoLive"** checkbox.

Extension modules can use a prefs Object to store persistent user preference data that is available to all extensions; for more information, see "prefs Object" on page 164.

app.tableStyles Object

Availability

6.0

This object provides access to all named table styles in the **Window>Table** palette. Other table styles, such as those read from a document, do not appear in this array unless added to it by this object's `addStyle` method.

Acquiring the app.htmlStyles Object

The `tableStyles` property of the application object provides a collection of `HTMLTableStyle` objects that appear in the **Window>Table** palette.

```
app.tableStyles
```

app.tableStyles Object Properties

This object's index operator retrieves table styles by name or by numeric index.

[*name*]	*String*	JavaScript name of the table style to retrieve.
[*num*]	*Number*	Numeric index of the table style to retrieve.

Box Object

Availability

5.0, 6.0

A box object manages the visual representation of a custom element in **Layout** view.

Acquiring Box Objects

GoLive passes box objects to event-handling functions that use them.

```
parseBox(box, reason)
drawBox(box, draw)
boxResized(box, width, height)
inspectBox(box)
```

GoLive creates a box object whenever the user drags a custom element's icon from the Objects palette to the Layout view of a GoLive document window, or when GoLive reads a document that contains a custom tag. GoLive does not create box objects when interpreting documents opened by the `openMarkup` method.

Each box object represents an instance of a custom markup element defined with the <jsxelement> and <jsxpaletteentry> tags. The <jsxelement> element defines a tag name, such as <myTag>, while the <jsxpaletteentry> element defines the tag's attributes and the HTML content it adds to the page. The <jsxinspector> tag creates an inspector window that interacts with the box object to get and set properties of the markup object that represents the custom element in the GoLive document.

Each of these tags has a `classid` attribute that associates the components of a particular custom element. This value identifies the palette entry that creates the box object and inserts HTML in the document, as well as the inspector window to display when the box is activated. This value can be any unique string value; however, the <jsxelement>, <jsxpaletteentry>, and <jsxinspector> elements that compose a particular custom element must specify the same `classid` attribute value. An individual `classid` value must be unique among all currently-running extensions, and must be used by one extension only.

The `parseBox`, `drawBox` and `boxResized` event-handling functions implement the box object's basic functionality:

- When the user drags a palette entry's icon from the **Objects** palette to **Layout** view, GoLive
  - adds the palette entry's predefined content to the document's markup tree.
  - adds to Layout view a box object that represents the palette entry in this view.

  GoLive then calls the extension's `parseBox` function, passing this box object as its argument. Your implementation of the `parseBox` function completes the initialization of the box object passed to it. It sets the `height` and `width` properties of the passed box according to values it retrieves from the `height` and `width` properties of the markup element the box represents. It's also common for this method to save previous `height` and `width` values for use by Undo operations.
- Once the box has been initialized, GoLive passes it to the extension's `drawBox` method to draw the palette entry's placeholder in Layout view. This method sets the height and width of the placeholder graphic to the height and width of the box passed as one of its arguments, then it draws the placeholder graphic.
- When the box resizes (because the user dragged the handles at its borders or entered data into its inspector window), GoLive calls the extension's `boxResized` method, passing as its arguments the new width, the new height, and the box object itself. Your implementation of this method must update the `height` and `width` properties of the box's associated markup element appropriately.

Multiple Inspectors

In GoLive 6, a Box object can have multiple inspectors. To create more than one inspector, define additional inspectors with alternate *classid* attributes, as the following example does:

```
<jsxinspector name="insp1" title="one" classid="myclass">
…
</jsxinspector>
<jsxinspector name="insp2" title="two" classid="alternate">
…
</jsxinspector>
```

At runtime, you can specify the inspector a box is to use by setting its classid property; for example, the following line of code specifies that the `myBox` object uses the inspector that has a classid property that returns the `alternate` value.

```
var myBox = boxes[0]; myBox.classid = "alternate";
```

Box Object Properties

Most of the box object's properties describe its visual representation. You can set these values to change the box's appearance in Layout view. To update the affected HTML, call methods of the markup object that the `box.element` property provides.

`bottomMargin`	*Number*	Number of pixels by which the visible representation of the content of a container box is inset from the bottom of the box. Used only by container boxes, such as those which represent binary tags.
`classid`	*String*	The identifier that associates this box with an inspector and a palette entry. This value is specified by the `classid` attributes of the `<jsxelement>`, `<jsxpaletteentry>` and `<jsxinspector>` elements associated with this box. Read-only in GoLive 5; in GoLive 6, set this value to activate an alternate inspector for this box.
`document`	*Document*	The document that contains this box.
`element`	*Markup*	The markup element associated with this box. Read-only.
`height`	*Number*	The height of the box, expressed as a number of pixels.
`inspector`	*Dialog*	A reference to the inspector dialog for this box. This property is set only when the box is selected and an inspector dialog is active for that box; otherwise, this property is `null`. Read-only.
`leftMargin`	*Number*	Number of pixels by which the visible representation of the content of a container box is inset from the left side of the box. Used only by container boxes, such as those which represent binary tags.
`links`	*LinkCollection*	Links to all files that reference this box (in links) and all files this box references (out links).
`name`	*String*	The JavaScript name of the box. GoLive assigns this value when it creates the box object and you cannot change it. You can use this value to retrieve the box by name from the collection in the boxes global variable. Read-only.
`rightMargin`	*Number*	Number of pixels by which the visible representation of the content of a container box is inset from the right side of the box. Used only by container boxes, such as those which represent binary tags.

Box object properties (*continued from preceding page*)

`topMargin`	*Number*	Number of pixels by which the visible representation of the content of a container box is inset from the top of the box. Used only by container boxes, such as those which represent binary tags.
`width`	*Number*	The width of the box, expressed as a number of pixels.
`x`	*Number*	The x-coordinate of the top-left corner of the box in relation to an origin at the top left corner of the document. Because this position is determined dynamically when GoLive interprets the document layout, this property is read-only.
`y`	*Number*	The y-coordinate of the top-left corner of the box in relation to an origin at the top left corner of the document. Because this position is determined dynamically when GoLive interprets the document layout, this property is read-only.

Box Object Functions

The box object itself provides only a few convenience functions. A box object's most important behaviors are provided by global event-handling functions that GoLive calls when the user interacts with the box.

NOTE: To use a box object, your extension must provide the global event-handling methods that the Boxes section of Chapter 6, "Event-Handling Functions," describes. These event-handling functions handle the vast majority of user interactions with the box and its inspector.

This section describes a few convenience functions the box object provides.

- When you do need to call the box object yourself, it's usually because you want to tell it to redraw itself; for example, it might represent a custom control that you're updating yourself. For such situations, the box object provides the `refresh` method.
- Inspector windows use the `createLink` and `removeLink` functions to create or remove links in the box object in response to user interaction with URL getter controls.

createLink

boxObj.createLink (*url*)

Creates a new link and attaches it to the box.

Availability

5.0, 6.0

Parameter

url	*String*	URL to create. If no value supplied, this method creates an invalid link which can be set to a valid URL at another time.

Returns

Link

refresh

boxObj.refresh()

Requests a repaint of the box.

Availability

5.0, 6.0

removeLink

boxObj.removeLink (*link*)

Removes the specified link from the box.

Availability

5.0, 6.0

Parameter

link	*Link*	Link object to remove.

Description

Removes the specified link from the box. The link object is invalid when this call returns.

Collection Object

Availability

5.0, 6.0

A collection object acts like an array that provides access to its elements by name. Like an array, a collection associates a set of objects or values as a logical group and provides random access to them. However, it is important to note that Collection objects are not arrays:

- You do not create Collection objects yourself. The SDK creates collections when the GoLive application opens.
- Most collection objects are read-only. You do not assign objects to them yourself—the SDK updates their contents automatically as objects are created or deleted.
- An Array places no restrictions on the type of data its elements store. A collection object always holds the type of data indicated by its object name: a `BoxCollection` object holds box objects; a `ControlCollection` object holds control objects; a `DocumentCollection` object holds document objects; and so on.
- Not all collections can be accessed by numeric index, as an array is.

```
var myDocs= new Array ();
myDocs[0] = document;
var aMember = myDocs [0]; //some Collections don't allow this
```

Acquiring Collection Objects

Table 4.1 describes the various collections the SDK provides, as well as how to access each of these objects in JavaScript.

TABLE 4.1 Collection Object Access

Object	JavaScript Access	Indexes		Contents
		Number	Name	
BoxCollection	`boxes` global variable	√	√	Read-only array of all boxes in the current document.
ControlCollection	`controls` global variable	AGL5 only	√	The current document's controls that have run at least once.
DialogCollection	`dialogs` global variable	√	√	The current document's dialogs that have run at least once. Read-only.
DocumentCollection	`documents` global variable	√	√	Documents open in GoLive.
History	`document.history` property	√	√	Undo actions involving `document`

Object	JavaScript Access	Indexes		Contents
		Number	Name	
HTMLStyleSetCollection	`app.htmlStyles` property	√	√	Every HTMLStyleSet Object in the **Window>Styles** palette. AGL6 only.
LinkCollection	*boxObj*`.links` property	√	√	Links to all files that reference this box (in links) and all files this box references (out links).
MarkupCollection	*markupObj*`.subElements` `document.element.subElements`	√	√	Immediate subelements of the *markup* object.
MenuCollection	`menus` global variable	√	√	All menus currently available in GoLive.
MenuItemCollection	`menus[`*value*`].items` property *menuObj*`.items.`*propertyName*	√	√	All the menu items belonging to the `menus[`*value*`]` menu object.
ModulesCollection	`modules` global variable	√	√	All currently-running GoLive Extend Script extensions.
PictureCollection	`pictures` global variable	√	√	All pictures accessible to this module.
WebsiteCollection	`websites` global variable	√	√	All websites currently open in GoLive.

Collection Object Properties

The properties of a Collection object provide a count of its elements and an index operator.

`length` *Number*

The number of elements of the collection. This value is zero if the members cannot be indexed by number.

NOTE: In GoLive 6, `ControlCollection` objects do not provide this property and cannot be indexed numerically.

`[index]` *Object*

Most collections provide name-based and numeric indexes of their elements; for information on a specific collection, see Table 4.1, in this section.

common Object

Availability

5.0, 6.0

The common object provides a means of sharing non-persistent string and primitive data among all currently-running Extend Script extensions. It is intended for sharing preference data and the like. You cannot store objects in the Common object.

Acquiring the Common Object

The common global property makes this object available to all **ExtendScript** extensions.

```
common
```

Using the Common Object

The common object provides only one method. The `create` method creates a new namespace in the common object. This namespace is itself another common object that the SDK makes available as a property of the Common object the global `common` property holds.

Call the `create` method once to create a namespace in the common object. This namespace holds all of your extension's shared data. When this method returns, the namespace is available as a property of the Common object.

The JavaScript name of the namespace is the string passed as the argument to the `create` method. For example, the **Kewl** extension could use the following line of JavaScript to create a `kewl` property in the common object.

```
var myCommon = common.create("kewl");
```

Store your shared data as properties of your namespace in the Common object. To create a property, simply declare it and assign a value to it; for example, the next line of code creates the `myProperty` property in the `kewl` namespace of the Common object, and assigns to it the `myValue` value.

```
common.kewl.myProperty = myValue;
```

For your convenience, the `create` method returns a reference to the namespace it creates in the Common object. You can assign properties to this object just as if it were the object the global `common` property provides, although it is not. Thus the following lines of code are equivalent.

```
myCommon.myProperty = myValue;
common.kewl.myProperty = myValue;
```

You cannot assign any kind of object to a property of the common object. The common object holds strings and primitives only. Thus, you can add properties to a Common object, but you cannot add properties to the properties of a Common object. To create properties like `customer.first` and `customer.last` in your namespace, you have to call the `create` method again to create the `customer` namespace, as the following example does.

```
var myCommon = common.create("kewl");
myCommon.create("customer");
myCommon.customer.first = "John";
myCommon.customer.last = "Perry";
writeln(common.kewl.customer.first);
writeln(common.kewl.customer.last);
```

Common Object Properties

The Common Object provides no properties of its own. An extension calls the Common Object's `create` method to create its shared namespace in the Common object, and then assigns properties to this namespace as necessary to hold its shared data.

Common Object Functions

The Common object provides only one method of its own—the `create` method creates a new namespace in the common object.

create

`common.create (`*nameSpace*`)`

Creates a new `nameSpace` property in the object that the `common` global property provides.

Availability

5.0, 6.0

Returns

The data object that implements the *nameSpace* property.

Description

Creates a new `nameSpace` property in the object that the `common` global property provides. If the common object does not have a *nameSpace* property, this method creates one, assigns the *nameSpace* name to it, and returns the *nameSpace* string. If the `common` data object already has a *nameSpace* property, this method replaces its contents with the new nameSpace value, but does not replace the data object that implements the nameSpace property. Thus, when a property's value changes, other extensions' references to it remain valid.

Control Object

Availability

5.0, 6.0

GoLive creates a Control object when it interprets a `<jsxcontrol>` element in the body of a `<jsxdialog>`, `<jsxpalette>`, or `<jsxinspector>` element.

The value of the `<jsxcontrol>` element's `type` attribute specifies whether GoLive is to create a radio button, an edit field, or some other kind of control when it interprets this element. Thus, each `type` value specifies a particular customized appearance, behavior, and set of JavaScript properties and functions for the Control object that GoLive creates as the result of interpreting the `<jsxcontrol>` tag. For example, a control of type `button` provides the appearance and behaviors of a pushbutton.

All control objects provide certain common behaviors, such as the ability to draw themselves in the location you specify. They also have certain common attributes, such as those which specify the control's position. Any of your `<jsxcontrol>` elements can define these attributes, and your JavaScript code can get or set their corresponding properties in the control objects these tags create.

Each type of control object also provides its own specialized attributes and functions that other types of control objects don't provide; for example, text-entry fields can capture keystrokes, but radio buttons cannot. Thus, some methods and properties described here are valid only for certain types of controls. When called for controls they do not support, these properties and methods return default values or simply ignore the bad input entirely.

- If your JavaScript code tries to set the value of an attribute or property that does not apply to the control it is trying to set, the control ignores the non-applicable input. For example, if you write a JavaScript statement that tries to set the `itemCount` property of a radio button, the control ignores the statement because radio buttons have no such property.
- Similarly, if your script tries to call a function that the control does not supply, the control ignores the statement. For example, if you try to call the `addItem` function of any control other than a popup menu, GoLive ignores the `addItem` function call.

Acquiring Controls

When the user interacts with a control, GoLive passes a control object to the `controlSignal` method of the extension that provided the control. The vast majority of extensions interact with all of their controls in this way.

```
controlSignal(control)
```

Additionally, you can use the value of a control's `name` property to retrieve it in the following ways:

- As a property of a dialog object retrieved from the global `dialogs` array.

```
dialogs[dlgName].controlName
```

- As an element of the global `controls` array.

  `controls[`*controlName*`]`

  `controls[`*num*`] // GoLive 5 only`

The dialog object also provides a `controls` property holding an array that is used similarly; for more information, see "Dialog Object" on page 72.

`dialogs[`*dlgName*`].controls[`*controlName*`]`

Using Controls

For more information on using controls, see the following sections:

- "Implementing the controlSignal Function" on page 65 of the *Extend Script SDK Programmer's Guide*.
- "Controls" on page 292 of Chapter 6, "Event-Handling Functions," in this book.
- "Custom Controls" on page 180 of the *Extend Script SDK Programmer's Guide*.
- "Tree Control" on page 180of the *Extend Script SDK Programmer's Guide*.

Control Object Properties

As noted in the preceding section, the value of a control's `type` property defines its appearance and behavior. Some control properties are type-specific; that is, they are not valid for all controls but only for those of certain types.

`color`	*String*	Get or set the color value for a color field control. HTML color values such as `"red"`, or `"#FF0000"` are valid values for this attribute. This property is valid for color fields only; all other controls return an empty string as color value.
`enabled`	*Boolean*	Get or set the enabled state of the control.
`group`	*Number*	Radio button group ID. This property is valid for radio buttons only. GoLive treats all radio buttons having the same `group` value as a group for the purposes of selection behavior; when one of the buttons in the group is selected, GoLive deselects the others in the group automatically.
`height`	*Number*	The height of the control, expressed as a number of pixels.
`itemCount`	*Number*	The number of menu items in a popup menu. This property is valid for popup menus only. All other controls return 0. Read-only.
`multi`	*Boolean*	Set to true to enable selection of multiple entries in this list box control. Valid for list boxes only. When multi-selection is turned on, the control.selection property returns an *Array* object.
`name`	*String*	The control's JavaScript name, as specified by the `name` attribute of the `<jsxcontrol>` element that defines this control. Read-only.

Control object properties (*continued from preceding page*)

`parent`	*Dialog*	The dialog window (modal or modeless) that displays the control.
`posx`	*Number*	The horizontal location of the control's upper-left corner in the coordinate plane of the dialog or palette that displays the control. The origin of this coordinate plane is the point at the upper-left corner of the dialog that displays the control. The value of the `posx` property specifies the number of pixels by which GoLive offsets the control's location from this origin. Positive values of the `posx` property specify a location to the right of the origin, while negative values specify a location to the left of the origin.
`posy`	*Number*	The vertical location of the control's upper-left corner in the coordinate plane of the dialog or palette that displays the control. The origin of this coordinate plane is the point at the upper-left corner of the dialog that displays the control. The value of the `posy` property specifies the number of pixels by which GoLive offsets the control's location from this origin. Positive values of the `posy` property specify a location below the origin, while negative values specify a location above the origin.
`selection`	*Number*	Get or set the currently-selected item in a popup menu or list box by numeric index. If the list box is multi-selectable, the property returns an Array object containing the index of all selected entries. When setting the property, a multi-selectable listbox turns on the selection for the given entry without deselecting the other entries. Set this property to `-1` to deselect all entries for popups and list boxes. This property is valid only for popup menu controls and list boxes; all other controls return -1.
`state`	*Boolean*	Retrieve or set the control's selected state. Setting this property to `true` causes the control to receive the input focus if possible.
`type`	*String*	The type of the control as specified by the <jsxcontrol> element GoLive interpreted to create the control. Read-only.
`value`	*Type-specific*	The control's current value. The data type this property provides is dependent on this control's type; for example, it may be a color value, a numeric index, or the text of an edit field. For buttons and static text fields, this value specifies the text the control displays to the user.
`values`	*String*	The contents of a list box or a popup menu expressed as a string of comma-separated values, such as `"One,Two,Three"`. Setting this value reloads the entire contents of the list box or popup menu. For other control types, this property acts similarly to the `value` property.
`width`	*Number*	The width of the control, expressed as a number of pixels.

Control Object Functions

The control object itself provides only a few convenience functions. A control's most important behaviors are provided by global event-handling functions that the Extend Script SDK calls when the user interacts with the control.

NOTE: To use a control object, your extension must provide the global event-handling methods that the Controls section of Chapter 6, "Event-Handling Functions," describes. The SDK calls these event-handling functions in response to most user interactions with most controls.

addItem

controlObj`.addItem (`*text*`) // (control.type == menu)`

menuObj`.addItem(`*text*`)`

Availability

5.0, 6.0

Parameter

text	*String*	Text of menu item to create. To add a separator item to the menu, specify a single dash as this value.

Description

Adds a new menu item or menu separator to this control of type `menu`. All other types of controls ignore this method. This method creates a menu item having the empty string ("") as its JavaScript name. You should set this MenuItem object's `name` property to your own unique string value immediately.

beginDraw

controlObj`.beginDraw()`

Returns a temporary Draw object you can call to draw a new *controlObj* appearance.

Availability

5.0, 6.0

Returns

Draw	Draw object you can call to change the appearance of *controlObj*.

Description

Call this method to obtain a temporary Draw object you can use to refresh a custom control's appearance immediately in response to a change in its state. When these drawing operations are complete, call the `endDraw` method to terminate the temporary `Draw` object.

IMPORTANT: *The temporary* `Draw` *object this method returns is not valid after the* `endDraw` *method is called. Do not call the temporary draw object from outside the calls to the* `beginDraw` *and* `endDraw` *methods that create it and terminate it.*

Only one temporary draw object can exist at any time. You cannot "nest" calls to the `beginDraw` method; that is, never call the `beginDraw` method more than once before calling the `endDraw` method.

IMPORTANT: *Each call to the* `beginDraw` *method call must be followed by a call to the* `endDraw` *method. Do not "nest" calls to the* `beginDraw` *method.*

endDraw

controlObj`.endDraw()`

Terminates the drawing operation the `beginDraw` method began.

Availability

5.0, 6.0

Description

This method invalidates the temporary Draw object that the `beginDraw` method created.

refresh

controlObj`.refresh()`

Request a repaint of this control.

Availability

5.0, 6.0

Description

Request a repaint of this control. Call this method to repaint an owner-draw custom control in response to user input.

removeAll

controlObj`.removeAll()`

Remove all items from the *controlObj* control of type `menu`.

Availability

5.0, 6.0

Description

Only controls of type `menu` provide this method. All other controls ignore this method call.

removeItem

controlObj`.removeItem (`*text*`)`

Removes the specified item from this control of type `menu`.

Availability

5.0, 6.0

Parameter

`text`	*String*	The value of the `name` property of the menu item to remove.

Description

Only controls of type `menu` provide this method. All other controls ignore this method call.

setLink

controlObj`.setLink (`*link*`)`

Set the link used by this control of type `urlgetter`.

Availability

5.0, 6.0

Description

Only controls of type `urlgetter` provide this method. All other controls ignore this method call.

Dialog Object

Availability

5.0, 6.0

GoLive creates a Dialog object when it interprets a <jsxdialog>, <jsxpalette>, or <jsxinspector> element. The dialog object provides a dialog window only. The dialog's content, such as its text and buttons, are control objects that GoLive creates when it interprets <jsxcontrol> elements the body of the <jsxdialog>, <jsxpalette>, or <jsxinspector> element contains.

Access to Dialog Objects

The following SDK elements define a dialog having myUnique as its JavaScript name, as well as two control objects named myUniqueButn1 and myUniqueButn2.

```
// portion of main.html file
<jsxdialog name="myUnique" ... >
      ...
   <jsxcontrol name="myUniqueButn1" type="button" value="121081"...>
   <jsxcontrol name="myUniqueButn2" type="button" value="112901"...>
      ...
</jsxdialog>
```

The following JavaScript code shows two ways to retrieve this dialog by name from the global JavaScript namespace.

```
var myDialog = myUnique; // dialog's JS name is a global property
var myDialog = dialogs ["myUnique"] // from dialogs collection
```

You can also retrieve dialogs from the global dialogs collection by numeric index.

The objects in the dialogs array represent dialogs that have run at least once. The SDK adds a dialog's JavaScript representation to the global dialogs array the first time the runModal method displays the dialog. The dialog object is not present in this array until the runModal method has displayed the dialog at least once.

When the user dismisses the dialog, the SDK hides it but does not terminate the dialog object; as a result,

- You can retrieve from the dialogs[] array any dialog that has run at least once, even if the user has already dismissed the dialog.
- The dialog's controls retain the values they had when the user dismissed the dialog.

  If you don't want to display these values, you must reinitialize them before displaying the dialog again.

Displaying and Hiding Dialogs

The means by which you display or hide a dialog is dependent on whether the dialog is modal or modeless.

- A dialog defined with the `<jsxdialog>` tag is displayed modally.
- A dialog defined with the `<jsxpalette>` or `<jsxinspector>` tags is displayed modelessly.

Modal Dialogs

To display a dialog object defined with the `<jsxdialog>` tag, call the dialog object's `runModal` method.

There are two ways to dismiss a modal dialog:

- A `<jsxcontrol>` of type `button` having `dialogOK` or `dialogCancel` as the value of its `name` attribute dismisses its modal dialog automatically when the user clicks it.
- To make another control dismiss its modal dialog, call the dialog's `endModal` method from that control's case in your extension's `controlSignal` method.

Palettes and Inspectors

Although both `<jsxpalette>` and `<jsxinspector>` dialogs are floating palette windows, they are displayed and hidden differently.

By default, GoLive displays a `<jsxpalette>` dialog and installs the palette's name in the **Window** menu the first time it loads the extension that provides the palette. In subsequent user sessions, GoLive displays or hides the palette according to whether it was displayed or hidden at the end of the previous GoLive user session.

To show or hide a palette window programmatically, assign a `Boolean` value to the `visible` property of the dialog object that represents it. Set the dialog object's `visible` property to `false` to hide the window, and set it to `true` to display the window.

```
// assume we created <jsxpalette name="myPalette" ... >
// get myPalette from the global dialogs array
var dlg = dialogs["myPalette"];
// show it if its hidden
if (dlg.visible == false)
   dlg.visible = true;
```

Only `<jsxpalette>` dialogs provide this behavior. The `visible` property is read-only in the Dialog objects that `<jsxinspector>` or `<jsxdialog>` elements create.

GoLive provides a single **Inspector** window that all extensions share. When a box object becomes the focus of input in **Layout** view, GoLive populates this inspector with controls provided by the `<jsxinspector>` element that has the same `classid` value as the box object. Before actually displaying the new controls, the SDK calls the `inspectBox` function of the extension that provides the inspector. Your implementation of the `inspectBox` function initializes the inspector's controls with values retrieved from the box object or custom element being inspected. Whether you obtain these values from the box or from the markup tree is an extension-specific implementation detail.

When the input focus changes to another object, the **Inspector** is repopulated with that object's controls. Interaction with a control invokes its extension's `controlSignal` method.

The user can show or hide the current **Inspector** or any other palette window by choosing its name from the **Window** menu. JavaScript callers cannot control the display of the **Inspector** window.

Dialog Object Properties

Properties of the dialog object provide access to the box it inspects (if the dialog is an inspector), its controls, and information about the dialog, such as its JavaScript name and the title it displays to the user.

`box` *Box*
The box being inspected by this active inspector dialog. This property is `null` for all other dialogs and palettes, and when this inspector dialog is inactive. Read-only.

`controls` *Collection*
Controls this dialog contains, presented as a read-only array that may be indexed numerically or by name. (Numeric index is available in GoLive 5 only.) If the array contains more than one element of the same name, it cannot be predicted which element is returned when selecting by name.

NOTE: When two or more controls in a collection have the same `name` value, those objects cannot be retrieved by name reliably. Because the controls in a dialog object's `controls` array consist only of the controls provided by that particular dialog object, this array may provide more reliable name-based access to poorly-named controls than the global `controls` array.

`focus` *Control*
The Control object which currently has the input focus. Assigning a Control object to this property sets the input focus to that control, if possible.

`frame` *Array*
Returns a four-element array specifying the dialog or palette's size and location. The elements of this array are:

[0] the X position

[1] the Y position

[2] width of the dialog or palette window, expressed as a number of pixels.

[3] height of the dialog or palette window, expressed as a number of pixels.

Inspectors return an array containing four zeroes when they are inactive. Assigning a four-element array to this property sets the size and location of a dialog or palette accordingly; however, inspectors ignore attempts to set this property.

.Dialog object properties (*continued from preceding page*)

`name`	*String*	The dialog's JavaScript name, as specified by the `name` attribute of the `<jsxdialog>` element that defines this dialog. Read-only.
`title`	*String*	The title of the dialog window. Applies only to modal dialogs.
`visible`	*Boolean*	Indicates whether this dialog, palette, or inspector is hidden. Set this property to true to show this palette window, or set it to false to hide this palette window. Modal dialogs and inspector windows ignore attempts to set this property. The value of this property does not indicate whether the dialog is within the boundaries of the screen.

Dialog Object Functions

The Dialog object provides only two functions of its own. These functions provide modal dialog behavior and are meaningful only for a dialog object that represents a `<jsxdialog>` SDK element. Because `<jsxpalette>` and `<jsxinspector>` windows are not intended to provide modal behavior, dialog objects that represent `<jsxpalette>` and `<jsxinspector>` windows ignore calls to their `runModal` and `exitModal` methods.

exitModal

dialogObj`.exitModal(`*n*`)`

Dismisses this modal dialog and causes the currently-executing `runModal` function call to return the value `n` passed as the argument to the `exitModal` method.

Availability

5.0, 6.0

Parameter

n	*String*	The value that the currently-executing `runModal` function call is to return; typically, this value indicates whether the user confirmed, cancelled, or made some other choice in a modal dialog. You can use this argument to cause the `runModal` function call to return any numeric value that suits your development goals.

Returns

Number	The value the currently-executing `runModal` function call is to return.

Description

Dismisses this modal dialog and causes the currently-executing `runModal` function call to return the value `n` passed as the argument to the `exitModal` method. Typically, your extension's `controlSignal` method calls the `exitModal` function when GoLive passes to it a control that dismisses the currently-executing modal dialog.

runModal

dialogObj`.runModal()`

Displays this dialog as a modal dialog.

Availability

5.0, 6.0

Returns

Number	The reason the dialog was terminated: `0` Cancel button clicked `1` OK button clicked `2` Another button clicked

Description

Displays this dialog as a modal dialog. The user must respond to this dialog before taking further action.

When the control that dismissed the dialog is a button having any of the JavaScript names `dialogCancel`, `dialogOK`, or `dialogOther`, this method returns the appropriate value automatically.

To set the value this method returns, don't use the `dialogCancel`, `dialogOK`, or `dialogOther` JavaScript names for your dialog's controls. To dismiss the dialog, the dismissing control's case in your `controlSignal` method calls the dialog's `endModal` method, passing as its argument the value the `runModal` method is to return.

If you have specified a timeout for scripts, it is disabled while the modal dialog is visible. For more information, see "Setting the JavaScript Timeout" on page 194 of the *Extend Script SDK Programmer's Guide*.

document Object

Availability

5.0, 6.0

The document object represents an open document. The precise set of behaviors and properties a particular document object provides depends on whether it represents a GoLive site document, a markup document such as the HTML file that defines a Web page, or some other kind of document, such as a text file.

For more information on working with document encodings, see the following sections:

- "Disabling Encoding Alerts" on page 44
- "Using a Specific Encoding to Read a Document" on page 44
- "Specifying the encoding that get/setinner/outerHTML methods use" on page 148

Methods of the File Object support the use of alternate encodings, also.

Acquiring document Objects

The `document` global variable always holds the Document object that represents the frontmost document window in GoLive.

```
document; // the active (frontmost) document
```

You can retrieve any document by its unique JavaScript name or by numeric index from the global `documents` array, as the following code demonstrates.

```
documents["mydoc.html"]; // the mydoc.html document
documents[n]; // the nth document in the collection
```

document Object Properties

The properties of the document object provide information about the current document, its elements, and the site that incorporates the page the document object represents.

- Every document object's properties provide
  - A description of its type (markup, site, or other).
  - A title string displayed to the user.
  - A unique JavaScript name.
  - A File object representing the disk file from which it was read.
  - The line break mode (Mac OS, Windows, or Unix) it uses.
  - A description of how it is displayed: Layout View, Source View, Outline View, and so on.
  - Access to its undo history.

- If the document object represents a markup document, additional properties provide
  - Access to the tree of markup elements GoLive generated when interpreting the document.
  - A reference to the site that contains the document. This reference is always available, even when the site file is not open. You can use this reference to navigate among the documents in a site programmatically.
  - A reference to the site window if it is open.
- If the document object represents a Site document, additional properties reference
  - The root folder of the site that uses the document.
  - The home page of the site that uses the document.

`element`	*Markup*	The tree of markup objects GoLive generated when it interpreted this markup document. GoLive does not generate a markup tree when it reads other kinds of documents, such as non-HTML text files or site documents; for such documents, this value is `null`. Read-only.
`encoding`	*String*	The character encoding used for the document. Values are the same strings used by the `encoding` attribute of the `<META>` tag, such as "`iso-8859-1`" or "`shift_jis`". Read-only.
`file`	*File*	A *File* object representing the disk-based representation of this document. If the document has not yet been saved, this value is `null`. Take care to use this object appropriately; for example, deleting or renaming an open document before saving it is likely to produce adverse results.
`history`	*History*	Undo/Redo history object. You can use this object to determine the number of currently-defined Undo/Redo actions and select one. When you select an action in the History object, GoLive executes the actions necessary to restore the document to the state specified by that action's position in the undo/redo history.
`homePage`	*SiteReference*	Reference to the home page of the site this document represents, usually a file reference. The property is `null` for documents that do not represent GoLive `.site` files. Read-only.
`lineBreakMode`	*Settings*	One of the strings "`mac`", "`unix`", "`win`" or "`mixed`". Can be set to "`mac`", "`unix`" or "`win`". When setting the value, the document's line breaks are actually changed.
`mainTextArea`	*TextArea*	Object that manages the GoLive 6 **Layout** view representation of a page layout document's main content area. When **Layout** view is active (`document.view == layout`), you can call this TextArea object's methods to insert text or boxes into the `document`. GoLive 6 only.
	null	The document that supplies *markupObj* is not displaying Layout view.
`ref`	*SiteReference*	Reference to the site that owns this document. This reference is available only when the corresponding Site document is open.

Document object properties (*continued from preceding page*)

`selection`	*Selection*	The current user selection in a markup document. This value can represent a selected range of text, a set of selected elements, or the current cursor position. Defined for markup documents only, null otherwise. Read-only. **IMPORTANT:** *Any change to the markup tree—even one that does not reparse the document—makes the current selection undefined. Reparsing the document (page 103 Guide) also makes the current selection undefined. After the markup tree is changed or the document is reparsed, your extension's JavaScript code must retrieve a new reference to the current selection from this property.*
`site`	*SiteReference*	Reference to the root of the site this site document represents, usually a reference to a folder. This property is `null` for non-site documents. Read-only.
`title`	*String*	The document's title, as displayed to the user. Read-only.
`type`	*String*	The type of document this object represents. The `markup` string indicates that this document defines a markup tree. The `site` string specifies that this document is a GoLive site document. The `unknown` string represents all other types of documents. Read-only.
`view`	*String*	The current view of the document. Read this value to determine the current view, or set this value to change the current view. Valid values are `layout` **Layout** view. `frame` **Frames** view. `source` **HTML Source** view. `outline` **Outline** view. `preview` **Preview** view. `webobjects` **WebObjects** view. `unknown` Not a layout document. Changing a document's view from `layout` to any other view causes GoLive to delete the document object because other views have different object models. Changing a document's view to `layout` causes GoLive to reparse the document; on Mac OS platforms, changing a document's view to `preview` also results in a reparse, because this view is a special read-only version of the layout view. On Windows platforms, Internet Explorer displays the preview, so GoLive does not create a new document when it switches to preview mode.
`website`	*Website*	Website that uses this document. The site window must be open, but it need not be frontmost. GoLive 6 only.

document Object Functions

The functions of the document object enable you to create an undo object for the document it represents, as well as to save, close, reparse, or reformat the document this object represents.

clone

documentObj`.clone()`

Returns an unsaved, untitled, not dirty copy of the *documentObj* document.

Availability

5.0, 6.0

Returns

Document	A copy of the called document object. Regardless of the state of the called object, the document returned is unsaved, untitled, and not marked as changed.

close

documentObj`.close()`

documentObj`.close (`*discard*`); // GoLive 6 only`

Closes the document.

Availability

5.0, 6.0

Parameter

discard	*Boolean*	Optional. Passing `true` discards changes to the document before closing it. GoLive 6 only.

Description

If the document has not been saved previously, this method prompts the user for a filename. If the document has unsaved changes, this method prompts the user to save or discard changes. To suppress user prompts and close the document without saving changes, pass `true` as the value of the optional *discard* parameter. (GoLive 6 only.)

If GoLive closes the document successfully, the called Document object is invalid when this method returns.

createUndo

documentObj`.createUndo (`*text*`)`

Creates an Undo object.

Availability

5.0, 6.0

Parameter

`text`	*String*	Text to display as the name of this undo action in the History palette and in the Undo menu item.

Returns

`Undo`

reformat

documentObj`.reformat()`

Does everything the `reparse` method does, and formats the source view of the document for printing.

Availability

5.0, 6.0

Description

Does everything the `reparse` method does, and formats the source view of the document for printing. For Site documents, this method operates on every page in the site.

IMPORTANT: *Do not call this method from within the* `parseBox` *or* `undoSignal` *methods.*

repairActions

documentObj`.repairActions()`

Regenerates JavaScript for GoLive actions on the page.

Availability

6.0

Description

Works only in **Layout View**. This method updates GoLive-specific JavaScript code; specifically, the code that the `<csscriptdict>` and `<csactiondict>` tags provide.

In GoLive 6, the `Document` object's `reparse` method does not affect action code; to regenerate such code, call this method after calling the `document.reparse()` method.

reparse

documentObj`.reparse()`

After changing the source representation of a markup element, call this method to generate a new markup tree that reflects these changes in the Layout view of the document.

Availability

5.0, 6.0

Description

After changing the source representation of a markup element, call this method to generate a new markup tree that reflects these changes in the Layout view of the document. Saved references to objects representing the contents of the document, such as boxes and markup elements, are invalid when this method returns. For Site documents, this method operates on every page in the site and updates the list of files the Site window displays; after adding files to a site programmatically, call this method to update the Site window.

IMPORTANT: *Do not call this method from within the* `parseBox` *or* `undoSignal` *methods.*

save

documentObj`.save()`

Saves the document as a disk file.

Availability

5.0, 6.0

Description

Saves the document as a disk file. If the document has not been saved previously, prompts the user for a filename.

saveAs

documentObj.saveAs (*fileName*)

Saves the document as a disk file, always displaying the file-saving dialog that is standard for the platform on which GoLive is running.

Availability

5.0, 6.0

Parameter

fileName	*String*	URL or platform-specific path name specifying the default name and location that appears in the file-saving dialog. The actual name and location used to save the file are those which the user confirms when dismissing the dialog.

Description

Saves the document as a disk file, always displaying the file-saving dialog that is standard for the platform on which GoLive is running. This dialog displays the default name and location the `fileName` argument specifies; before dismissing the dialog, the user can confirm or change these default settings.

selectElement

documentObj.selectElement(*element*)

Select the specified *element* in **Layout** view.

Availability

6.0

Parameters

element	*Markup*	Element to select.

Returns

Boolean	`true`	Selected specified element successfully.
	`false`	Any of the following error conditions may apply: - The *documentObj* document is not in layout mode. - The *element* does not exist as specified.

document.history Object

Availability

5.0, 6.0

The SDK makes the `history` object available as a property of the document Object. The `history` object provides access to the document's Undo/Redo history.

history Object Properties

Properties of the `history` object provide information about the current Undo history, such as the number of actions it holds and the index of the current undo history state.

`current`	*Number*	Index position that represents the document's current state in the undo/redo history. This value is a number between 0 and the value of the `length` property. Setting this value causes GoLive to perform all undo/redo actions necessary to adopt the document state the new index position represents.
`length`	*Number*	The number of history entries. Read-only.
`maxCount`	*Number*	The maximum count of undo/redo actions this document allows. This value is a number between 1 and 999. Setting this property changes its value only for the current GoLive user session. The user can set this value permanently in the History palette.
[*int*]	*String*	The History object's numeric index returns the name of the Undo/Redo action that occupies the specified position. Read-only.

document.selection Object

Availability

5.0, 6.0

The `selection` object describes the current selection in a document object. Most of its properties are read-only, except for the `element` property. Assigning a markup element to this property selects that element. Assigining `Null` or anything other than a markup element to this property discards the current selection.

Acquiring selection Objects

You can get this object from the `selection` property of a Document object.

```
document.selection // when Layout view is open
```

selection Object Properties

Properties of the selection object provide access to the current selection in the document that provides the called selection object.

`box` *Markup* When the selection is a custom element defined with the `<jsxelement>` tag, the `selection` object's `box` property contains the box object that manages this element's visual representation in Layout view. For all other elements, the `box` property is `null`. Read-only.

`element` *Markup* The first selected element. Assign a Markup object to this property to attempt to select its corresponding element. Assigning anything other than a Markup object to this property discards the current selection.

`length` *Number* The length of the selection. Read-only.

`start` *Number* Offset from the first character in the outer HTML of the selected element's source representation to the first character of the current selection. Read-only.

`text` *String* The selected text, according to the current values of the `start` and `length` properties. Read-only.

`type` *String* Read-only description of the kind of selection. Values are:

- `point` No selection. The selection reflects the position of the cursor in the document's HTML source.
- `part` Part of the current markup element is selected. For example, this value would indicate a partial selection within a simple text block.
- `full` The entire markup element is selected. For example, when the user clicks a box object in Layout view, the SDK selects the entire markup element the box represents.
- `complex` More than one markup element is selected or partially selected.

Draw Object

Availability

5.0, 6.0

The Draw object provides methods extensions can call to perform basic drawing operations. This object provides a cursor for drawing lines and text. When calling methods of the draw object, specify coordinates in the coordinate plane of the box or window that displays the draw object being called.

Acquiring Draw Objects

The SDK passes a `Draw` object to your extension's `drawBox` and `drawControl` methods.

The `beginDraw` method of the Control object also returns a draw object.

Draw Object Functions

Functions of the Draw object draw primitive graphics, text and images. Your extension's `drawControl` method can call these functions to refresh the appearance of an owner-draw control. Most extensions do not create owner-draw controls. The SDK redraws all controls other than those of type `custom` for you.

drawString

drawObj`.drawString (`*text*`)`

Draws a string at the graphics cursor's current location.

Availability

5.0, 6.0

Parameter

`text`	*String*	Text string this method draws.

fillOval

drawObj`.fillOval (`*x*`, `*y*`, `*width*`, `*height*`)`

Draws a filled oval in the specified location.

Availability

5.0, 6.0

Parameters

`x`	*Number*	X-coordinate of the upper-left corner of the smallest rectangle that would enclose the oval to draw.
`y`	*Number*	Y-coordinate of the upper-left corner of the smallest rectangle that would enclose the oval to draw.
`width`	*Number*	The oval's width, expressed as a number of pixels.
`height`	*Number*	The oval's height, expressed as a number of pixels.

fillRect

drawObj.`fillRect (`*x*`, `*y*`, `*width*`, `*height*`)`

Draws a filled rectangle in the specified location.

Availability

5.0, 6.0

Parameters

`x`	*Number*	X-coordinate of the upper-left corner of the rectangle to draw.
`y`	*Number*	Y-coordinate of the upper-left corner of the rectangle to draw.
`width`	*Number*	The rectangle's width, expressed as a number of pixels.
`height`	*Number*	The rectangle's height, expressed as a number of pixels.

frameOval

drawObj.`frameOval (`*x*`, `*y*`, `*width*`, `*height*`)`

Draws an outlined oval in the specified location.

Availability

5.0, 6.0

Parameters

`x`	*Number*	X-coordinate of the upper-left corner of the smallest rectangle that would enclose as the oval to draw.
`y`	*Number*	Y-coordinate of the upper-left corner of the smallest rectangle that would enclose the oval to draw.
`width`	*Number*	The oval's width, expressed as a number of pixels.
`height`	*Number*	The oval's height, expressed as a number of pixels.

frameRect

drawObj.`frameRect (`*x*`,` *y*`,` *width*`,` *height*`)`

Draws an outlined rectangle in the specified location.

Availability

5.0, 6.0

Parameters

`x`	*Number*	X-coordinate of the upper-left corner of the rectangle to draw.
`y`	*Number*	Y-coordinate of the upper-left corner of the rectangle to draw.
`width`	*Number*	The rectangle's width, expressed as a number of pixels.
`height`	*Number*	The rectangle's height, expressed as a number of pixels.

getDrawInfo

drawObj.`getDrawInfo()`

Retrieves a magic number to pass to an external binary library that implements custom drawing.

Availability

5.0, 6.0

Returns

Number	The external library function casts this number to a pointer. The pointer provides a `JSADrawInfo` structure for the external function's use.

Description

Retrieves a magic number to pass to an external binary library that implements custom drawing. The external function to which this number is passed casts it to a pointer. This pointer references a `JSADrawInfo` structure GoLive provides to support native drawing operations.

When GoLive is running on a Windows platform, this `JSADrawInfo` structure provides a DC for the external function's use. When GoLive is running on a Mac OS platform, this `JSADrawInfo` structure provides a `GrafPort` for the external function's use. The `JSADrawInfo` structure always provides the coordinates of the upper-left and lower-right corners that define the drawing area.

```
typedef struct _JSADrawInfo {
     long context;// a DC (Windows) or a GrafPort (Mac)
     long left, top;// upper left corner of the drawing rect
     long right, bottom;// lower right corner of the drawing rect
} JSADrawInfo;
```

invertRect

drawObj.invertRect (*x*, *y*, *width*, *height*)

Inverts the colors of the specified rectangle, producing an XOR effect.

Availability

5.0, 6.0

Parameters

x	*Number*	X-coordinate of the upper-left corner of the rectangle to draw.
y	*Number*	Y-coordinate of the upper-left corner of the rectangle to draw.
width	*Number*	The rectangle's width, expressed as a number of pixels.
height	*Number*	The rectangle's height, expressed as a number of pixels.

Description

Inverts the colors of the specified rectangle, producing an XOR effect. Calling this method a second time undoes the effect of the first call to this method.

lineTo

drawObj.lineTo (*x*, *y*)

Draws a line from the pen's current location to the specified position and moves the graphics cursor to the new location.

Availability

5.0, 6.0

Parameters

x	*Number*	X-coordinate of the endpoint of the line this method draws; also the x-coordinate of the pen's final location when this method returns.
y	*Number*	Y-coordinate of the endpoint of the line this method draws; also the y-coordinate of the pen's final location when this method returns.

moveTo

drawObj.moveTo (*x*, *y*)

Moves the graphics cursor to the specified location.

Availability

5.0, 6.0

Parameters

x	*Number*	X-coordinate of the pen's new location.
y	*Number*	Y-coordinate of the pen's new location.

penSize

drawObj.penSize (*width*)

Sets the width of the drawing pen.

Availability

5.0, 6.0

Parameters

width	*Number*	The pen's new width, expressed as a number of pixels.

setColor

drawObj.setColor (*color*)

Sets the pen's *color* as specified.

Availability

5.0, 6.0

Parameters

color	*String*	The pen's new color, expressed as an HTML color string or an RGB triplet.

Description

Examples of valid HTML color strings are "red" or "#FF0000". An RGB triplet value consists of three comma-separated integer values between 0 and 255. The values in the triplet specify the amount of red, green, and blue, respectively, that define the new color; for example (255, 0, 0) would light all red pixels at 100% with no green or blue pixels lit.

stringWidth

drawObj.`stringWidth` (*text*)

Using current font settings, returns a value specifying the width of the specified string as a number of pixels.

Availability

5.0, 6.0

Parameter

`text`	*String*	The text this method measures.

Returns

Number	Number of pixels wide the `text` argument would be if drawn using current font settings.

stringHeight

drawObj.`stringHeight` (*text*)

Using current font settings, calculates the height of the specified string as a number of pixels.

Availability

5.0, 6.0

Parameters

`text`	*String*	The text this method measures.

Returns

Number	Number of pixels tall the `text` argument would be if drawn using current font settings.

textFace

drawObj.textFace (*bits*)

Sets the font face GoLive uses for subsequent text output.

Availability

5.0, 6.0

Returns

Number Font face to use for text output, specified as one or more of the following values:

1	Bold
2	Italic
4	Underlined (Mac OS only)
8	Outlined (Mac OS only)
16	Shadowed (Mac OS only)
32	Condensed (Mac OS only)
64	Extended (Mac OS only)

Add these values to specify multiple faces; for example the value 3 specifies that text output is to be bolded and italicized.

textFont

drawObj.textFont (*fontName*)

Sets the font GoLive uses for subsequent text output.

Availability

5.0, 6.0

Parameter

fontName	*String*	Name of the font to use for text output; for example, "Courier" is a valid value. Pass "ApplicationFont" as this value to set the font to the application font. The application font is the font GoLive uses to draw static text and control labels; contrast with the system font.

textSize

drawObj.`textSize (`*points*`)`

Sets the size of the font GoLive uses for subsequent text output.

Availability

5.0, 6.0

Parameter

`points`	*String*	Size of the font to use for subsequent text output, expressed as a number of typographer's points. One inch is equivalent to approximately 72 typographer's points.

DynamicContent Object

Availability

6.0

The `DynamicContent` object provides programmatic access to Dynamic Content features of the **Dynamic Bindings** palette and the **Site Settings** window in version 6.0 of GoLive.

Acquiring the DynamicContent Object

You can obtain the `DynamicContent` object from the global variable of the same name.

```
DynamicContent
```

DynamicContent Object Functions

`DynamicContent.goOnline`()
: Attempts to set the state of Dynamic Content to online.

`DynamicContent.goOffline`()
: Sets the state of Dynamic Content to offline.

`DynamicContent.isDynamic`()
: Returns `true` if the current document is dynamic.

`DynamicContent.makeDynamic`()
: Makes the current document dynamic.

`DynamicContent.makeStatic`()
: Makes the current document static.

`DynamicContent.getOutputFormat`()
: Returns the language (output format) of the current document.

`DynamicContent.setOutputFormat`(*language*)
: Adds the specified *language* to the current document's output format.

goOnline

`DynamicContent.goOnline()`

Attempts to set the state of Dynamic Content to online.

Availability

6.0

Returns

Boolean	`true` indicates success.

goOffline

`DynamicContent.goOffline()`

Sets the state of Dynamic Content to offline.

Availability

6.0

Returns

Boolean	`true` indicates success.

isDynamic

`DynamicContent.isDynamic()`

Returns `true` if the current document is dynamic.

Availability

6.0

Returns

Boolean	`true`	Current document is dynamic.
	`false`	Current document is not dynamic.

makeDynamic

`DynamicContent.makeDynamic()`

Makes the current document dynamic.

Availability

6.0

Returns

Boolean	`true`	The method succeeded.
	`false`	The method failed.

makeStatic

`DynamicContent.makeStatic()`

Makes the current document static.

Availability

6.0

Returns

Boolean	`true`	The method succeeded.
	`false`	The method failed.

getOutputFormat

`DynamicContent.getOutputFormat()`

Returns the current language (output format) of the current (frontmost or active) document.

Availability

6.0

Returns

Boolean	`true`	The method succeeded.
	`false`	The method failed.

setOutputFormat

`DynamicContent.setOutputFormat(language)`

Adds the specified *language* to the current document's output format.

Availability

6.0

Parameters

language	*String*	Name of the output format to add.

Returns

Boolean	`true`	The method succeeded.
	`false`	The method failed.

File Object

Availability

5.0, 6.0

The File Object enables Extend Script extensions to create and manipulate files and folders on local file systems and HTTP servers. This object provides methods that read, write, and append to files in a manner similar to the file-streaming functions of the C-language `stdio.h` library. Other methods of the file object create files or folders, copy files, and move files.

A newly-constructed file object simply encapsulates a path name. The existence of a File object does not imply the existence of the referenced file or folder; for example, you might

have created the file object in order to call methods that create files or folders programmatically. You must validate all file objects before using them:

- Ensure that the file object is neither null nor undefined.
- Ensure that the file object represents a folder or file, as appropriate.
- If the file object represents an existing folder or file, determine whether the file or folder actually exists on a disk available to the host platform.

Acquiring File Objects

You can obtain a file object from the app Object, or you can create one yourself.

- Properties of the Application object provide File objects that represent commonly-used folders, such as the one that holds the GoLive application.

```
app.currentFolder
app.folder
app.settingsFolder
app.systemFolder
app.tempFolder
app.trashFolder
```

- To access other directories or files, use the File Object Constructor to create your own File objects explicitly.

```
JSXFile (path)
```

File Object Properties

A file object always represents a file-system location. Properties of the file object present this location as a URL and in platform-specific formats for Windows and Mac OS platforms. Additional properties provide information about this file-system reference, such as whether it represents a file or a folder, and whether the referenced file or folder exists at the specified location.

`alias`	*String*	GoLive 6.0 on Mac OS only: The alias name of an existing Mac OS file system alias to the disk entity the file object represents.
	Link	GoLive 6.0 on Windows only: A link to the file or folder the file object represents.
`eof`	*Boolean*	`true` indicates that GoLive encountered an EOF condition during a read operation. Read-only.
`encoding`	*String*	The name of the document encoding charset this file uses.
`error`	*Boolean*	`true` indicates that an error was detected during a read or write operation. Read-only.

File object properties (*continued from preceding page*)

`exists`	*Boolean*	`true` indicates that the file or folder this file object represents is available at the location this File object represents. Read-only.
`isFolder`	*Boolean*	`true` when the file object holds the path to a folder that is currently available to the host platform's file system; that is, the `true` value confirms the folder's existence as well as the fact that it's not a file. Returns `false` when the file object holds the path to a file or the folder referenced by this File object is not available. Read-only.
`lastError`	*String*	Holds explanatory text describing the last I/O error related to this file. You can use this property to check the result of an HTTP upload or download operation. You can set this property to any string value; to clear the current value, set this property to the empty string.
`macCreator`	*String*	On Mac OS platforms, the referenced file's creator ID as a four-character string. The `"????"` value indicates that GoLive cannot determine this file's Mac OS creator code, or that GoLive is running on a Windows platform. Read-only.
`macType`	*String*	On Mac OS platforms, the referenced file's file type as a four-character string. The `"????"` value indicates that GoLive cannot determine this file's Mac OS file type, or that GoLive is running on a Windows platform. Read-only.
`name`	*String*	The referenced file's filename without its path. Read-only.
`parent`	*File*	The parent object of this file object. this value is `null` when this file object represents an entity in the root directory of the file system.
`path`	*String*	Returns the fully-qualified path to the referenced file using the conventions of the current host file system; in other words, this value is specific to the platform on which GoLive is running. Read-only.
`readOnly`	*Boolean*	Mirrors the Readonly/Locked flag of the referenced file on disk. May be set to change the status of that flag.
`size`	*Number*	Returns the file size (number of characters.) The value `0` indicates that this file object represents a folder. Read-only.
`url`	*String*	Returns the path to the referenced file encoded as a fully-qualified URL. Read-only.

File Object Functions

Functions of the file object read, write, create, delete, move, copy, rename, and provide information about files or folders. The File object methods that create a file, read data from a file, and write data to a file are not equivalent to the user commands that perform such tasks. They are similar to functions that the C-language `stdio.h` library provides.

NOTE: You must validate a file object before calling its functions. The existence of a file object does not imply the existence of the file or folder it represents.

Functions of the File object may fail. This object issues requests to a local or remote file system that actually manipulates the disk-based resource the File object represents. Such requests may not complete; if your script depends on successful completion of a file operation, be sure to check the result the File method returns.

File Object Constructor

`JSXFile(`*path*`)`

Returns a new File object that encapsulates the specified *path* value.

Availability

5.0, 6.0

Parameter

path	*String*	The pathname with which the File object is initialized.

Returns

File Object

Description

If the *path* argument is omitted, the function returns a File object that refers to the current working directory, as reported by the `app.currentFolder` property.

The value of the *path* argument is a pathname or as a URL. Specifying this argument as a URL avoids platform-specific differences in pathname specifications. When this argument is a pathname, it must observe the file system conventions of the host platform, such as the use of the appropriate delimiter character. On Windows platforms, use a backslash (\) to separate directories within a pathname, and on Mac OS, use a colon (:) to separate directories within a pathname. The presence of the colon (:) character causes the file object to consider the path value to be a Mac OS pathname; thus, pathnames that include dates, such as `"BackupFolder:12/19/01"`, are acceptable to this constructor.

Example

```
JSXFile ("MySite:12/19/01:myPage.html) // Mac OS pathname (:) w/ date
```

close

fileObj`.close()`

Attempts to close the file, saving it to the disk if it has changed.

Availability

5.0, 6.0

Returns

Boolean	`true` indicates that the file was closed successfully.

copy

fileObj`.copy (`*newPath*`)`

Attempts to copy the *fileObj* file or folder to the *newPath* location, and returns a value indicating success or failure.

Availability

5.0, 6.0

Parameter

newPath	*String*	The file's new location, specified as a partial pathname, a full pathname or a local URL.

Returns

Boolean	`true` indicates that ● The disk-based entity this *fileObj* represents was copied successfully. ● The called *fileObj* has been updated to reflect the referenced file or folder's new location on disk.

createFolder

fileObj`.createFolder()`

Attempts to create a folder at the location the called file object represents.

Availability

5.0, 6.0

Returns

Boolean	`true` indicates that the folder was created successfully.

get

fileObj`.get (`*remoteURL* `[,` *mimeType*`])`

Downloads a file from a remote HTTP server as the *fileObj* file.

Availability

5.0, 6.0

Parameters

remoteURL	*Number*	The remote file to download.
mimeType	*String*	Optional. This file's MIME type.

Returns

Boolean	`true` indicates that the file was retrieved successfully. If this value is `false`, the f*ileObj*.`lastError` property holds the last HTTP status code received from the remote server.

Description

If a file already exists on disk at the exact pathname *fileObj* specifies, this method overwrites it.

getFiles

fileObj.getFiles ([*mask*, *type*])

Returns references to files or folders in the folder the called *fileObj* represents.

Availability

5.0, 6.0

Parameter

mask	*String*	Optional. The string a File object's filename must match to be included in the result. Question marks and asterisks in this string are wildcard characters interpreted as on Windows platforms: A question mark (?) represents exactly one occurrence of any character. For example, the search mask `Test?.html` retrieves folders or files named `Test1.html`, `Test2.html`, `TestN.html`, `TestX.html`, and so on. However, it would not retrieve folders or files named `Test.html` or `Test11.html`. An asterisk (*) represents zero or more occurrences of a single character. For example, the search mask `Test*.html` retrieves folders or files named `Test.html`, `Test1.html`, `Test11.html`, `Test111.html`, and so on. Passing no `mask` value, passing a single asterisk (*), or passing the special string `"anymask"` retrieves all files and folders in the directory the called *fileObj* represents.
type	*String*	Optional. Mac OS file type, expressed as a case-sensitive four-character string; for example, `TEXT` or `JPEG` would be valid values for this parameter. When running on the Windows platform, the SDK ignores this parameter.

Returns

Array	Each element of this array holds a File object representing one file or one folder in the folder the called *fileObj* represents. The return value is `null` if the call is not used on a folder object.

Description

All parameters are optional; you use them to specify additional criteria an item must meet to be included in this function's result. For example, you could use the `mask` parameter to retrieve only files having names which end in the `.html` extension. To retrieve File objects representing every file or folder in a particular folder, you need not specify any search mask;

by default, the `getFiles` method uses a search mask that consists of the asterisk (`*`) character only.

This method operates only in the folder the called *fileObj* represents. To retrieve the contents of a subfolder of the *fileObj* folder, you must write your own code that calls the `getFiles` method of a File object that represents the subfolder.

Example

The following example returns File objects representing all `.jpg` files in the current working directory.

```
// get all .jpg files in the current working directory
var theDir = new JSXFile();
var returnedItems = theDir.getFiles("*", "JPEG");
```

To retrieve references to folders only, pass the `fldr` string as the *mask* argument, like the following example does.

```
var returnedItems = theDir.getFiles("*", "fldr");
```

On Mac OS, you pass any Mac OS four-character file type as the value of the second argument.

```
var returnedItems = theDir.getFiles("*", "CARO");
```

move

fileObj`.move (`*pathName*`)`

Attempts to move the *fileObj* file to the *pathName* location, and returns a value indicating success or failure.

Availability

5.0, 6.0

Parameter

pathName	*String*	The file's new location, specified as a partial pathname, a full pathname or a local URL.

Returns

Boolean	`true` indicates that ● The disk-based entity this *fileObj* represents was moved successfully. ● The called *fileObj* has been updated to reflect the referenced file or folder's new location on disk.

Description

This method does not operate on folders. You can provide the illusion of moving a folder by creating your own destination folder, moving files into it, and then deleting the empty original folder.

open

fileObj.open ([*openMode*, *charSet*])

Open the *fileObj* file in the specified *openMode*, using the *charSet* document encoding.

Availability

5.0, 6.0

Parameters

openMode *String* Optional. Permissions for subsequent reading or writing operations on this file; equivalent to open modes used by the C-language `fopen()` library function. Valid values are:

- `r` Open for reading. If the file does not exist or cannot be found, the call fails.
- `w` Opens an empty file for writing. If the file exists, its contents are destroyed.
- `a` Open for writing at the end of the file (appending); creates the file if it does not exist.
- `r+` Opens for both reading and writing. The file must exist on disk already; this open mode does not create it.
- `w+` Opens an empty file for both reading and writing. If the file exists, this open mode overwrites it, destroying the file's previous content.
- `a+` Opens for reading and appending; creates the file if it does not exist. When a file is opened in the "a" or "a+" modes, all write operations are appended to the end of the file. You can still reposition the file pointer in this mode, but the SDK always moves it back to the end of the file before it carries out any write operation in this mode. Thus, you can't overwrite the file's existing data when you open it in this mode.
  When this method opens an existing file for reading/writing in "a+" mode, it searches for a **CTRL+Z** at the end of the file and removes it, if possible.
- `t` Open in text (translated) mode. In this mode, **CTRL+Z** is interpreted as an end-of-file character on input, and the system attempts to replace **CRLF** sequences with Line Feed characters. This is the default mode.
- `b` Open in binary (untranslated) mode; translations involving carriage-return and linefeed characters are suppressed. May be appended to one of the other opening modes, as in the following example.

```
var myDoc = fileObj.open("r+b");
```

charSet *String* Optional. The name of a document encoding charset. When this argument is supplied, the method changes the encoding of the *fileObj* file accordingly. When this argument is omitted, the method attempts to detect the file's encoding; if successful, it sets *fileObj*.encoding accordingly.

Returns

Boolean `true` indicates that the file was opened successfully in the specified open mode.

openDocument

fileObj.openDocument([*charSet*]) // GoLive 6 only

fileObj.openDocument()

Attempts to open the *fileObj* file in a new document window; the programmatic equivalent of using the **File>Open...** menu item to open an existing disk file.

Availability

5.0, 6.0

Parameter

charSet (6.0 only)	*String*	Optional. The name of an encoding character set. This character set overrides the default encoding that GoLive would use if the document did not contain a META tag. The presence of a <META> tag in the *fileObj* document causes GoLive to ignore this argument. If this argument specifies an unknown or unsupported encoding, GoLive generates an "Unsupported encoding" runtime error.

Returns

Document	The document object added to the documents global array when *fileObj* was opened successfully.
Null	Encountered an error.

openMarkup

fileObj`.openMarkup([`*encoding*`]) // GoLive 6 only`

fileObj`.openMarkup()`

Reads the contents of the specified file into a new document object without displaying it in a window.

Availability

5.0, 6.0

Parameter

encoding (6.0 only)	*String*	Optional. The name of an encoding character set. This character set overrides the default encoding that GoLive would use if the document did not contain a META tag. A META tag causes the supplied encoding to be ignored. If the encoding name is unknown or unsupported, GoLive generates an "Unsupported encoding" runtime error.

Returns

Document	The document object added to the documents global array when *fileObj* was opened successfully.
`Null`	Encountered an error.

Description

Opening the document in this way enables you to modify it without causing GoLive to reparse.

put

fileObj.put (*remoteURL* [, *mimeType*])

Uploads the *fileObj* file to the *remoteURL* location on a remote HTTP server.

Availability

5.0, 6.0

Parameters

remoteURL	*Number*	Path to the remote location that is to store the uploaded file.
mimeType	*String*	Optional. The *fileObj* file's MIME type.

Returns

Boolean	`true` indicates that the file was uploaded successfully. If this value is `false`, the *fileObj*.`lastError` property holds the last HTTP status code received from the remote server.

Description

NOTE: The server must be able to fulfill HTTP `PUT` requests

read

fileObj.read ([*count*, *charSet*]) // GoLive 6 only

fileObj.read (*count*)

Reads the contents of the *fileObj* file from the current position forward.

Availability

5.0, 6.0

Parameters

count	*String*	Optional. The number of characters to read. When this argument is omitted, the method reads the entire *fileObj* file.
charSet *(6.0 only)*	*String*	Optional. The name of an encoding character set. This character set overrides the default encoding that GoLive would use if the document did not contain a META tag. The presence of a <META> tag in the *fileObj* document causes GoLive to ignore this argument. If this argument specifies an unknown or unsupported encoding, GoLive generates an "Unsupported encoding" runtime error. When this argument is omitted, the method attempts to detect the file's encoding; if successful, it sets *fileObj*.encoding accordingly.

Returns

String	The characters read from the file. This string never contains more than *count* characters.

Description

Reads the contents of the *fileObj* file from the current position forward. When in translated mode, translates CRLF character sequences to Line Feed characters.

If you pass no arguments to this method, it reads the entire content of the *fileObj* file.

readln

fileObj.readln([*charSet*]) // GoLive 6 only

fileObj.readln()

Reads one line of text from the *fileObj* file, using the *charSet* document encoding if supplied.

Availability

5.0, 6.0

Parameter

charSet (6.0 only)	*String*	Optional. The name of an encoding character set to be used for this method call only. This character set overrides the default encoding that GoLive would use if the document did not contain a META tag. The presence of a <META> tag in the *fileObj* document causes GoLive to ignore this argument. If this argument specifies an unknown or unsupported encoding, GoLive generates an "Unsupported encoding" runtime error. When this argument is omitted, the method attempts to detect the file's encoding; if successful, it sets *fileObj*.encoding accordingly.

Returns

String	The line of characters read from the file.

Description

Reads one line of text from the *fileObj* file. In binary mode, line feeds are recognized as CR, LF or CRLF pairs.

remove

fileObj.remove()

Deletes the file or folder that the called *fileObj* represents.

Availability

5.0, 6.0

Description

Deletes the file or folder that the called *fileObj* represents.

IMPORTANT: *The* remove *method deletes the referenced file or folder immediately. It does not move the referenced file or folder to the system trash. The effects of the* remove *method cannot be undone. It is recommended that you prompt the user for permission to delete a file or folder.*

rename

fileObj.rename (*newName*)

Attempts to set the *fileObj* object's name property to the *newName* string, and returns a value indicating success or failure.

Availability

5.0, 6.0

Parameter

newName	*String*	Name to which this method sets the File object's name property. This value must not specify a path.

Returns

Boolean	true indicates that ● The disk-based entity this *fileObj* represents was renamed successfully. ● The called *fileObj* has been updated to reflect the referenced file or folder's new location on disk.

seek

fileObj.seek (*pos*)

Set position as specified and return a value indicating success or failure.

Availability

5.0, 6.0

Parameter

pos	*Number*	The cursor's new position, expressed as the number of characters from that position back to the beginning of the file.

Returns

Boolean	true indicates that the position was changed to the *pos* position.

tell

fileObj.tell()

Returns the number of characters between the beginning of file and the current position.

Availability

5.0, 6.0

write

fileObj.write (*myText1*,*myText2*, ...)

Concatenates its comma-separated list of *myText* arguments to a single string and attempts to write this string to the *fileObj* file.

Availability

5.0, 6.0

Parameter

myTextN	*String*	Text string this method writes to the file.

Returns

Boolean	true indicates that the write operation succeeded.

Description

In GoLive 6, this method uses the encoder the *fileObj*.encoding property specifies.

writeln

fileObj.writeln (*myText1*,*myText2*, ...)

Concatenates its comma-separated list of *myText* arguments to a single string terminated by a linefeed character and attempts to write this string to the *fileObj* file.

Availability

5.0, 6.0

Parameter

myTextN	*String*	Text string this method writes to the file.

Returns

Boolean	true indicates that the write operation succeeded.

Description

In GoLive 6, this method uses the encoder the *fileObj*.encoding property specifies.

HTMLStyle Object

Availability

6.0

The `HTMLStyle` object encapsulates a single style in an HTMLStyleSet object. You can read its properties to retrieve style information, and you can write to them to set style information.

For example, to specify left-aligned text, set the following values:

```
name="div", attribute="align", value="left"
```

You can specify **bold** text as follows:

```
name="b", attribute="", value=""
```

Acquiring HTMLStyle Objects

The `htmlStyles` property of the Application object provides an `HTMLStyleCollection` object that returns all `HTMLStyleSet` objects currently available to the GoLive application. You can retrieve individual `HTMLStyleSet` objects from this collection by name, by attribute name, or by numeric index; subsequently, you can extract individual `HTMLStyle` objects from the `HTMLStyleSet` object in similar fashion.

```
app.htmlStyles[styleSetName].[styleName]
```

HTMLStyle Object Properties

Properties of the `HTMLStyle` object enable you to retrieve style objects and their attributes by name.

`name`	*string*	Name of the style (the element name)
`attribute`	*string*	Name of the style attribute.
`value`	*string*	Value of the attribute.

HTMLStyleSet Object

Availability

6.0

An `HTMLStyleSet` object represents a named styleset. This styleset may appear in the **Window>Styles** palette or it may be provided by a document.

Acquiring HTMLStyleSet Objects

You can retrieve an `HTMLStyleSet` object from the `htmlStyles` global collection, from the GoLive application, or from a `TextArea` object.

- The GoLive application object is available from the `app` global property. The `htmlStyles` property of the application object provides a collection of `HTMLStyleSet` objects that represent stylesets in the **Window>Styles** palette.

  ```
  app.htmlStyles[name]
  app.htmlStyles[n]
  ```

- A textArea Object manages a single block of HTML text in GoLive 6 Layout view. This object's `styleSet` property holds style information for that text block. A text block can stand on its own, or it can be the content of at table cell. Stylesets read from textArea objects don't appear in the Styles palette or the `app.htmlStyles` array unless installed there by the `addStyle` method.

  ```
  // getting style set from the markup tree
  markupObj.subElements["body"].textArea.styleSet
  markupObj.getSubElement["body"].textArea.styleSet
  // getting style set from the main text area of the active document
  document.mainTextArea.styleSet
  // getting style set from a table cell
  markupObj.layout.tableCell.textArea.styleSet
  ```

HTMLStyleSet Object Properties

`name`	*string*	Styleset name.
`paragraph`	*boolean*	true if this styleset used at the total paragraph.
`replace`	*boolean*	true if the style replace the currently existing style. (Paragraph styles replaces existing paragraph styles as well as inline styles replaces existing inline styles)
`style`	*HTMLStyle*	Array of HTMLStyle objects. Read-only.

HTMLStyleSet Object Functions

HTMLStyleSetObj.addStyle (*style*)
HTMLStyleSetObj.addStyle (*elementname*, *attributename*, *attributevalue*)
HTMLStyleSetObj.removeStyle (*indexNumber*)
HTMLStyleSetObj.removeStyle (*HTMLStyleObject*)
HTMLStyleSetObj.removeStyle (*elementname*, *attributename*, *attributevalue*)
HTMLStyleSetObj.setStyle (*indexNumber*)
HTMLStyleSetObj.setStyle (*newStyle*)
HTMLStyleSetObj.setStyle (*newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle (*newEltName*, *newAttrName*, *newAttrVal*, *newStyle*)
HTMLStyleSetObj.setStyle (*oldStyleIndex*, *newStyle*)
HTMLStyleSetObj.setStyle (*indexNumber*, *newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle (*oldStyle*, *newStyle*)
HTMLStyleSetObj.setStyle (*oldStyle*, *newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle (*oldEltName*, *oldAttrName*, *oldAttrVal*, *newStyle*)
HTMLStyleSetObj.setStyle (*oldEltName*, *oldAttrName*, *oldAttrVal*, *newEltName*, *newAttrName*, *newAttrVal*)

addStyle

HTMLStyleSetObj.addStyle(*style*)
HTMLStyleSetObj.addStyle(*elementname*, *attributename*, *attributevalue*)

Add the specified style to the styleset.

Availability

6.0

Parameters

style	*HTMLStyle*	Style to add.
elementname	*String*	Name of element that gets the new style.
attributename	*String*	Name of style attribute to add.
attributevalue	*String*	Value of style attribute to add.

Returns

Boolean	`true` indicates success.

Description

You can describe the style to add by passing an existing **HTMLStyle** object or by passing string arguments that describe it.

removeStyle

HTMLStyleSetObj.`removeStyle`(*indexNumber*)

HTMLStyleSetObj.`removeStyle`(*HTMLStyleObject*)

HTMLStyleSetObj.`removeStyle`(*elementname*, *attributename*, *attributevalue*)

Remove the specified style from the called `HTMLStyleSet` object.

Availability

6.0

Parameters

indexNumber	*Number*	Index number of style to remove. Index number 0 is the first style in the collection.
HTMLStyleObject	*HTMLStyle*	A HTMLStyle object that is the same as the one to remove.
elementname	*String*	Name of element from which to remove the style.
attributename	*String*	Name of style attribute to remove.
attributevalue	*String*	Value of style attribute to remove.

Returns

Boolean	`true` indicates success.

Description

You can specify the styleset to remove by numeric index into the styleset; by passing an existing **HTMLStyle** object; or by passing string arguments that describe it.

setStyle

HTMLStyleSetObj.setStyle(*indexNumber*)
HTMLStyleSetObj.setStyle(*newStyle*)
HTMLStyleSetObj.setStyle(*newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle(*newEltName*, *newAttrName*, *newAttrVal*, *newStyle*)
HTMLStyleSetObj.setStyle(*oldStyleIndex*, *newStyle*)
HTMLStyleSetObj.setStyle(*oldStyleIndex*, *newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle(*oldStyle*, *newStyle*)
HTMLStyleSetObj.setStyle(*oldStyle*, *newEltName*, *newAttrName*, *newAttrVal*)
HTMLStyleSetObj.setStyle(*oldEltName*, *oldAttrName*, *oldAttrVal*, *newStyle*)
HTMLStyleSetObj.setStyle(*oldEltName*, *oldAttrName*, *oldAttrVal*,
newEltName, *newAttrName*, *newAttrVal*)

Set attributes of the current style or those of a specified *HTMLStyleSetObj* object.

Availability

6.0

Parameters

indexNumber	*Number*	Index number of style to set. Index number 0 is the first style in the collection.
oldStyleIndex	*Number*	Index number of style to set. Index number 0 is the first style in the collection.
newStyle	*HTMLStyle*	The HTMLStyle object that provides the style to set.
oldStyle	*HTMLStyle*	The HTMLStyle object to replace.
newEltName	*String*	Name of style element to use as current style.
oldEltName	*String*	Name of style element to replace.
newAttrName	*String*	Name of style attribute to use in current style.
oldAttrName	*String*	Name of style attribute to replace.
newAttrVal	*String*	Value of style attribute to use in current style.
oldAttrVal	*String*	Value of style attribute to replace.

Returns

Boolean	`true` indicates success.

layout Object

Availability

6.0

The `layout` object enables you to manipulate box objects, grids, tables and other HTML elements in Layout view without reparsing. For descriptions of the HTML elements the `layout` object can manage, see Appendix B, "Supported Layout Tags." Elements not listed in this appendix cannot be managed by a Layout object; if your extension manipulates such objects with Layout View open, it must support automatic Layout View reparsing behavior.

For information about using this object, see "Using Layout Objects" on page 109 of the *Extend Script SDK Programmer's Guide*.

Acquiring Layout Objects

You can get this object from the `layout` property of the `Markup` object that manages the HTML content to manipulate.

markupObj`.layout`

Layout Object Properties

Properties of the `layout` Object provide access to the `LayoutGrid`, `GridLayout`, `TableLayout` and `TableCell` objects it manages.

`grid`	*LayoutGrid*	If the element is a layout grid or the element is a box positioned on a layout grid then this property contains a **LayoutGrid** object. Read-only.
`gridLayout`	*GridLayout*	If the element is a box positioned on a layout grid then this property contains a **GridLayout** object. Read-only.
`table`	*TableLayout*	If the element is an **table** element then this property contains an **TableLayout** object. Otherwise this property is **null**. Read-only.
`tableCell`	*TableCell*	If the element is an **td** or **th** element then this property contains a **TableCell** object, otherwise **null**. Read-only.

Layout Object Functions

Functions of the `layout` object enable you to get and set attribute values in HTML source it manages.

markupObj`.layout.getAttribute(`*attr*`)`
Returns the current value of *layoutObject.attr* as a string.

markupObj`.layout.setAttribute(`*attr* `[,` *value1*, ... , *valueN-1*, *valueN*`])`
Sets *layoutObject.attr* to the specified value(s).

markupObj`.layout.hasAttribute(`*attr*`)`
Returns `true` if *layoutObject* has an *attr* property.

For these methods, the value of *attr* is the name of a property listed in Appendix B, "Supported Layout Tags."

getAttribute

markupObj`.layout.getAttribute(`*attr*`)`

Returns the current value of *markupObj*`.layout.`*attr* as a string.

Availability

6.0

Parameters

arg	*String*	Name of the attribute to retrieve.

Returns

String	The current *markupObj*`.layout.`*attr* value.

setAttribute

markupObj.layout.setAttribute(*attr* [, *value1*, ... , *valueN-1*, *valueN*])

Sets the *markupObj*.layout.*attr* value; pass no value to remove *attr* from the *markupObj*.layout object.

Availability

6.0

Parameters

attr	*String*	Name of the attribute to set.
value	*String*	Value to which this method sets the *attr* attribute.The number of values to supply varies according to the attribute being set. If this argument is omitted, the method removes the attribute.

Returns

Boolean	true indicates success.

hasAttribute

markupObj.layout.hasAttribute(*attr*)

Returns true if *markupObj*.layout has an *attr* property.

Availability

6.0

Parameters

attr	*String*	Name of the attribute to test.

Returns

Boolean	true indicates that *attr* exists in the *markupObj*.layout object.

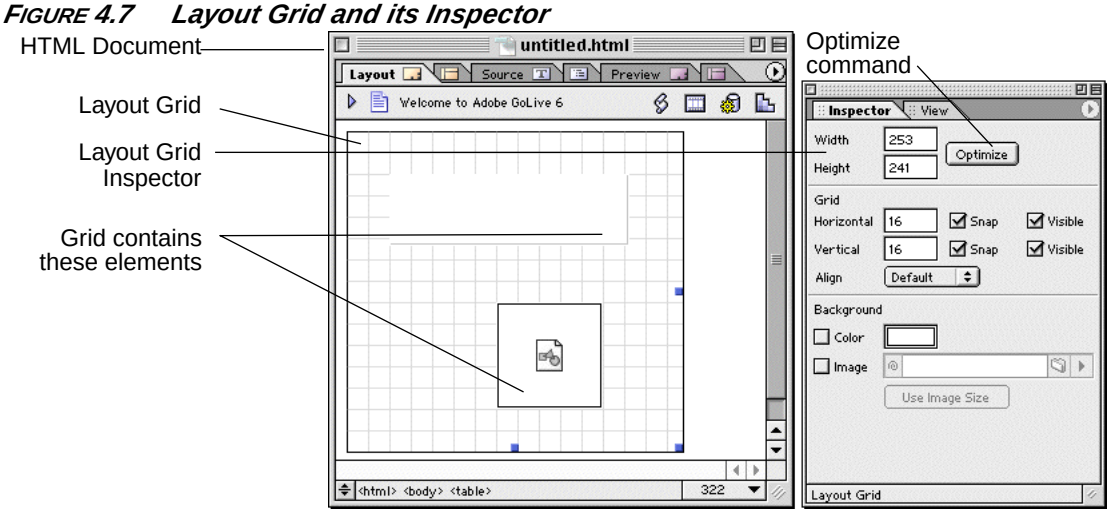
layout.grid Object

Availability

6.0

The layout.grid object represents a Layout Grid in Layout view, as Figure 4.7 shows. This object enables Extend Script extensions to edit and optimize Layout Grid objects just as the user can with the Layout Grid's Inspector window and its Optimize command.

FIGURE 4.7 Layout Grid and its Inspector



Acquiring a layout.grid Object

You can get a `LayoutGrid` object from the `grid` property of the `Layout` object that is a Layout grid or one of the objects the grid contains.

markupObj`.layout.grid`

layout.grid Object Properties

`align`	*String*	Alignment of the layout grid. Valid values are `left`, `center`, and `right`.
`background`	*String*	URL to the image to display as the layout's background.
	Boolean	Assigning `true` or `false` switches display of the background image on or off, respectively.
`bgcolor`	*String*	Name of the background color of the layout grid; example values include `"red"` or `"blue"`.
	Number	Background color of the layout grid, expressed as a hexadecimal value.
	Boolean	Assigning `true` or `false` switches display of the background color on or off, respectively
`height`	*Number*	Height of the layout grid.
`width`	*Number*	Width of the layout grid.

`layout.grid` object properties (*continued from preceding page*)

`xGrid`	*Number*	Horizontal distance of grid points.
`xGridSnap`	*Boolean*	If **true** the positions of boxes on the grid snap on the horizontal grid points.
`xGridVisible`	*Boolean*	If **true**, the vertical gridlines are visible.
`yGrid`	*Number*	Vertical distance of grid points.
`yGridSnap`	*Boolean*	If **true** the positions of boxes on the grid snap on the horizontal grid points.
`yGridVisible`	*Boolean*	Horizontal distance of grid points.

layout.grid Object Functions

markupObj`.layout.grid.optimize()`
Optimize the size of the layout grid for the dimensions of all boxes it contains.

markupObj`.layout.grid.setCursor (`*x*, *y*`)`
Sets the cursor at the given grid point.

markupObj`.layout.grid.cut()`
Cut out the current selected objects.

markupObj`.layout.grid.copy()`
Copies the current selected objects.

markupObj`.layout.grid.paste()`
Paste the current clipboard content at the current position.

markupObj`.layout.grid.insertBox(`*boxName*`)`
Inserts a box object at the current position.

markupObj`.layout.grid.insertHTML(`*htmlToInsert*`)`
Inserts the *htmlToInsert* HTML source at the current position.

markupObj`.layout.grid.`select(*markup*, *multipleSel*)
Sets the selection to *markup*; if *multipleSel* is `true`, adds *markup* to the selection.

markupObj`.layout.grid.`deselect(*all*)
markupObj`.layout.grid.deselect(`*element*`)`
markupObj`.layout.grid.deselect(`*element*, *all*`)`
Deselects one or all elements.

optimize

markupObj`.layout.grid.optimize()`

Optimize the size of the layout grid to the dimension of all containing boxes.

Availability

6.0

Returns

Boolean	`true` indicates success.

setCursor

markupObj`.layout.grid.setCursor(`*x*`, `*y*`)`

Sets the cursor at the given grid point.

Availability

6.0

Parameters

x	*Number*	Horizontal coordinate of the cursor's new location.
y	*Number*	Vertical coordinate of the cursor's new location.

Returns

Boolean	`true` indicates success.

cut

markupObj`.layout.grid.cut()`

Cut out the current selected objects.

Availability

6.0

Returns

Boolean	`true` indicates success.

copy

markupObj`.layout.grid.copy()`

Copies the current selected objects.

Availability

6.0

Returns

Boolean	`true` indicates success.

paste

markupObj`.layout.grid.paste()`

Paste the current clipboard content at the current position.

Availability

6.0

Returns

Boolean	`true` indicates success.

insertBox

markupObj`.layout.grid.insertBox(`*boxName*`)`

Adds a box object to Layout View at the current cursor position and sets its `name` property to the *boxName* value.

Availability

6.0

Parameter

boxName	*String*	JavaScript name of the box to insert.

Returns

Boolean	`true` indicates success.

Description

The *boxName* argument specifies the JavaScript name under which the new box appears in the GoLive environment. To insert a custom element created with the `<jsxelement>` tag, pass the value of its `classid` attribute. To insert an element the standard GoLive palettes provide, use one of the values Table 4.2 provides. This table also indicates the name that describes this element in the GoLive user interface, as well as the palette group in which similar elements appear.

TABLE 4.2 Valid values of boxName argument

JavaScript name	User Interface name	Palette group
anchor	Anchor	**Basic**
applet	Java Applet	
comment	Comment	
floatingbox	Floating Box	
image	Image	
layoutgrid	Layout grid	
layouttextbox	Layout Textbox	
linebreak	Line break	
line	Line	
marquee	Marquee	
plugin	Plug-in	
swf	SWF Plug-in	
quicktime	QuickTime Plug-in	
real	Real Plug-in	
svg	SVG Plug-in	
javascript	JavaScript	
spacer	Spacer	
table	Table	
w3c	W3C Object	
form	Form	**Forms**
submit	Submit button	
reset	Reset button	
button	Button	
inputimage	Input image	
label	Label	
textfield	Text field	
password	Password field	
textarea	Text area	

JavaScript name	User Interface name	Palette group
checkbox	Checkbox	
radiobutton	Radio button	
popup	Popup	
listbox	Listbox	
filebrowser	File browser	
hidden	Hidden	
keygenerator	Key generator	
fieldset	Field set	

insertHTML

markupObj.layout.grid.insertHTML(*htmlToInsert*)

Inserts a layout textbox at the current position and inserts the *htmlToInsert* HTML source into this box.

Availability

6.0

Parameters

htmlToInsert	*String*	HTML code to insert.

Returns

Boolean	`true` indicates success.

Description

You can use this method to add a new HTML element to a document without causing GoLive to reparse.

select

markupObj.layout.grid.select(*element*, *multipleSel*)

Makes the specified element the new selection or adds the specified element to the current selection.

Availability

6.0

Parameters

element	*Markup*	The element to select.
multipleSel	*Boolean*	true adds this element to the current selection. false makes this element the new selection.

Returns

Boolean	`true` indicates success.

Description

If *element* is a box positioned on the grid, then this box will selected. If *multipleSel* is `true` then the prior selected boxes on the grid are still selected.

deselect

markupObj.layout.grid.deselect(*all*)

markupObj.layout.grid.deselect(*element*)

markupObj.layout.grid.deselect(*element*, *all*)

Deselects one or all elements on the layout grid.

Availability

6.0

Parameters

all	*Boolean*	true deselects all currently-selected boxes on the layout grid.
element	*Markup*	A Markup object to deselect.

Returns

Boolean	`true` indicates success.

Description

If the *element* argument is a `Markup` element on the grid and it is selected, this method deselects it. Passing true as the first argument deselects all of this layout grid's elements.

layout.gridlayout Object

Availability

6.0

The `gridlayout` Object is a layout object that manages a layout grid's content, specifying precisely the position and alignment of that content in the layout grid's coordinate plane.

Acquiring GridLayout Objects

You can get the `gridlayout` object from the `gridlayout` property of the layout Object, as the following line of code does.

markupObj`.layout.gridlayout`

GridLayout Object Properties

`x`	*number*	X position on the grid.
`y`	*number*	Y position on the grid.
`width`	*number*	Width of the box.
`height`	*number*	Height of the box.
`halign`	*string*	Horizontal alignment on the grid. Values are **left**, **center**, **right**.
`valign`	*string*	Vertical alignment on the grid.alues are **top**, **center**, **bottom**.

GridLayout Object Functions

Functions of the `GridLayout` object change its size and position.

markupObj`.layout.gridlayout.setPosition` (*x*, *y*)
Sets the position of the `gridlayout` box.

markupObj`.layout.gridlayout.setSize` (*width*, *height*)
Sets the size of thethe `gridlayout` box.

setPosition

markupObj.layout.gridlayout.setPosition(*x*, *y*)

Sets the position on the grid.

Availability

6.0

Parameters

x	*Number*	Horizontal coordinate of the new location.
y	*Number*	Vertical coordinate of the new location.

Returns

Boolean	`true` indicates that the position was changed successfully.

setSize

markupObj.layout.gridlayout.setSize(*width*, *height*);

Sets the size of the gridlayout box.

Availability

6.0

Parameters

width	*Number*	New width of `gridlayout`, expressed as a number of pixels.
height	*Number*	New height of `gridlayout`, expressed as a number of pixels.

Returns

Boolean	`true` indicates that the size of `gridlayout` was changed successfully.

layout.table Object

Availability

6.0

The TableLayout object is a layout object that manipulates a <table> element programmatically; in addition to setting the size, number of rows/columns and so on, methods of this object add, delete, split, or merge rows, columns, or cells.

Acquiring TableLayout Objects

The TableLayout object is available only when the document that provides the *markupObj* Markup Object is displaying Layout view. You can retrieve the `TableLayout` object from the `layout.table` property of the *markupObj* that manages the `<table>` element source to manipulate.

markupObj`.layout.table`

TableLayout Object Properties

`rows`	*number*	Number of rows. You can set the value of this property to change the number of rows in the table.
`columns`	*number*	Number of columns. You can set the value of this property to change the number of columns in the table.
`cell`	*TableCellLayoutCollection*	Array of **TableCellLayout** objects. The index is count from left to right beginning with 0. Read-only.
`style`	*TableStyle*	The current **TableStyle** object. Read-only.
`cellStyle`	*TableCellStyleCollection*	All the TableCellStyle objects this TableLayout object uses. Cells are indexed from left to right beginning at 0. Read-only.
`width`	*number*	Total width of the table.
`height`	*number*	Total height of the table.
`border`	*number*	Border width.
`cellpadding`	*number*	Cell padding of the table.
`cellspacing`	*number*	Cell spacing of the table.
`bgolor`	*string colorname* *or* *number colorvalue* *or* *boolean onoff*	The background color.
`background`	*string url* *or* *boolean onoff*	The background image.
`align`	*string*	The alignment of the table. Values are: **left**, **right** and **center**.
`captionPosition`	*string*	The position of the caption. Values are: **none**, **top** and **bottom**.
`captionCell`	*TableCell*	The caption cell.Read-only.

TableLayout Object Functions

insertRowAt

markupObj`.layout.table.insertRowAt(`*pos*`)`

Inserts a row before the *pos* row.

Availability

6.0

Parameters

pos	*Number*	Row at which to insert the new row. Row `0` is the first row in the table.

Returns

boolean	`true` indicates success.

insertColumnAt

markupObj`.layout.table.insertColumnAt(`*pos*`)`

Inserts a column before the *pos* column.

Availability

6.0

Parameters

pos	*Number*	Column at which to insert the new column. Column `0` is the first column in the table.

Returns

boolean	`true` indicates success.

removeRowAt

markupObj.layout.table.removeRowAt(*pos*)

Removes the *pos* row.

Availability

6.0

Parameters

pos	*Number*	Row to remove. Row `0` is the first row in the table.

Returns

boolean	`true` indicates success.

removeColumnAt

markupObj.layout.table.removeColumnAt(*pos*)

Removes the *pos* column.

Availability

6.0

Parameters

pos	*Number*	Column to remove. Column `0` is the first column in the table.

Returns

boolean	`true` indicates success.

mergeCells

markupObj`.layout.table.mergeCells(`*left*`, `*top*`, `*right*`, `*bottom*`)`

Merge all cells which lie inside the specified rectangle.

Availability

6.0

Parameters

left	*Number*	X-coordinate of the upper-left corner of the rectangle.
top	*Number*	Y-coordinate of the upper-left corner of the rectangle.
right	*Number*	X-coordinate of the lower-right corner of the rectangle.
bottom	*Number*	Y-coordinate of the lower-right corner of the rectangle.

Returns

boolean	`true` indicates success.

splitCells

markupObj`.layout.table.splitCells(`*x*`, `*y*`)`

Separate any merged cells that intersect the specified coordinates.

Availability

6.0

Parameters

x	*Number*	X-coordinate of the point this method tests.
y	*Number*	Y-coordinate of the point this method tests.

Returns

boolean	`true` indicates success.

addCurrentStyle

markupObj`.layout.table.addCurrentStyle(`*name*`)`

Adds the current tablestyle to the global `tableStyles` array under the specified JavaScript *name*.

Availability

6.0

Parameters

name	*String*	JavaScript name under which to add the tablestyle.

Returns

boolean	`true` indicates success.

applyStyle

markupObj`.layout.table.applyStyle(`*tableStyleObj*`)`

Applies the *tableStyleObj* table style to this table.'

Availability

6.0

Parameters

tableStyleObj	*TableStyle*	Table style this method applies to the called table layout object

Returns

boolean	`true` indicates success.

importCellText

markupObj.layout.table.importCellText(*filePath*[, *separator*])

Import tab-delimited text from *filePath* into cells of the table; optionally, import a file that uses a *separator* other than the tab character.

Availability

6.0

Parameters

filePath	*String*	Relative or fully-qualified path to the file to import.
separator	*String*	Optional. Pass this value to designate a character other than tab to be interpreted as the record-separator character.

Returns

boolean	`true` indicates success.

exportCellText

markupObj.layout.table.exportCellText(*filePath*)

Export the cells of the `table` as a tab-delimited text file located at *filePath*.

Availability

6.0

Parameters

filePath	*String*	Name and location of this method's output file, specified as a relative or fully-qualified path or URL.

Returns

boolean	`true` indicates success.

getCellAt

markupObj.layout.table.getCellAt(*row*, *col*)

Retrieves the (*row*, *col*) table cell from the table.

Availability

6.0

Parameters

row	*Number*	Row number of the cell to retrieve. Row 0 is the first row.
col	*Number*	Column number of the cell to retrieve. Column 0 is the first column.

Returns

TableCellLayout	The (*row*, *col*) table cell.

getCellStyleAt

markupObj.layout.table.getCellStyleAt(*row*, *col*)

Retrieves the (*row*, *col*) table cell style from the table.

Availability

6.0

Parameters

row	*Number*	Row number of the cell from which to retrieve style information. Row 0 is the first row.
col	*Number*	Column number of the cell from which to retrieve style information. Column 0 is the first column.

Returns

TableCellStyle	Table cell style this method retrieves from the called table cell layout object

setCellStyleAt

markupObj.layout.table.setCellStyleAt(*row*, *col*, *cellStyle*)

Set the cell style of the table's (*row*, *col)* cell to *cellStyle*.

Availability

6.0

Parameters

row	*Number*	Row number of the cell from which to retrieve style information. Row 0 is the first row.
col	*Number*	Column number of the cell from which to retrieve style information. Column 0 is the first column.
tableCellStyleObj	*TableCellStyle*	Table cell style to which this method sets the called table layout object's (*row*, *col*) cell.

Returns

boolean	`true` indicates success.

layout.table.style Object

Availability

6.0

A TableStyle object encapsulates a <table> element's style information. This object provides access to the table's name, border width, cell spacing/padding, number of header rows, number of footer rows, and so on.

Acquiring TableStyle Objects

You can get a TableStyle object from

- The `style` property of the layout.table Object

  markupObj.layout.table.style

- The collection in the `tableStyles` property of the app Object.

  app.tableStyles[*tableStyleName*]

  app.tableStyles[*n*]

TableStyle Object Properties

`name`	*string*	Style name.
`border`	*number* *boolean*	Table border width in pixels. Set to **false** to *switch off* the value in the style. I.e. The value will not use if you assign a **false** irrespective of the current value.
`cellspacing`	*number* *boolean*	Table cell spacing in pixels. Set to **false** to *switch off* the value in the style. I.e. The value will not use if you assign a **false** irrespective of the current value.
`cellpadding`	*number* *boolean*	Table cell padding in pixels. Set to **false** to *switch off* the value in the style. I.e. The value will not use if you assign a **false** irrespective of the current value.
`xSize`	*number*	Number of columns the style affects.
`ySize`	*number*	Number of rows the style affects.
`numHeaderRows`	*number*	Number of header rows. If this style is used for a table, than the first *n* rows of the style are used once for the table and never again. All other rows of the style repeats if the real table is bigger than the style.
`numFooterRows`	*number*	Number of footer rows. If this style is used for a table, than the last *n* rows of the style are used once for the table and never again. All other rows of the style repeats if the real table is bigger than the style.
`numHeaderColumns`	*number*	Number of header columns. If this style is used for a table, than the first *n* columns of the style are used once for the table and never again. All other columns of the style repeats if the real table is bigger than the style.
`numFooterColumns`	*number*	Number of footer columns. If this style is used for a table, than the last *n* columns of the style are used once for the table and never again. All other columns of the style repeats if the real table is bigger than the style.

TableStyle Object Functions

getCellStyleAt

tableStyleObj`.getCellStyleAt(`*row*`,` *col*`)`

Retrieves from the *tableStyleObj* the cell style for the (*row*, *col*) cell.

Availability

6.0

Parameters

row	*Number*	Row number of the cell from which to retrieve style information. Row 0 is the first row.
col	*Number*	Column number of the cell from which to retrieve style information. Column 0 is the first column.

Returns

TableCellStyle	Table cell style this method retrieves from the called table cell layout object

setCellStyleAt

tableStyleObj`.setCellStyleAt(`*row*`,` *col*`,` *cellStyle*`)`

Set the cell style of the called TableStyle object's (*row*, *col)* cell to *cellStyle*.

Availability

6.0

Parameters

row	*Number*	Row number of the cell from which to retrieve style information. Row 0 is the first row.
col	*Number*	Column number of the cell from which to retrieve style information. Column 0 is the first column.
tableCellStyleObj	*TableCellStyle*	Table cell style to which this method sets the called table style object's (*row*, *col*) cell.

Returns

boolean	`true` indicates success.

layout.tableCell Object

Availability

6.0

A **TableCellLayout** object manages the content of a cell in a TableLayout object. It enables you to manipulate the size and style of the cell itself and to get the TextArea object that holds the cell's text content.

Acquiring TableCellLayout Objects

This object is available only when the document that holds the table is displaying Layout view. Layout view provides this object as

- The `tableCell` property of the table cell's markup object's layout object.

  markupObj`.layout.tableCell`

- The return value of the `getCellAt` function of the layout.table Object.

  markupObj`.layout.table.getCellAt(`*row*`, `*col*`)`

- The `captionCell` property of the TableLayout object.

  markupObj`.layout.table.captionCell`

TableCellLayout Object Properties

`background`	*string url*	The background image.
`bgolor`	*string colorstring* or *number colorvalue*	The background color.
`halign`	*string*	Horizontal alignment. Values are **left**, **center**, **right**.
`height`	*number*	Height of the cell.
`nowrap`	*boolean*	Switch on/off **nowrap**.
`style`	*TableCellStyle*	The style of the cell.
`textArea`	*TextArea*	The textbox object of the cell.
`valign`	*string*	Vertical alignment.alues are **top**, **center**, **bottom**.
`width`	*number*	Width of the cell.

TableCellLayout Object Functions

The TableCellLayout object provides only one method, the `applyStyle` method, which applies a table cell style to a TableCellLayout object.

applyStyle

markupObj`.layout.tableCell.applyStyle(`*tableCellStyleObj*`)`

Apply the specified table cell style to the called table cell layout object.

Availability

6.0

Parameters

tableCellStyleObj	*TableCellStyle*	Table cell style this method applies to the called table cell layout object

Returns

boolean	`true` indicates success.

layout.tableCell.style Object

Availability

6.0

A **TableCellStyle** object encapsulates a cell's style information. It provides information such as background color, text color, font face, and so on.

In the **TableStyle** object, each table cell's style information is represented as a **TableCellStyle** object. Properties of the TableCellStyle object affect the cells of the **TableStyle** object.

Acquiring TableCellStyle Objects

TableCellLayout object's `style` property

markupObj`.layout.tableCell.style`

Return value of `getCellStyleAt` method of TableLayout object

markupObj`.layout.table.getCellStyleAt(`*row*`,` *col*`)`

Return value of `getCellStyleAt` method of TableStyle object

TableCellStyle Object Properties

Assign **true** or **false** to *switch on/off* use of the value in the style; that is, the style's value is used if you set the property's value to true. If you set the property to `false`, GoLive does not use the style's value, regardless of the property's current value.

`bgcolor`	*number Color* *or* *string Colorname* *or* *boolean useValue*	Background color of the cell. The value of this attribute is a colorname or a value in the format **#rrggbb** (**r** – red value, **g –** green value, **b** – blue value). Assign **true** or **false** to *switch on/off* use of the value in the style.
`textColor`	*number Color* *or* *string Colorname* *or* *boolean useValue*	Textcolor of the cell. The value of this attribute is a colorname or a value in the format **#rrggbb** (**r –** red value, **g –** green value, **b** – blue value). Assign **true** or **false** to *switch on/off* the value in the style.
`header`	*boolean Headercell*	Switch between normal cell and header cell.
`nowrap`	*boolean NoWrap*	Switch on/off nowrap
`fontFace`	*string FontFace* *or* *boolean useValue*	The fontface of the cell. The Table Cell Style Object supports **plain**, **bold**, **italic**, **underline** and **strikeout**. This property could contain one or more of this styles as a comma separated list. Assign **true** or **false** to *switch on/off* the value in the style.
`fontName`	*string FontName*	Contains the font names of the fontset to use.
`fontSize`	*string FontSize* *or* *boolean useValue*	Contains the size of the font. The size could be a simple numeric value or a value in the form of **+2**,**-3**,etc. Assign **true** or **false** to *switch on/off* the value in the style.
`halign`	*string halign* *or* *boolean useValue*	Contains the horizontal alignment. Values are **left**, **center**, **right** and **default**. Assign **true** or **false** to *switch on/off* the value in the style.
`valign`	*string valign* *or* *boolean useValue*	Contains the vertical alignment. Values are **top**, **center**, **bottom** and **default**. Assign **true** or **false** to *switch on/off* the value in the style.

Link Object

Availability

5.0, 6.0

A link object encapsulates an active URL. GoLive creates a Link object to represent each URL it encounters when parsing a document's source representation. Use the Link object to initialize a control of type `urlgetter` in an **Inspector** palette.

Acquiring Link Objects

GoLive passes a link object to your linkChanged function when a link changes.

```
linkChanged(link)
```

When GoLive creates a box object to represent an HTML element that provides a URL, the link object that encapsulates the URL is in the link property of the box.

```
box.link
```

To add a link to an HTML element represented by a box, use the box object's createLink method.

```
box.createLink(url)
```

Link Object Properties

Properties of the link object provide a URL reference and information about the referenced entity, such as whether it resides on a local disk, and transport protocols to use if it does not.

`local`	*Boolean*	`true` when this link points to a file residing on a local disk; `false` otherwise. Read-only.
`longUrl`	*String*	The absolute URL to the referenced file on disk. Read-only.
`mimeType`	*String*	The MIME type of the file associated with this link. Read-only.
`protocol`	*String*	The transfer protocol used to upload or download the file associated with this link. Read-only.
`url`	*String*	The URL that this link object encapsulates. You can set or read this property.

Link Object Functions

A Link object's most important behaviors are provided by the `linkChanged` event-handling function that the Extend Script SDK calls when the user interacts with a link.

NOTE: To inspect a link object, your extension must provide the `linkChanged` global event-handling method that the Links section of Chapter 6, "Event-Handling Functions," describes. The SDK calls this event-handling function in response to user interactions with a link object.

The Link object itself provides only the `drawIcon` convenience function, which draws the screen icon that reflects the link's status. A custom element that adds a link to the page can use this function to draw the link's icon in the custom element's graphical placeholder in **Layout** view. To do so, its `drawBox` method calls `box.link.drawIcon()` after drawing the rest of the custom icon's placeholder. When the `drawIcon` method draws the link's icon on top of the custom placeholder graphic, it looks as if it's part of the graphic.

drawIcon

linkObj`.drawIcon(`*x*`, `*y*`[, `*width*`, `*height*`])`

Associated with each link object is an icon that reflects its current state.Call this method to draw the link's state icon at the specified location; optionally, resizes the icon as well.

Availability

5.0, 6.0

Parameters

`x`	*Number*	X-coordinate of the upper-left corner of the smallest rectangle that would enclose the icon to draw.
`y`	*Number*	Y-coordinate of the upper-left corner of the smallest rectangle that would enclose the icon to draw.
`width`	*Number*	The icon's width, expressed as a number of pixels.
`height`	*Number*	The icon's height, expressed as a number of pixels.

Markup Object

Availability

5.0, 6.0

The Markup object represents an individual markup element. GoLive generates a tree of markup elements when it interprets a document as HTML.The SDK represents these elements to JavaScript callers as markup objects. The markup objects define their own binary tree structure which mimics the element structure defined in the source HTML.

An individual markup object provides programmatic access to the text block that holds its source text representation. This text item is a markup element, a text block, or a comment. You can call the markup object's methods to modify, replace, or delete the text that represents the element in the source document.

Properties of the markup object provide references you can use to navigate the markup tree programmatically or to retrieve specified elements from the markup tree.

Acquiring Markup Objects

When a document object represents a markup document (not a site file or a text file), its `element` property holds the markup object that represents the `GoLiveMarkup` element. This special markup object is reserved for use by GoLive. It represents the root of a binary tree of markup objects that represent the content of the document.

```
var tree = document.element();
```

When this code returns, the `tree` variable holds a Markup object representing the `GoLiveMarkup` element that is the root of the markup tree. The page's content takes the form of one or more subelements of this Markup object.

As the root of this tree, the `GoLiveMarkup` element has no parent, but all other markup objects provide a `parent` property which enables access to the represented element's immediate ancestor in the markup tree. All markup objects have a `subElements` property, which enables access to all of their descendents; thus, you can use the `parent` and `subElements` properties to navigate the markup tree.

To retrieve a specified subelement by tag name or by numeric index, call the `getSubElement` method. This method can retrieve any subelement of the called markup object, not just immediate subelements.

A page's visible elements are subelements of its `body` element. You can use the `getSubElement` method of the markup object to retrieve a page's `<body>` subelement by tag name, as in the following example.

```
// retrieve current page's <body> element
var bodyElt = document.element.getSubElement("body");
```

Retrieving Markup Objects By Type

GoLive creates different kinds of markup elements to manage different kinds of content; for example, an element might be a simple HTML element (type `plain`); a container element (various types, such as `container`, `binary`, `SSI`, or others); or a comment. In GoLive 6.0, the markup object's `getSubElement` and `getSubElementCount` methods can retrieve markup elements by type. For details, see the reference listings for these methods.

Markup Object Properties

Properties of the markup object provide information about the element it represents, such as its tag name and whether it is a markup element, a comment, or a text block. This object's properties also provide access to the referenced element's attributes, and access to its textual representation.

Like all JavaScript objects, this object implements its attributes as an array. Additionally, this object's `subElements` property holds an array of all of its subelements. You can use tag names as search keys to retrieve elements from this array by name; for example, you could retrieve all of a page's `<IMG>` elements in this way.

The attributes in an element's source representation defines become the properties of the object GoLive creates to represent that element.

- Setting a property writes the corresponding attribute value to the element's HTML source representation.
- Reading a property returns the property value or undefined if the attribute does not exist.
- You cannot delete a property from a JavaScript object; however, you can delete from the element's source representation the attribute which defines that property and then regenerate the element without the deleted property.To do so, use methods of the markup object to edit the element's source representation, then reparse the document to generate a markup element from the edited source representation. The new markup object does not have a property representing the deleted attribute. For more information, see Chapter 4, "Custom Elements," in the *Extend Script SDK Programmer's Guide*.

`elementType`	*String*	The type of text this markup object represents. The `"tag"` value indicates that this object represents the source text of an element of the markup tree. The `"comment"` value indicates that this object represents a source code comment. The `"text"` value indicates that this object represents non-HTML text.
`layout`	*Layout*	This *markupObj* manages an HTML element listed in Appendix B, "Supported Layout Tags," and the document that supplies *markupObj* is displaying Layout view.
	null	Any of the following error conditions may apply: This *markupObj* manages an HTML element not supported by the Layout object and not listed in Appendix B, "Supported Layout Tags." The document that supplies *markupObj* is not displaying Layout view.

Markup object properties (*continued from preceding page*)

`subElements`	*Collection*	Read-only array of Markup objects representing the subelements of this Markup object's element. You can retrieve Markup objects from this array by `tagName` values or by numeric index. If this array contains more than one element of the same name, it cannot be predicted which element is returned when selecting by name. **NOTE:** When two or more objects in a collection have the same `name` value, those objects cannot be retrieved by name reliably.
`symmetric`	*Boolean*	true indicates that the end of the tag matches the beginning of the tag, as in the "<%xxx%>" tag.
`tagName`	*String*	The tag name of the element the SDK interpreted to create this markup object; for example, if this value is `"myTag"`, this markup object was created when the SDK interpreted a `<myTag>` element.
`tagStart`	*String*	The tag delimiter of the tag used to define the element this markup object represents. Typical values for this property are "" (empty string),"<", "[", "percent", and "<%".
`textArea`	*TextArea*	The document that supplies *markupObj* is displaying Layout view, and *markupObj* manages an HTML text element or a box object embedded in a text area.
	null	Any of the following error conditions may apply: This *markupObj* does not manage an HTML text element or a box object embedded in a text area. The document that supplies *markupObj* is not displaying Layout view.
`virtual`	*Boolean*	`true` indicates an element which, by design, cannot be modified programmatically. The Layout view of a document may create certain markup elements even if they are not part of the source code; by default, Layout view creates <HTML>, <HEAD>, and <BODY> elements whenever it opens a new HTML document. These elements are not present in the document's source text, so the setInnerHTML() and setOuterHTML() methods cannot modify them.

Markup Object Functions

Markup Object functions modify, replace, or delete the markup element's text representation in the text buffer of the markup object GoLive created when it interpreted the text as HTML.

NOTE: The SDK is intended for use in **Layout view** of GoLive 5 or the split Layout/Source view of GoLive 6. Attempts to modify HTML source programmatically in other views, such as **Source view**, can yield incorrect results. To get or set the user interface view a document window presents, use the `view` property of the document Object.

The markup tree always reflects structural changes immediately; for example, when you add or delete markup objects, the markup tree reflects the new structure immediately. Other changes may not be reflected in the markup tree's elements until the document is reparsed. If the markup object's text buffer has changed since the markup object was created, JavaScript calls operate on the contents of the text buffer as if the source document contained that text. In this way, JavaScript callers and Inspectors realize the results of a call that changes the source representation without actually having to reparse the document.

In order to realize all of the changes as HTML, however, you must reparse the entire document. GoLive does not write the contents of the markup object's text buffer to the document until the document is reparsed. The contents of the document object, in turn, are not written to the disk file the document represents until the document is saved.

If GoLive displays the document in Layout view, numerous situations may cause GoLive to reparse the document automatically at any time. Every time GoLive reparses the document, it calls your extension's `parseBox` method. This method must reinitialize all of your extension's references to JavaScript objects, because reparsing replaces the data structures that represent the markup tree with new ones.

If you've used the `openMarkup` method to access the document's markup tree, GoLive does not display the document, so Layout view cannot cause your extension to reparse the document automatically. Regardless of how you've opened a document, you can reparse it on demand by calling its `reparse` method.

Simply closing a document and saving changes also has the effect of reparsing it.

Table 4.3 illustrates the manner in which the `setInnerHTML` and `setOuterHTML` functions discern "inner" and "outer" HTML according to whether a unary or binary tag defines the element the called Markup object represents.

TABLE 4.3 Inner and outer HTML examples

	Unary	Binary	Binary (no optional end tag)
Example	`<img src="Images/ducky.jpg">`	`<H1>Hello</H1>`	`<LI>first` `<LI>second`
Inner HTML	`img src="Images/ducky.jpg"`	`Hello`	`first`
Outer HTML	`<img src="Images/ducky.jpg">`	`<H1>Hello</H1>`	`<LI>first`

Attribute Accessor Methods

In GoLive 6.0, the getAttribute, setAttribute, and deleteAttribute methods of the Markup object provide an alternative to accessing attribute values by property name. These methods are especially useful for accessing attributes that have names that conflict with JavaScript naming conventions.

Specifying the encoding that get/setinner/outerHTML methods use

In GoLive 6, the getOuterHTML, getInnerHTML, setOuterHTML, and setInnerHTML methods accept an optional *encoding* argument that is the name of an encoding character set. When this argument is provided, the encoding it specifies overrides either the encoding for the document (defined in the META element) or the default encoding of the document. This feature makes it possible to extract elements that may have been encoded in a different encoding than the other parts of a document.

A typical scenario is a user creating a document in ISO-2022-JP. If the user comments out one paragraph of text then changes the document encoding to X-SJIS, the change does not affect comments, leaving the commented out paragraph in its initial encoding. Using these new markup object functions, an extension can scan all comments in a document and attempt to convert them to the same encoding the rest of the document uses.

If you specify an unknown or unsupported encoding, GoLive generates an "Unsupported encoding" runtime error.

canSetHTML

markupObj.`canSetHTML ([`*outer*`])`

Returns `true` when the `setInnerHTML` method can be called safely without adverse side effects.

Availability

5.0, 6.0

Parameter

outer	*Boolean*	Pass `true` to test ability to call `setOuterHTML` method safely.

Returns

`Boolean`

Description

Returns `true` when the `setInnerHTML` method can be called safely without adverse side effects. When the value of the *outer* parameter is `true`, this method tests the ability to call the `setOuterHTML` method without adverse side effects.

deleteAttribute

markupObj.`deleteAttribute (`*name*`)`

Deletes the *name* attribute from the markup element the *markupObj* manages.

Availability

5.0, 6.0

Parameter

name	*String*	Name of the attribute to delete.

Returns

Boolean	`true` indicates success.

Description

This method is especially useful for deleting attributes that have names that conflict with JavaScript naming conventions.

getAttribute

markupObj.`getAttribute (`*name*`)`

Returns a String representation of the value of the *name* attribute of the markup element the *markupObj* manages.

Availability

5.0, 6.0

Parameter

name	*String*	Name of the attribute to retrieve.

Returns

String	String representation of the value of the attribute actually retrieved.

Description

This method is especially useful for accessing attributes that have names that conflict with JavaScript naming conventions.

getInnerHTML

markupObj.`getInnerHTML()`

Retrieves the *markupObj* element's HTML representation, excluding the outermost tag delimiters.

Availability

5.0, 6.0

Returns

String	Inner HTML text of the markup element *markupObj* manages.

Description

Retrieves the *markupObj* element's HTML representation, excluding the outermost tag delimiters. See Table 4.3 on page 147 for examples.

getOuterHTML

markupObj.`getOuterHTML()`

Retrieves the *markupObj* element's HTML representation, including the outermost tag delimiters.

Availability

5.0, 6.0

Returns

`String`

Description

Retrieves the *markupObj* element's HTML representation, including the outermost tag delimiters. See Table 4.3 on page 147 for examples.

getSubElement

markupObj.`getSubElement ([`*tagName*`,` *index*`,` *tagType*`])`

Retrieves a subelement of the *markupObj* element by name, index, or type.

Availability

5.0, 6.0

Parameters

tagName	*String*	Tag name of the element for which this method searches. This argument is case-insensitive; for example, specifying body as this value retrieves the body element regardless of whether is defined with a <BODY> tag or a <body> tag.

`markup.getSubElement()` parameters (*continued from preceding page*)

index	*Number*	Optional. Number of <*tagName*> elements to skip before returning one; that is, an index value of 0 retrieves the first <*tagName*> element found.
tagtype	*String*	Optional. Type of tag to retrieve. GoLive 6 only. Valid values are:

`plain`	simple tag
`binary`	binary tag
`container`	container tag
`/`	end tag
!	SGML tag
`bracket`	Lasso tag [xxx]
`percent`	tag%%
`ssi`	Server Side Includes
GoLive 6 Only	
`!--`	comment tags (not for `<jsxelement>`, only for `getSubElement()`)
`asp` or `'%'`	ASP tags <% xxxx %>
`websiphon` or `<<`	WebSiphon tags << xxx >>
`?`	Processing instructions/JSP

Description

All arguments are optional. When called with no argument, this method returns the document's `<HTML>` element. When the optional *n* argument is supplied, this method returns the markup object representing the *n*th occurrence of the <*tagName*> element. In GoLive 6.0, this method can retrieve markup elements by type.

The *tagtype* argument specifies the kind of markup element to retrieve; its value corresponds to the value of the `type` property of the markup object GoLive creates and inserts in the DOM when it interprets a markup element.You can also think of the valid set of *tagType* values as those which correspond to valid values of the `type` attribute of the `<jsxelement>` tag.The *tagType* values `"plain"`, `"binary"`, and `"container"` retrieve HTML or XML elements.

These methods ignore the tagname actually used to mark text as a comment. The following line of code retrieves HTML and XML comments.

```
markup.getSubElement ("", n, "!—");
```

Returns

`Markup`

getSubElementCount

markupObj.`getSubElementCount ([`*tagName*`])`

markupObj.`getSubElementCount ([`*tagName*`,` *tagType*`]) //GoLive 6 only`

Returns the count of *tagName* elements found among this element's subelements.

Availability

5.0, 6.0

Parameters

tagName	*String*	The tag name for which this method searches. For example, pass `"img"` to count the number of `<img>` subelements of the element the called markup object represents.
index	*Number*	Optional. Number of *<tagName>* elements to skip before returning one; that is, an index value of 0 retrieves the first *<tagName>* element found.
tagType	*String*	Optional. Type of tag to retrieve. GoLive 6 only. Valid values are:

`plain`	simple tag
`binary`	binary tag
`container`	container tag
`/`	end tag
!	SGML tag
`bracket`	Lasso tag [xxx]
`percent`	tag%%
`ssi`	Server Side Includes
`!--`	comment tags
`asp` or `'%'`	ASP tags <% xxxx %>
`websiphon` or <<	WebSiphon tags << xxx >>
`?`	Processing instructions/JSP

NOTE: Versions of the SDK prior to SDK 6.0r1.0 do not support the *tagtype* argument to the `getSubElement` and `getSubElementCount` methods.

Description

All arguments are optional. Omit the `tagName` argument to count all subelements. In GoLive 6.0, this method can count markup elements of a specified type.

The *tagType* argument specifies the kind of markup element to count; its value corresponds to the value of the `type` property of the Markup object GoLive creates and inserts in the DOM when it interprets a markup element.You can also think of the valid set of *tagType* values as

those which correspond to valid values of the `type` attribute of the `<jsxelement>` tag.The *tagType* values `"plain"`, `"binary"`, and `"container"` count HTML or XML elements.

This method ignores the tagname actually used to mark text as a comment. The following line of code counts HTML or XML comments.

```
markup.getSubElementCount ("", "!–");
```

Returns

`Number`

setAttribute

markupObj`.setAttribute (`*name*`,` *value*`);`

Set to *value* the *name* attribute of the HTML element that *markupObj* manages, creating the attribute if necessary.

Availability

6.0

Parameter

name	*String*	Name of the attribute this method creates in the *markupObj* object.
value	*Object*	Value of the attribute this method creates.

Returns

Boolean	`true` indicates success.

Description

This method is especially useful for setting attributes that have names that conflict with JavaScript naming conventions.

Example

```
//assume an html document holding an image is open in Layout view
var img = document.element.getSubElement("img", i);
img.layout.setAttribute("src", "/picture1.gif");
```

setInnerHTML

markupObj.`setInnerHTML (`*text*`)`

Replace the HTML representation of the *markupObj* element without altering its surrounding tag delimiters.

Availability

5.0, 6.0

Parameter

text	*String*	Replacement text.

Description

Replace the HTML representation of the *markupObj* element without altering its surrounding tag delimiters.

For important information about using this method correctly, see "Manipulating HTML Programmatically" on page 126 of the *Extend Script SDK Programmer's Guide*.

IMPORTANT: *Do not call this method from within the* `parseBox` *method or the* `undoSignal` *method. Do not call this method from within the* `controlSignal` *method if the control that invoked* `controlSignal()` *is displayed by an inspector window.*

setOuterHTML

markupObj.`setOuterHTML (`*text*`)`

Replace the HTML representation of the *markupObj* element, including its outermost tag delimiters.

Availability

5.0, 6.0

Parameter

text	*String*	Replacement text.

Description

Replace the HTML representation of the *markupObj* element, including its outermost tag delimiters.

For important information about using this method correctly, see "Manipulating HTML Programmatically" on page 126 of the *Extend Script SDK Programmer's Guide*.

IMPORTANT: *Do not call this method from within the* `parseBox` *method or the* `undoSignal` *method. Do not call this method from within the* `controlSignal` *method if the control that invoked* `controlSignal()` *is displayed by an inspector window.*

split

markupObj.split ([*sep*])

Returns this *markupObj* element's inner HTML as an array of strings.

Availability

5.0, 6.0

Parameter

sep	*String*	Optional. Character to interpret as separator.

Returns

Array

Description

Returns this *markupObj* element's inner HTML as an array of strings. Optionally, this method uses the sep character as a separator; if this argument is not supplied, this method interprets spaces as separators. This method does not split quoted strings containing the separator character. This method is useful for checking the contents of custom tags.

Menu Object

Availability

5.0, 6.0

The SDK creates a menu object when GoLive interprets a <jsxmenu> element containing one or more <jsxitem> elements.

```
<jsxmenubar> // opens definition of all menus
   <jsxmenu name="ADBEHello" title="Hello, GoLive!"> // Hello menu
      <jsxitem name="ADBEThis" title="Do Something"> // menu item
      <jsxitem name="ADBEThat" title="Do Something Else" > menu item
   </jsxmenu> // closes definition of Hello menu
</jsxmenubar> // closes definition of all menus
```

Each <jsxitem> element defines an individual menu item. By default, GoLive enables menus and menu items when it creates them; however, you can enable or disable a menu or any menu item programmatically.

When a <jsxitem> element provides the dynamic attribute, GoLive enables or disables its corresponding menu item dynamically. Every time GoLive is about to display the menu, it calls the menuSetup method of each menu item that provides the dynamic attribute. Your implementation of the menuSetup method sets either or both of the enabled and checked states of the menu item GoLive passed to it. GoLive then displays the menu and its items as specified by the values of these properties.

When the user chooses a menu item, GoLive calls the menuSignal method of the extension that provided the item, passing the selected menu item as its argument. Your implementation of the menuSignal method performs the tasks associated with the menu item passed to it.

Access to Menu Objects

Much of a menu's functionality resides in the menuSignal method that does the job of the menu item GoLive passes to it. Thus, most extensions don't need to deal with menus themselves very often. You might do so in order to enable or disable a menu in the menu bar, or to obtain a list of menu items programmatically.

If GoLive interpreted the markup source shown at the beginning of this Menu Object section, you could retrieve this menu object by name from the menus global array as the following code does.

```
var myMenu = menus["ADBEHello"]
```

You can retrieve menus from this array by numeric index, as well.

The items property of the menu object provides an array of menuitem objects that are this menu's menu items. You can retrieve individual menu items from this array by name or by numeric index. For example, the following line of code retrieves the ADBEThis menu item by name.

```
var myFirstItem = myMenu.items["ADBEThis"]
```

To retrieve this item by numeric index, use JavaScript code similar to the following line.

```
var myFirstItem = myMenu.items[0]
```

Note that element 0 is the first element in the array.

In GoLive 5, the menus global collection contains only menus created by Extend Script extensions. In GoLive 6, the **File**, **Edit**, **Type**, and **Help** menus always appear in this order as the first four elements of this collection. ExtendScript extensions can also add items to these additional menus in GoLive 6. An ExtendScript extension cannot redefine the behavior of a built-in menu or menu item.

Menu Object Properties

Properties of the menu object allow you to use its JavaScript `name` to retrieve it; obtain the `title` it displays in the GoLive menu bar; get or set the currently-selected menu item; and retrieve its menu items by name or by numeric index.

`items`	*Collection*	Read-only array of this menu's menu items. You can retrieve items from this array by numeric index or by name. When two or more objects in a collection have the same `name` value, those objects cannot be retrieved by name reliably.
`name`	*String*	The JavaScript name of the menu object, as specified by the `name` attribute of the `<jsxmenu>` tag that defines the menu. Read-only.
`selection`	*Number*	Index of the currently-selected menu item. Set this property to highlight a menu item programmatically.
`title`	*String*	The title of the menu as displayed to the user. Use the ampersand character (&) to cause Windows to underline the character following the ampersand character and use this character to alt-navigate this menu item. Use double ampersands to display an ampersand character. The property value is the title string with the ampersand characters removed. On Mac OS platforms, the SDK ignores ampersand characters and removes them from the displayed string.

Menu Object Functions

You can use functions of the menu object to add or remove menu items programmatically.

addItem

menuObj`.addItem (`*name*`,` *title* `[,` *before*`])`

Appends a new menu item as specified.

Availability

5.0, 6.0

Parameters

name	*String*	JavaScript name of the menuitem object this method creates.
title	*String*	The new menu item's display text.
before	*Number*	Optional. Menu item position before which this method inserts the new menu item.

removeItem

menuObj.removeItem (*name*)

Removes the specified menu item.

Availability

5.0, 6.0

Parameters

name	*String*	JavaScript name of the menuitem object this method removes.
	Number	Numeric index of the menu item to remove. Item 0 is the first item in the menu.

MenuItem Object

Availability

5.0, 6.0

GoLive creates a menuitem object when it interprets a <jsxitem> element that is part of a <jsxmenu> element. For example, the following markup tags define a menu and two menu items.

```
<jsxmenubar> // opens definition of all menus
   <jsxmenu name="ADBEHello" title="Hello, GoLive!"> // Hello menu
      <jsxitem name="ADBEThis" title="Do Something"> // menu item
      <jsxitem name="ADBEThat" title="Do Something Else" > menu item
   </jsxmenu> // closes definition of Hello menu
</jsxmenubar> // closes definition of all menus
```

In GoLive 5, this tag adds a menu item to the **Special** menu or any menu created by an Extend Script extension. In GoLive 6, Extend Script extensions can add menuitem objects to the **File**, **Edit**, **Type**, and **Help** menus, also.

By default, GoLive enables menus and menu items when it creates them; however, you can enable or disable a menu or any menu item programmatically.

When a <jsxitem> element provides the dynamic attribute, GoLive enables or disables its corresponding menu item dynamically. Every time GoLive is about to display the menu, it calls the menuSetup method of each menu item that provides the dynamic attribute. Your

implementation of the `menuSetup` method sets either or both of the `enabled` and `checked` states of the menu item GoLive passed to it. GoLive then displays the menu and its items as specified by the values of these properties.

When the user chooses a menu item, GoLive calls the `menuSignal` method of the extension that provided the item, passing the selected menu item as its argument. Your implementation of the `menuSignal` method performs the tasks associated with the menu item passed to it.

Access To MenuItem Objects

This section describes several ways in which GoLive makes menu items available to your extension.

MenuItem Objects Passed to the menuSignal Method

When the user interacts with one of your extension's menu items, GoLive passes the appropriate menuitem object to your extension's `menuSignal` method.

Retrieving MenuItem Objects From the MenuItems Global Array

You can also retrieve menuitem objects by name or by numeric index from the `menuitems` global array, as the following line of JavaScript code does.

```
var aMenuItem = menuitems["ADBEThat"];
var firstMenuItem = menuitems[0];
```

Retrieving MenuItem Objects From Their Containing Menu Object

The `items` property of a menuitem object's containing menu object holds an array of menuitem objects that are this menu's menu items. You can retrieve individual menu items from this array by name or by numeric index. For example, the following line of code retrieves the menu item having `ADBEThis` as the value of its `name` property.

```
var myFirstItem = myMenu.items["ADBEThis"]
```

To retrieve this item by numeric index, use JavaScript code similar to the following line.

```
var myFirstItem = myMenu.items[0]
```

Note that element `0` is the first element in the array.

MenuItem Object Properties

Properties of the menuitem object enable JavaScript callers to retrieve a menu item by name; get and set its display text; get or set its checked and enabled states; and initialize it dynamically.

`checked`	*Boolean*	State of the menu item's check mark. Set to `true` to add a checkmark to the menu item, or set to `false` to remove the menu item's checkmark. When reading this property, `true` indicates that the menu item is checked, and `false` indicates that it is not.
`dynamic`	*Boolean*	Use of dynamic initialization. When this valueless property is present, the SDK calls the menuSetup function before displaying this menu item. Your implementation of this function sets the menu item's enabled or checked states as necessary. When this property is not present, GoLive enables the menu item without placing a checkmark next to it.
`enabled`	*Boolean*	Menu item's enabled or disabled state. By default, menu items are enabled upon creation. Set this property to `false` to disable the menu item; conversely, set to true to enable a menu item that is already disabled. When reading this property, `true` indicates that the menu item is enabled, and `false` indicates that it is disabled. The user cannot choose a disabled menu item, which is drawn in gray text.
`menu`	*Menu*	The menu in which this item appears. Read-only.
`name`	*String*	The JavaScript name of the menu item object, as specified by the `name` attribute of the `<jsxmenuitem>` tag that defines the menu item. Read-only.
`title`	*String*	The text this item displays in a menu. If the value of this property is a single dash, the menu item is a separator.

MenuItem Object Functions

The menuitem object provides no functions of its own. A menu item's most important behaviors are provided by event-handling functions that the Extend Script SDK calls when the user interacts with a menu item.

NOTE: To use a menu item, your extension must provide the global event-handling methods that the Menus and Menu Items section of Chapter 6, "Event-Handling Functions," describes. The SDK calls these event-handling functions in response to user interactions with menus and menu items.

Module Object

Availability

5.0, 6.0

The Module object represents an individual extension module. The SDK creates a module object when it interprets a `main.html` file, regardless of whether that file provides a `<jsxmodule>` element. In GoLive 6.0, this object's `folder` property returns a reference to the folder that holds the `main.html` file the SDK interpreted to create the module object.

Acquiring a Module Object

The `module` global variable makes available the Module object that represents the currently-running extension to JavaScript callers within that module's scope.

```
module
```

The `modules` global variable holds a collection of module objects representing all currently-running modules. You can retrieve modules from this collection by name or by numeric index.

```
modules[name]
modules[num]
```

Module Object Properties

Properties of the module object provide the JavaScript name of the extension module, enable debugging services, and enable dynamic localization of the module.

`debug`	*Boolean*	true indicates that debugging services are enabled. The SDK sets this property to `true` when any currently-loaded `<jsxmodule>` element provides the valueless `debug` attribute.
`locale`	*String*	The language in which the host GoLive application is running. This value is a country code defined by the ISO-3166-1 standard; for example, the `DE` code represents German and the `FR` code represents French. Note that the US code represents all dialects of English. The value is upper-case. Setting this value affects the way the `localize` method works; it does not affect the behavior of menus or dialogs.
`folder`	*File*	A File object representing the folder that holds the `Main.html` file GoLive interpreted to create this module. GoLive 6 only.
`name`	*String*	This extension module's JavaScript name. This value is specified by the `name` attribute of the `<jsxmodule>` tag the SDK interpreted to create this module. If the extension does not supply a `<jsxmodule>` element or does not supply a `name` attribute for this element, the SDK sets this value to the name of the folder that holds this extension's `main.html` file. Read-only.

Module Object Functions

The module object defines only one function of its own, the `localize` function.

localize

`module.localize(`*message*`)`

Attempts to translate the *message* string from the English language to the language in which GoLive is running.

Availability

5.0, 6.0

Returns

String	Attempted translation of *message* argument.

Description

Attempts to translate the *message* string from the English language to the language in which GoLive is running. Setting the `locale` property to `"NONE"` disables this method.

First, the SDK searches for this string in the localization table that this module's `<jsxlocale>` element provides; if the table provides a translation of *message* which fits the current language setting, this method returns the translated string.

If this attempt fails, the SDK searches for the *message* string in the GoLive application's own translation database; if this database provides a translation of *message* which fits the current language setting, this method returns the translated string. In this case, the SDK ignores the value of the `locale` property. (For example, suppose that your module's locale table translates English-language strings into German and Norwegian strings, and that you are running it on a Spanish-language host system. None of the translations supplied by the locale table are appropriate, so GoLive returns a Spanish-language translation from its own database, if possible.)

If both of these attempts fail, this method returns the original *message* argument.

Picture Object

Availability

5.0, 6.0

The SDK creates a picture object when it interprets an <img> element that references a graphical image file, such as a GIF or JPEG file. The SDK provides its own version of the <IMG> tag which allows extensions to assign JavaScript names to <IMG> elements. For more information, see the description of the `<img>` tag on page 275 of this Reference.

Picture Object Properties

Properties of the picture object associate a JavaScript name with an image, and scale the picture to a specified width and height.

`height`	*Number*	The height of the picture, expressed as a number of pixels. Setting this property causes the picture to be stretched accordingly the next time GoLive draws it.
`name`	*String*	The JavaScript name of this picture object, as specified by the name attribute of the `<img>` tag the SDK interpreted to create this picture object. Read-only.
`width`	*Number*	The width of the picture, expressed as a number of pixels. Setting this property causes the picture to be stretched accordingly the next time GoLive draws it.

Picture Object Functions

draw

pictureObj`.draw (`*x*`, `*y*`)`

Draws the *pictureObj* at the specified location in Layout View.

Availability

5.0, 6.0

Parameters

`x`	*Number*	The horizontal location of the picture's upper-left corner in the coordinate plane of the dialog, palette, or document window that displays it. The origin of this coordinate plane is the point at the upper-left corner of the container that displays the picture. The value of the `x` property specifies the number of pixels by which GoLive offsets the picture's location from this origin. Positive values of the `x` property specify a location to the right of the origin, while negative values specify a location to the left of the origin.
`y`	*Number*	The vertical location of the picture's upper-left corner in the coordinate plane of the dialog, palette, or document window that displays it. The origin of this coordinate plane is the point at the upper-left corner of the container that displays the picture. The value of the `y` property specifies the number of pixels by which GoLive offsets the picture's location from this origin. Positive values of the `y` property specify a location below the origin, while negative values specify a location above the origin.

prefs Object

Availability

5.0, 6.0

Extension modules can use the `prefs` object to store preference data that persists between user sessions. Because the `prefs` global property makes this object always available to all extension modules, extensions can also use it to share or exchange user preference data.

Acquiring the prefs Object

You can get this object from the `prefs` global variable.

```
prefs
```

Description

Extension modules create their own preferences as properties of the Prefs object. Creating a new preference value is as easy as writing to a prefs property. For example, the following JavaScript code creates the `present` property that holds the `I am present` string value.

```
prefs.present = "I am present";
```

The following JavaScript code tests for the presence of this property and this value.

```
if (prefs.present == "I am present")
{
   // your code that uses prefs
}
```

ProgressLog Object

Availability

6.0

The ProgressLog object implements a text message buffer with stack-like capabilities. You can use it to log error messages and progress information.

Acquiring a ProgressLog Object

You can create a ProgressLog object whenever you need one; to do so, call the createProgressLog method of the app Object.

```
app.createProgressLog()
```

Before attempting to add messages to the progress log this method returns, call *progressLogObj*`.begin()` once.

ProgressLog Object Functions

You can call functions of the ProgressLog object to manipulate the contents of the progress log stack and specify the nesting level at which to add a new message.

progressLogObj.`begin()`
: Call once before calling any other *progressLogObj* methods.

progressLogObj.`addMessage`(*type*, *message*, *url*)
: Adds *message* string to *progressLogObj* at the stack pointer's current level.

progressLogObj.`pushMessage`(*type*, *message*, *url*)
: Creates a new nesting level in *progressLogObj* and adds *message* to this level.

progressLogObj.`popMessage()`
: Removes and returns the topmost message from the *progressLogObj* stack.

addMessage

progressLogObj`.addMessage(`*kind*, *message*`[,` *url*`])`

Adds *message* string to *progressLogObj* at the stack pointer's current level.

Availability

6.0

Parameters

kind	*String*	Kind of icon to display. Valid values are:
	`error`	Error message icon.
	`fatal`	Icon for fatal errors.
	`info`	Icon for information.
	`item`	Icon for a single item.
	`warning`	Warning icon.
message	*String*	String to add to the message stack.
url	*String*	Optional. URL associated with this message, used when user double-clicks this line or displays its contextual menu.

begin

progressLogObj`.begin()`

Initialize *progressLogObj*.

Availability

6.0

Description

Call once before calling any other *progressLogObj* methods.

end

progressLogObj.end()

Stop using *progressLogObj* to store progress messages.

Availability

6.0

popMessage

progressLogObj.popMessage()

Removes the topmost level of the *progressLogObj* message stack.

Availability

6.0

pushMessage

progressLogObj.pushMessage(*type*, *message*, *url*)

Places *message* string on the top of the *progressLogObj* message stack and creates a new top level.

Availability

6.0

Parameters

message	*String*	String to add to the message stack.
url	*String*	URL associated with this message (for double-click or context).

ServerInfo Object

Availability

6.0

The `ServerInfo` object enables Extend Script extensions to request data from server information files on a Dynamic Content server. The `ServerInfo` object's methods can return the results of these page requests as `String` or `Document` objects.

Acquiring the ServerInfo Object

You can obtain the `ServerInfo` object from the global variable of the same name.

```
ServerInfo
```

ServerInfo Object Functions

Methods of this object pass *name,value* pairs to request data from the *infofile*, which is a named server information page. The "non-refresh" functions get results from the cache if available. The *xx*`AsString` functions always return a `String`, even if the result is an error page. The *xx*`AsDocument` functions return a `Document` object that holds the results of the request; if the result is an error, this `Document` object's `element` property (the `GoLiveMarkup` object) has no children.The Markup object that the *xx*`AsMarkup` functions return has an extra `document` property that is document that contains the markup element GoLive parsed to create the markup object. If you change this markup object, you must close and save its associated document object; otherwise, the user will be prompted to save the "" document when GoLive closes.

`ServerInfo.requestDataAsString(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Return as a `String` the `ServerInfo` data specified by arguments.

`ServerInfo.requestDataAsDocument(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Return as a `Document` object the `ServerInfo` data specified by arguments.

`ServerInfo.refreshAndRequestDataAsString(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Refresh cache and return as a `String` the `ServerInfo` data specified by arguments.

`ServerInfo.refreshAndRequestDataAsDocument(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Refresh cache and return as a `Document` object the `ServerInfo` data specified by args.

requestDataAsString

`ServerInfo.requestDataAsString(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Return as a `String` the `ServerInfo` data specified by arguments.

Availability

6.0

Parameters

infoFileName	*String*	Filename of the server information page (`.asp`, `.jsp`, or `.php` file) from which to obtain data. Not a pathname, just the filename.
name1 ... nameN	*String*	The name in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.
val1 ... valN	*String*	The value in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.

Returns

String	String representation of the data specified by arguments. Returns cached results if available. Always return a `String`, even if the result is an error page.

requestDataAsDocument

`ServerInfo.requestDataAsDocument(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Return as a `Document` object that holds the `ServerInfo` data specified by arguments.

Availability

6.0

Parameters

infoFileName	*String*	Filename of the server information page (`.asp`, `.jsp`, or `.php` file) from which to obtain data. Not a pathname, just the filename.
name1 ... nameN	*String*	The name in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.
val1 ... valN	*String*	The value in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.

Returns

Document	A `Document` object that holds the results of the request this method's arguments specify; if the result is an error, this document's `element` property has no child nodes. Returns cached results if available.

refreshAndRequestDataAsString

`ServerInfo.refreshAndRequestDataAsString(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Refresh cache and return as a `String` the `ServerInfo` data specified by arguments.

Availability

6.0

Parameters

infoFileName	*String*	Filename of the server information page (`.asp`, `.jsp`, or `.php` file) from which to obtain data. Not a pathname, just the filename.
name1 ... nameN	*String*	The name in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.
val1 ... valN	*String*	The value in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.

Returns

String	String representation of the data specified by arguments. Always return a `String`, even if the result is an error page.

refreshAndRequestDataAsDocument

`ServerInfo.refreshAndRequestDataAsDocument(`*infoFileName, name1, val1, name2, val2, ... nameN, valN*`)`

Refresh cache and return a `Document` object that holds the `ServerInfo` data specified by arguments.

Availability

6.0

Parameters

infoFileName	*String*	Filename of the server information page (`.asp`, `.jsp`, or `.php` file) from which to obtain data. Not a pathname, just the filename.
name1 ... nameN	*String*	The name in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.
val1 ... valN	*String*	The value in a *name*,*value* pair that specifies `ServerInfo` data to retrieve.

Returns

Document	A `Document` object that holds the results of the request this method's arguments specify; if the result is an error, this `Document` object's `element` property has no child nodes.

settings Object

Availability

6.0

The Settings object provides access to a variety of settings, configurations, and user preferences in the GoLive 6.0 application running the extension. You can get and set properties of this object to load and extend DTDs, user settings, registered file types, Web database settings, and so on. You can also use this object to add your own tag definitions and file types to the internal databases GoLive uses to parse documents.

Acquiring a Settings Object

You can get the Settings object from the `settings` global property in GoLive 6.0.

```
settings
```

NOTE: The Settings object is not available in GoLive 5.0 but extensions still have access to some user preference settings in this environment. For more information, see "app.prefs Object" on page 56.

settings Object Properties

Properties of the Settings object provides access to browser settings, style sheet settings and more; for example, the following line of code retrieves the SettingsCSS object, which holds style sheet information.

```
// get settings object from the settings global
var cssStyles = settings.css;
```

`aglmodules`	*SettingsAGLModules*	This collection allows you to enable or to disable GoLive modules programmatically. Changes take effect on restart. You can either use the module name as index or access the collection by numeric index
`css`	*SettingsCSS*	The access object for style sheet information.
`fileMappings`	*SettingsFileMappings*	The *fileMappings* object permits the loading of additional file mappings that GoLive should recognize
`markup`	*SettingsMarkup*	The markup object allows the configuration of markup languages and the text generation.
`sdkmodules`	*SettingsSDKModules*	This collection allows you to enable or to disable GoLive SDK extensions programmatically. Changes take effect on restart. You can either use the extension name as index or access the collection by numerical index
`userAgentProfiles`	*SettingsUAP*	This object enables you to add user agent profiles to the GoLive environment.

settings.aglmodules Object

Availability

6.0

This collection allows you to enable or to disable GoLive modules programmatically. Changes take effect on restart. This collection's index operator `[]` retrieves its elements by module name or by numeric index.

Acquiring the settings.aglmodules Object

This object is available as a property of the object that the `settings` global variable provides.

```
settings.aglmodules
```

You can retrieve individual modules from this collection by numeric index or by name:

```
settings.aglmodules[0].switchOff() // turn off the first module
settings.aglmodules[name].switchOff() // turn off name module
```

settings.aglmodules Object Properties

[*name*]	*String*	JavaScript name of the GoLive module to retrieve.
[*num*]	*Number*	Numeric index of the GoLive module to retrieve.
`length`	*Number*	The number of elements in this collection. Read only.

settings.aglmodules Object Functions

isOn

`settings.aglmodules[`*name*`].isOn()`

`settings.aglmodules[`*num*`].isOn()`

Returns `true` if the specified module is active.

Availability

6.0

Returns

boolean	`true` indicates that the specified module is active.

isKnown

`settings.aglmodules[`*name*`].isKnown()`

`settings.aglmodules[`*num*`].isKnown()`

Returns `true` if the specified module is known to the GoLive application.

Availability

6.0

Returns

boolean	`true` indicates that GoLive loaded the specified module successfully.

Description

NOTE: A module can be known but not active.

switchOn

settingsAGLModulesObj`[`*indexNum*`].switchOn([`*keyword*`])`

settingsAGLModulesObj`[`*name*`].switchOn([`*keyword*`])`

Call this method on one of the elements to activate the module.

Availability

6.0

Parameter

keyword	*String*	Optional. Pass `$$$all$$$` to enable all GoLive modules.

Description

This method cannot enable the **ModulesManager** or **ExtendScript** modules.

switchOff

settingsAGLModulesObj`[`*indexNum*`].switchOff([`*keyword*`])`

settingsAGLModulesObj`[`*name*`].switchOff([`*keyword*`])`

Call this method on one of the elements to deactivate the module.

Availability

6.0

Parameter

keyword	*String*	Optional. Pass `$$$all$$$` to disable all GoLive modules.

Description

This method cannot disable the **ModulesManager** or **ExtendScript** modules.

settings.css Object

Availability

6.0

The `settings.css` object provides programmatic access to style sheet information.

Acquiring a settings.css Object

```
settings.css
```

settings.css Object Functions

`settings.css.loadBasic` (*relURL*)
Loads the basic CSS.

`settings.css.loadPrefs` (*relURL*)
Loads the CSS preferences.

`settings.css.loadSelectors` (*relURL*)
Loads the CSS selectors.

loadBasic

`settings.css.loadBasic(`*relURL*`)`

Loads the basic CSS the app uses to render HTML.

Availability

6.0

Parameters

relURL	*String*	Relative path to the style sheet.

Returns

boolean	`true` indicates success.

Description

NOTE: The former basic CSS is cascaded with the loaded one - it does not replace it!

loadPrefs

`settings.css.loadPrefs(`*relURL*`)`

Loads CSS preferences from the specified file.

Availability

6.0

Parameters

relURL	*String*	Relative path to the style sheet.

Returns

boolean	`true` indicates success.

loadSelectors

`settings.css.loadSelectors(`*relURL*`)`

Loads CSS selectors from the specified file.

Availability

6.0

Parameters

relURL	*String*	Relative path to the style sheet.

Returns

boolean	`true` indicates success.

Description

The CSS selectors are additions to a sub menu in the Context Menu in the CSS Editor.

settings.fileMappings Object

Availability

6.0

The `fileMappings` object permits the loading of additional file mappings that GoLive should recognize.

Acquiring a SettingsFileMappings Object

```
settings.fileMappings
```

settings.fileMappings Object Properties

`useInternetConfig` *Boolean* Indicates whether the host platform uses Internet Config to get file mapping information. On Windows, this value is always `false`.

- *true* GoLive is running on a Mac OS that uses Internet Config to get file-mapping information.
- *false* GoLive is running on a Windows platform or this Mac OS platform does not use Internet Config.

settings.fileMappings Object Functions

openInternetConfig

```
settings.fileMappings.openInternetConfig()
```

Call this method to open the Mac OS Internet Config panel.

Availability

6.0

Description

Only on the Macintosh! On Windows, this call does nothing.

loadMappings

`settings.fileMappings.loadMappings(`*relURL*`)`

Reads `.aglfmi` files from the *relURL* folder to extend the GoLive application's file mapping information.

Availability

6.0

Parameters

relURL	*String*	Relative URL to folder containing `.aglfmi` files.

Returns

Boolean	`true` indicates success.

Description

An `.aglfmi` file is an XML file that has the same syntax as the file mapping files that reside in the `Settings/FileMappings/` folder.

---

settings.markup Object

Availability

6.0

The `settings.markup` object enables you to get and set markup-language and text-generation preferences.

Acquiring a settings.markup Object

`settings.markup`

settings.markup Object Properties

Properties of the `settings.markup` object enable you to get and set preferences that govern source parsing and text generation.

`commentScripts`	*String*	Comma-separated list of languages in which the content of <script></script> are not put into "<!--" and "-->".
`foreignBinaryToken`	*String*	Corresponds to *app.symmetricTokens*.
`foreignStartToken`	*String*	Corresponds to *app.asymmetricTokens*.
`htmlAttrCase`	*String*	Stores how new HTML attributes names are written to source code. Values are the strings "upper", "lower", "capital" and "database". The same strings are used to set the property. Only meaningful for HTML - other markup languages are case sensitive per definition.
`htmlElementCase`	*String*	Stores how new HTML element names are written to source code. Values are the strings "upper", "lower", "capital" and "database". The same strings are used to set the property. Only meaningful for HTML - other markup languages are case sensitive per definition.
`indentAmount`	*Number*	Stores the number of indention characters which are written for every indention.
`indentAsciiNum`	*Number*	Stores the character that is used to create markup source code indention. Value is the ASCII number - 9 and 32 are the two only values that make sense!
`lineBreakMode`	*String*	Stores the global line break mode of the application. Returns "mac", "unix" or "win". The same strings can be used to set the line break mode.
`quoteMode`	*String*	Stores the information how new attribute values are quoted. Valid values are "always", "exceptNums" and "necessary".
`scanBrackets`	*Boolean*	Stores a Boolean whether the characters "[" and "]" in source code are treated as braces for special elements. Old versions of Lasso used this syntax. This property is equal to the property *app.scanBrackets*

settings.markup Object Functions

You can call these functions to manipulate glue files:

`settings.markup.loadGlueBase` (*relURL*)
Load all `.aglmgb` files found in the *relURL* folder.

`settings.markup.loadGlueAdditions` (*relURL*)
Load additional `.aglmgb` files found in the *relURL* folder.

loadGlueBase

`settings.markup.loadGlueBase(`*relURL*`)`

Loads markup glue base from the `.aglmgb` file from the *relURL* location.

Availability

6.0

Parameters

relURL	*String*	Relative path to the markup glue base (`.aglmgb`) file or the folder that contains it.

Returns

Boolean	`true` indicates success.

Description

You can use this method to patch existing Markup Glues.

This method scans the *relURL* folder for `.aglmgb` (Adobe GoLive Markup Glue Base) files. These XML files contain information GoLive needs to work with markup documents. Information on how to create your own `.aglmgb` file is not yet available; for examples, look at the `.aglmgb` files that GoLive provides.

loadGlueAdditions

`settings.markup.loadGlueAdditions(`*relURL*`)`

Loads markup glue additions from `.aglmga` files at the *relURL* location.

Availability

6.0

Parameters

relURL	*String*	Relative path to an additional markup glue (`.aglmga`) file or path to a folder that contains one or more `.aglmga`) files

Returns

Boolean	`true` indicates success.

Description

This method scans the *relURL* folder for `.aglmga` (Adobe GoLive Markup Glue Addition) files. GoLive treats these files as an addition to existing markup glues.

The `.aglmga` file is an XML file that specifies the glue it extends and the additions it defines. Information on how to create your own `.aglmga` file is not yet available; for examples, look at the `.aglmga` files that GoLive provides.

settings.sdkmodules Object

Availability

6.0

This collection allows you to enable or to disable GoLive SDK extensions programmatically. Changes take effect on restart. You can either use the extension name as index or access the collection by numeric index.

Acquiring the SettingsSDKModules Object

`settings.sdkmodules`

SettingsSDKModules Object Properties

[*name*]	*String*	JavaScript name of the Extend Script extension module to retrieve.
[*num*]	*Number*	Numeric index of the Extend Script extension module to retrieve.
`length`	*Number*	The number of elements in this collection. Read only.

settings.sdkmodules Object Functions

isKnown

`settings.sdkmodules[`*name*`].isKnown()`

`settings.sdkmodules[`*num*`].isKnown()`

Returns `true` if the specified Extend Script extension module is available to the GoLive application.

Availability

6.0

Returns

Boolean	`true` indicates that the SDK loaded the specified extension module successfully.

isOn

`settings.sdkmodules[`*name*`].isOn()`

`settings.sdkmodules[`*num*`].isOn()`

Returns `true` if the specified Extend Script extension module is active.

Availability

6.0

Returns

Boolean	`true` indicates that the specified extension module is active.

switchOn

`settings.sdkmodules[`*name*`].switchOn([`*keyword*`])`

`settings.sdkmodules[`*num*`].switchOn([`*keyword*`])`

Activates a specific Extend Script extension module; optionally, enables all Extend Script extensions.

Availability

6.0

Parameter

keyword	*String*	Optional. Pass `$$$all$$$` to enable all Extend Script extensions.

switchOff

`settings.sdkmodules[`*name*`].switchOff()`

`settings.sdkmodules[`*num*`].switchOff()`

Deactivates a specific Extend Script extension module; optionally, disables all Extend Script extensions.

Availability

6.0

Parameter

keyword	*String*	Optional. Pass `$$$all$$$` to enable all Extend Script extensions.

settings.userAgentProfiles Object

Availability

6.0

The `settings.userAgentProfiles` object provides access to preferences that govern the way GoLive Layout view and syntax checking models the behavior of various HTML “user agents” such as browsers or mobile internet device displays.

A user agent specifies its preferences as an `.agluap` file that resides in the agent’s own subfolder of the `GoLive/Settings/User Agent Profiles` folder. If necessary, this folder can also hold other agent-specific files, such as fonts or graphics GoLive needs to simulate the user agent’s display.

Acquiring a settings.userAgentProfiles Object

```
settings.userAgentProfiles
```

settings.userAgentProfiles Object Properties

`defaultProfile`	*String*	The unique ID name of the default User Agent Profile.

settings.userAgentProfiles Object Functions

loadProfiles

`settings.userAgentProfiles.loadProfiles(`*relURL*`)`

Reads `.agluap` files from the *relURL* folder to define new User Agent Profiles or patch existing ones.

Availability

6.0

Parameters

relURL	*String*	Relative path to `.agluap` files.

Returns

Boolean	`true` indicates success.

SiteReference Object

Availability

5.0, 6.0

The `SiteReference` object encapsulates a reference to a file, a folder, or an external link such as an email address. This object provides information about the referenced resource, such as its location, file type, size, lock status, and a list of its anchors.

The site reference object's methods get files and links of a specified type from the referenced location, making these objects useful for batch-processing a site's files. These methods return a `SiteRefIterator` object, which provides name-based, numeric, and iterative access to a group of `SiteReference` objects.

In GoLive 6, the `SiteReference` object can also store persistent string or integer data as a named value. You can use this feature to add your own properties to reference objects.

Acquiring SiteReference Objects

All documents provide the `ref` property, which holds a `SiteReference` representing the site that uses the document. This `SiteReference` is available only when document's site window is open.

When a document represents a site file that is open,

- its `site` property holds a `SiteReference` object representing the root folder of that site.

```
document.site// site's root fld when site window is frontmost
```

- its `homePage` property holds a `SiteReference` object representing the home page of the that site.

```
document.homePage// site's homepage when site window is frontmost
```

The document object's `site` and `homePage` properties are `null` when the site document is closed or when the document object does not represent a site file; for example, these properties are `null` for markup documents.

Both the File object and the SiteReference object provide methods that return collections of `SiteReference` objects as the elements of a `SiteRefIterator` object. For additional information, see the description of the "File Object" on page 96.

SiteReference Object Properties

All `SiteReference` properties are read-only.

(*name*)	*String*	Add as strings the names of any properties you need for your Undo operation.
`anchors`	*Array*	The array of anchors this file contains. This array is empty if there are no anchors on the page.
`fileSize`	*Number*	The physical size of a file; `0` for other types of references.
`local`	*Boolean*	`true` if the reference is a file on local storage; otherwise, `false`.
`lockStatus`	*Number*	Locking status of the reference: `0` Unknown `1` Read-only. `2` Read/write `3` Checked in `4` Checked out exclusively `5` Checked out non-exclusively `6` broken

SiteReference object properties (*continued from preceding page*)

`longUrl`	*String*	`"file:///"` followed by the URL of the `.site` file with spaces and special characters represented as in Unix; for example, `"%20"` represents a space. Thus, if the referenced `.site` file is on the disk named `PB`, this value would look like the following example: `"file:///PB/Desktop%20Folder/Adobe%20GoLive%205.0/"`
`mimeType`	*String*	The MIME type of the reference.
`name`	*String*	The name, usually the file name or its URL.
`prefs`	*Prefs*	A preferences object you can use to store any data you need to associate with this reference; for example, you might use this object to mark the reference in some way or to associate your own values with this reference. For more information, see "prefs Object" on page 164.
`protocol`	*String*	The file-transfer protocol (HTTP, FTP, etc.) this object's `upload` method uses to transfer the file this object represents. •• are the values enumerated anywhere
`siteDoc`	*Document*	The Site document to which this object refers. This property is `null` if the Site document is not open.
`status`	*String*	One of the following strings, indicating status of the reference: `error` Parsing error or other error `empty` Empty reference `checking` SDK is checking this reference now. `invalid` Invalid reference. `ok` The reference is valid.
`title`	*String*	The document title, as specified in the `<title>` tag of the document's HTML source representation. Empty for other types.
`type`	*String*	One of the following strings indicating the type of resource this object represents: `html` HTML file `folder` Folder or directory `alias` File or folder alias `image` Image file `mail` E-mail address `invalid` Invalid reference
`url`	*String*	The URL of the `.site` file, as used in the document. For example, if `PB` is the name of the disk that holds the `.site` file, this value would look like the following example: `"PB/Desktop Folder/Adobe GoLive 6.0/"`

SiteReference Object Functions

Methods of the `SiteReference` object get files by location and type; get a referenced page's incoming or outgoing links; highlight the referenced object in Layout view; and open the referenced object programmatically.

getFiles

siteReferenceObj`.getFiles ([`*type*`])`

Retrieves `SiteReference` objects representing the files in the *siteReferenceObj* folder.

Availability

5.0, 6.0

Parameter

type	*String*	Optional. A string composed of one or more values of the `type` property, separated by any non-alpha character. For example, the `html+folder` string is a valid argument to this function. **NOTE:** The `mail` value is not a valid argument to this function.

Returns

`SiteRefIterator`

Description

Optionally, filters results on one or more values of the `SiteReference.type` property.

NOTE: This method works only when *siteReferenceObj* represents an open `.site` file.

getIncoming

siteReferenceObj.`getIncoming (`*type*`)`

Retrieves `SiteReference` objects that link to the *siteReferenceObj* page.

Availability

5.0, 6.0

Parameter

type	*String*	Optional. Specifies the kinds of links to retrieve, as a string composed of one or more values of the `type` property, separated by any non-alpha character. For example, the `html+image` string is a valid argument to this function. **NOTE:** The `folder` value is not a valid argument to this function.

Returns

`SiteRefIterator`

Description

Retrieves `SiteReference` objects that link to the *siteReferenceObj* page.

This method works when no Site document is open, but it retrieves links only from currently-open documents; thus it would return a reference only if another open document referenced the *siteReferenceObj* page.

getNamedLong

siteReferenceObj.`getNamedLong(`*name*`)`

Returns the integer data stored under the specified *name*.

Availability

6.0

Parameters

name	*String*	Name of the data to retrieve, as created by the `setNamedLong` method.

Returns

Number	The long integer data stored as *name*.

getNamedString

siteReferenceObj.`getNamedString`(*name*)

Returns the string data stored under the specified *name*.

Availability

6.0

Parameters

name	*String*	Name of the data to retrieve, as created by the `setNamedString` method.

Returns

Number	The string stored as *name*.

getOutgoing

siteReferenceObj.`getOutgoing` (*type*)

Retrieves `SiteReference` objects representing pages that the *siteReferenceObj* page references.

Availability

5.0, 6.0

Parameter

type	*String*	Optional. Specifies the kinds of links to retrieve, as a string composed of one or more values of the `type` property, separated by any non-alpha character. For example, the `html+image` string is a valid argument to this function.

Returns

`SiteRefIterator`

open

siteReferenceObj`.open()`

Opens the *siteReferenceObj* reference in GoLive as an HTML document, returns a document object representing it, and makes that document the current document.

Availability

5.0, 6.0

Returns

`Document`

Description

Opens the *siteReferenceObj* reference in GoLive as an HTML document, returns a document object representing it, and makes that document the current document.

NOTE: This method is intended to open HTML documents only. Do not use this method to open other kinds of documents.

setNamedLong

siteReferenceObj`.setNamedLong(`*name*`,` *value*`)`

Creates a *name* property in *siteReferenceObj* and puts the specified long integer *value* in it.

Availability

6.0

Parameters

name	*String*	Name of the property to create.
value	*Number*	Value to store.

Returns

Boolean	`true` indicates success.

setNamedString

siteReferenceObj.setNamedString(*name*, *value*)

Creates a *name* property in *siteReferenceObj* and puts the specified string *value* in it.

Availability

6.0

Parameters

name	*String*	Name of the property to create.
value	*String*	Value to store.

Returns

Boolean	true indicates success.

show

siteReferenceObj.show()

Displays and highlights the *siteReferenceObj* reference in the Site window.

Availability

5.0, 6.0

SiteRefIterator Object

Availability

5.0, 6.0

The SiteRefIterator object provides iterative access to a collection of SiteReference objects that represent the resources a particular site uses, such as files, active documents, and links.

Acquiring SiteRefIterator Objects

The site property of a document object that represents a Site window provides a collection of SiteReference objects representing resources the site currently uses.

```
document.site // site window frontmost
documentObj.site // only documents that are Sites
```

NOTE: In GoLive 6, the SiteReport Object provides a more flexible and powerful means of retrieving site resources according to criteria you specify

SiteRefIterator Object Properties

The `index` property of a `SiteRefIterator` object provides direct access to this collection's elements by means of a numeric index.

[*index*]	*SiteReference*	Index position of the element to retrieve. Valid values are integers ≥ 0.

SiteRefIterator Object Functions

The `SiteRefIterator` object provides functions that facilitate batch-processing its elements.

first

siteRefIteratorObj`.first()`

Returns the first element of the *siteColl* collection, or `null`.

Availability

5.0, 6.0

Returns

SiteReference	The first element of the *siteColl* collection.
`null`	The *siteColl* collection is empty.

last

siteRefIteratorObj`.last()`

Returns the last element of the *siteColl* collection, or `null`.

Availability

5.0, 6.0

Returns

SiteReference	The last element of the *siteColl* collection.
`null`	The *siteColl* collection is empty.

next

siteRefIteratorObj`.next()`

Returns the next element of the *siteColl* collection, or `null`.

Availability

5.0, 6.0

Returns

SiteReference	The next element of the *siteColl* collection.
`null`	No next element in the collection.

prev

siteRefIteratorObj`.prev ()`

Returns the previous element of the *siteColl* collection, or `null`.

Availability

5.0, 6.0

Returns

SiteReference	The next element of the *siteColl* collection.
`null`	No previous element in the collection.

SiteReport Object

Availability

6.0

The `SiteReport` object generates a custom report about the files in a site, and it displays this report in the **Site Report** window. You specify the content of the report by configuring the SiteReport object before generating the report. For example, you might configure this object to list pages created on a specific date that have broken links. To configure a `SiteReport` object, call any of its various `set`*Xyz* methods.

You can display custom data in its own column in the Site Report as well. As the `SiteReport` object iterates over the set of files that compose the site, it sends events your extension uses to

supply custom data to cells (`onReportWinColumnContent Event`) and sort cells according to content (`onReportWinColumnCompare Event`).

For more information, see

- "Site Reporting" on page 201 of the *Extend Script SDK Programmer's Guide*.
- "Custom Columns in the Site Report Window" on page 309 of this Reference.

Acquiring a SiteReport Object

You can use the `new` operator to create a new site report object at any time.

```
new JSXSiteReport;
```

Call this object's methods to specify reporting criteria and generate the site report.

SiteReport Object Functions

Most methods of the SiteReport object are setters that specify the kind of information the site report is to provide. For example, you can call this object's `setSize` method to specify that the report describes files of a specified size. You can call as many or as few of the `set`*Xx* methods as your reporting needs dictate, building a complex set of criteria a file must meet in order to be cited in the report.

Once you've specified the reporting criteria, call the `runReport` method to generate the report.

```
// report on broken links and other HTML problems
var mySiteReport = new JSXSiteReport;
mySiteReport.setBrokenImages();
mySiteReport.setHTMLErrors();
mySiteReport.setHTMLWarnings();
var result = mySiteReport.runReport();
```

You can save reporting criteria for future use by calling the `saveQuery` method, and you can call the `loadQuery` function to load previously-saved criteria.

```
var result = mySiteReport.saveQuery("//C/myfld/savedQuery.txt");
```

To discard the report, call the `resetReport` method. Use this method to reuse the same SiteReport object for multiple reports. It's also useful for discarding interim results as you refine your reporting criteria during extension development.

```
// now use same SiteReport object to generate a different report
mySiteReport.resetReport();
mySiteReport.loadQuery("/customers/data/savedOrder09Aug01.txt");
result = mySiteReport.runReport();
```

Other methods of this object manage custom columns in the report window. Extensions that supply data to custom columns in the report window can register custom functions for GoLive to call in response to the `onReportWinColumnCompare Event` and the `onReportWinColumnContent Event` notifications.

Setting Reporting Criteria

siteReportObj.setSize(*size, size_condition*)
Report each file that is greater than, less than, or equal to a specified *size* in kilobytes.

siteReportObj.setDownloadTime(*time, time_condition, time_unit, download_speed*)
Report each file that downloaded before, at, or after a specified *time* in *time_units*.

siteReportObj.setCreateModDate(*time, create_or_mod_condition, when_condition, time_unit*)
Report each file that has a create/mod date before, at, or after a specified *time* in *time_units*

siteReportObj.setBrokenImages()
Report each file that contains at least one <IMG> element with a broken src link.

siteReportObj.setBadTitles(*use_blank, use_default, use_same_as_filename*)
Report each file that has no title, the default title, or its own filename as its title.

siteReportObj.setHTMLErrors(*browser_set_name*)
Report each file containing HTML that would generate errors in the specified browsers.

siteReportObj.setHTMLWarnings(*browser_set_name*)
Report each file containing HTML that would generate GoLive warnings in the specified browsers.

siteReportObj.setUsesComponent(*component_name*)
Report each file that uses the *component_name* component.

siteReportObj.setNumClicks(*address_name*)
Report each file that uses the *address_name* email address.

siteReportObj.setUsesFont(*fontset_name*)
Report each file that uses the *fontset_name* fonts.

siteReportObj.setUsesColor(*color_hex_value*)
Report each file that uses the *color_hex_value* color.

siteReportObj.setUsesSiteColor(*site_color_name*)
Report each file that uses the *site_color_name* site color.

siteReportObj.setUsesExternalLinks()
Report each file that contains links to external resources, such as images.

siteReportObj.setFileExtension(*extension_name*)
Report each file that ends its name with the *extension_name* extension.

siteReportObj.setProtocol(*uses_http, uses_https, uses_file, uses_ftp, uses_gopher, uses_mailto, uses_news, uses_nntp, uses_telnet, uses_wais*)
Report each file that uses the specified protocols.

siteReportObj.setNumClicks(*path_to_item, num_clicks_away, click_condition*)
Report each file that is more than, less than or equal to *num_clicks_away* from *path_to_item* page.

siteReportObj.setAccessibility *(criterion1* [, *criterion2*, ... , *criterionN*])
Report each file that has accessibility requirements (like use of frames) or problems (like no alt tag).

siteReportObj.registerCustomReport(*functionName, reportString*)
Requests that the runReport function call your *functionName* function.

siteReportObj.setUseAnyOrAllCriteria(*any_or_all_condition*)
Specifies whether files included in site report must meet all or any *siteReportObj* criteria.

Generating Reports

siteReportObj.`runReport`(*show_window*)
Generate a report on the current site using the criteria specified by *siteReportObj*.
siteReportObj.`resetReport`()
Discard all criteria and custom functions to reuse *siteReportObj* for another report.
siteReportObj.`saveQuery`([*path*])
Save the current *siteReportObj* reporting criteria to the *path* file or site folder.
siteReportObj.`loadQuery`([*path*])
Load site reporting criteria from the *path* file or user preferences.
siteReportObj.`addColumn`(`name, width, alignment`)
Add a custom column to the **Report Results** window.
siteReportObj.`removeColumn`(*name*)
siteReportObj.`removeColumn`(*columnID*)
Remove the specified custom column from the **Report Results** window.

Managing Custom Data in the Site Report Window (Site Report Events)

`onReportWinColumnCompare Event`(*custColNumber, custColRowNum, compareColNumber, compareColRowNum*)
Compare data in the custom column to the corresponding data in the specified column.
`onReportWinColumnContent Event`(*colNum, rowNum*)
Supply data to the custom column to the corresponding data in the specified column.

setSize

siteReportObj.`setSize`(*size, size_condition*)

Report each file greater than, less than, or equal to the specified *size* in bytes.

Availability

6.0

Parameters

size	*Number*	Size of the file, in bytes.
size_condition	*String*	The condition a file's size must meet for inclusion in the site report. Specify this value by passing one of the following strings:
	`lessThan`	File is smaller than *size* bytes.
	`moreThan`	File is larger than *size* bytes.
	`exactly`	File is equal to *size* bytes.

Returns

Boolean	`true` indicates success.

setDownloadTime

siteReportObj.`setDownloadTime(`*time*`,` *time_condition*`,` *time_unit*`,` *download_speed*`)`

Report each file which can download in more, less, or the exact amount of *time* in *time_units* at the specified *download_speed*.

Availability

6.0

Parameters

time	*Number*	The download time this method uses for comparisons.	
time_condition	*String*	The condition a file's download time must meet for inclusion in the site report. Specify this value by passing one of the following strings:	
		`lessThan`	Created or modified before *time*.
		`moreThan`	Created or modified after *time*.
		`exactly`	Created or modified at *time*.
time_unit	*String*	The *time* argument's unit of time. Specify this value by passing one of the following strings:	
		`seconds`	*Time* is a number of seconds.
		`minutes`	*Time* is a number of minutes.
		`hours`	*Time* is a number of hours.
download_speed	*String*	The download speed this method uses to calculate download times. Specify this value by passing one of the following strings:	
		`9600`	9,600 baud.
		`14400`	14,400 baud.
		`19600`	19,600 baud
		`28800`	28,800 baud.
		`33600`	33,600 baud.
		`56000`	56,000 baud.
		`ISDN`	ISDN speed; nominally 16 - 64 kilobits per second (kbps), up to 1.54 Million bits per second (mbps) for high-volume transmission.
		`T1`	T1 carrier speed; 1.54 Mbps
		`T3`	T3 speed; up to 44.736 Mbps.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

Example

To select files that take less than five minutes to download at a speed of 56,000 bps, call this method as the following example code does.

```
var myReport = new JSXSiteReport;
myReport.setCriteriaDownloadTime(5, "lessThan","minutes", "56000")
```

setBadTitles

siteReportObj.`setBadTitles`(*use_blank*, *use_default*, *use_same_as_filename*)

Report each file that has no title, a default title, or its own filename as its title.

Availability

6.0

Parameters

use_blank	*Boolean*	Pass `true` to report each HTML file that has no `<title>` element.
use_default	*Boolean*	Pass `true` to report each HTML file that has a default title such as `Welcome To Adobe GoLive...`, `"untitled"`, `"untitled-`*n*`"`, `"Welcome to Adobe.."`, `"Welcome to GoLive..."`, or `""` (no title).
use_same_as_filename	*Boolean*	Pass `true` to report each HTML file that has its own filename as its title. This test is case-insensitive; that is, it considers “Welcome” and “welcome” to be the same.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setCreateModDate

siteReportObj.`setCreateModDate`(*time*, *create_or_mod_condition*, *when_condition*, *time_unit*)

Report each file that has a creation date or modification date which matches *time* or is within the specified number of *time_unit* values before or after *time*.

Availability

6.0

Parameters

time	*Number*	The creation or modification time this method uses for comparisons.
create_or_mod_condition	*String*	Test creation date or modification date. Specify this value by passing one of the following strings:
	`created`	Test creation date.
	`modified`	Test modification date.
when_condition	*String*	Condition the tested date must meet for inclusion in the site report. Specify this value by passing one of the following strings:
	`lessThan`	Created or modified before *time*.
	`moreThan`	Created or modified after *time*.
	`exactly`	Created or modified at *time*.
time_unit	*String*	The *time* argument's unit of time. Specify this value by passing one of the following strings:
	`minutes`	*Time* is a number of minutes.
	`hours`	*Time* is a number of hours.
	`days`	*Time* is a number of days.
	`weeks`	*Time* is a number of weeks.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setBrokenImages

siteReportObj.`setBrokenImages()`

Report each file that contains one or more flawed or broken `<img>` elements.

Availability

6.0

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

Description

Report each file containing one or more `<img>` elements that have any of the following problems:

- image source file missing
- does not provide valid `height` attribute
- does not provide valid `width` attribute

setFileExtension

siteReportObj.`setFileExtension(`*filenameExt*`)`

Report each file that contains a link to a *.filenameExt* file.

Availability

6.0

Parameters

filenameExt	*String*	Filename extension for which this method tests; typically, a dot (`.`) followed by three characters, such as `.img`, `.gif`, `.css`, `.cgi`, and so on.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setHTMLErrors

siteReportObj.`setHTMLErrors`(*browser_set_name*)

Report each file that the GoLive parser marks with HTML errors under the *browser_set_name* rules.

Availability

6.0

Parameters

browser_set_name	*String*	Name of a GoLive browser set currently available from the browser set tab of the Layout window.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

Description

This method instructs the `runReport` method to pass each file to the GoLive parser for interpretation under the *browser_set_name* rules; if the parser marks the file with an HTML error, the filename appears in the site report.

setHTMLWarnings

siteReportObj.`setHTMLWarnings`(*browser_set_name*)

Report each file that the GoLive parser marks with HTML warnings under the *browser_set_name* rules.

Availability

6.0

Parameters

browser_set_name	*String*	Name of a GoLive browser set currently available from the browser set tab of the Layout window.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

Description

This method instructs the `runReport` method to pass each file to the GoLive parser for interpretation under the *browser_set_name* rules; if the parser marks the file with an HTML warning, the filename appears in the site report.

setNumClicks

siteReportObj.`setNumClicks(`*target*`, `*numClicks*`, `*click_condition*`)`

Report each file that is more than or less than *numClicks* from *target* file.

Availability

6.0

Parameters

target	*String*	Relative or absolute path to an item in the current website.
num_clicks	*Number*	Minimum number of links one must traverse to navigate from the file being tested to the *target* page.
click_condition	*String*	Condition the *num_clicks_away* value must meet. Specify this value by passing one of the following strings:
	`lessThan`	Report files less than *num_clicks* from the *target* page.
	`moreThan`	Report files more than *num_clicks* from the *target* page.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setUsesAddress

siteReportObj.`setUsesAddress(`*address_name*`)`

Report each file that uses the *address_name* email address.

Availability

6.0

Parameters

address_name	*String*	Email address for which this method tests. A string in the form *abc@def.ghi* that is an email address.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setUsesComponent

siteReportObj.`setUsesComponent(`*component_name*`)`

Report each file that uses the *component_name* component.

Availability

6.0

Parameters

component_name	*String*	JavaScript name of the component for which this method tests.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setUsesFont

siteReportObj.`setUsesFont(`*font_name*`)`

Report each file that uses the *font_name* font.

Availability

6.0

Parameters

font_name	*String*	Name of font for which this method tests.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

NOTE: You cannot use both `setUsesFont()` and `setUsesFontset()` in one report. If you do, the criteria specified by the last of these methods to execute are effective.

setUsesFontset

siteReportObj.`setUsesFontset(`*fontset_name*`)`

Report each file that uses the *fontset_name* fontset.

Availability

6.0

Parameters

fontset_name	*String*	Name of fontset for which this method tests.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

NOTE: You cannot use both `setUsesFont()` and `setUsesFontset()` in one report. If you do, the criteria specified by the last of these methods to execute are effective.

setUsesColor

siteReportObj.`setUsesColor(`*color_hex_value*`)`

Report each file that uses the *color_hex_value* color.

Availability

6.0

Parameters

color_hex_value	*String*	A string of the form *#nnnnnn* that is the hexadecimal value of the color on which to report.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

NOTE: You cannot use both `setUsesColor()` and `setUsesSiteColor()` in one report. If you do, the criteria specified by the last of these methods to execute are effective.

setUsesSiteColor

siteReportObj`.setUsesSiteColor(`*site_color_name*`)`

Report each file that uses the *site_color_name* site color.

Availability

6.0

Parameters

site_color_name	*String*	A string of the form *#nnnnnn* that is the hexadecimal value of the color on which to report.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

NOTE: You cannot use both `setUsesColor()` and `setUsesSiteColor()` in one report. If you do, the criteria specified by the last of these methods to execute are effective.

setUsesExternalLinks

siteReportObj`.setUsesExternalLinks()`

Report each file that contains links to external resources, such as images.

Availability

6.0

Parameters

URL	*String*	Relative path to folder.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

setProtocol

siteReportObj.`setProtocol(`*protocol1* `[,` *protocol2*`, ... ,` *protocolN*`])`

Report each file that contains a link element or URL of any of the protocols specified.

Availability

6.0

Parameters

protocol	*String*	The protocol on which to report. Valid values are any of the following strings:

`http`	Report files that use HTTP URLs.
`https`	Report files that contain HTTPS URLs.
`file`	Report files that contain file URLs.
`ftp`	Report files that contain FTP URLs.
`gopher`	Report files that contain Gopher URLs.
`mailto`	Report files that contain mailto URLs.
`news`	Report files that contain news URLs.
`nntp`	Report files that contain NNTP URLs.
`telnet`	Report files that contain telnet URLs.
`wais`	Report files that contain WAIS URLs.

To search for multiple protocols, pass a comma-separated list of any of these strings, in any order:

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

Description

To search for more than one protocol, pass a comma-separated list of strings as the arguments to this function.

Example

```
var mySiteRpt = new JSXSiteReport;
mySiteRpt.setProtocol ("http", "wais", "gopher");
mySiteRpt.setUseAnyOrAllCriteria("any")
mySiteRpt.runReport();
```

setAccessibility

siteReportObj.setAccessibility(*criterion1* [, *criterion2*, ... , *criterionN*])

Report each file that has accessibility requirements (like use of frames) or accessibility problems (like no alt tag), as specified by arguments.

Availability

6.0

Parameters

criterion	*String*	Any of the following strings, which correspond to the checkboxes in the Accessibility tab of the Site Report dialog:

missingNoFramesTag	Report each file that does not provide a <NOFRAMES> element.
frameWOTitleAttr	Report each file in which a <frame> element does not provide a title attribute.
tableWOSummaryAttr	Report each file in which a <table> element does not provide a summary attribute.
tableWOCaptionTag	Report each file in which a <table> element does not provide a <caption> element.
missingNoScriptTag	Report each file that does not provide a <NOSCRIPT> element.
appletWOAltAttr	Report each file in which an <applet> element does not provide an alt attribute.
needsPlugins	Report each file that uses at least one plugin.
hasServerSideMap	Report each file that uses at least one server-side map.
imageWOAltAttr	Report each file in which an <img> element does not provide an alt attribute.

Returns

SiteReport	The called SiteReport object, configured as specified by the arguments to this method.

Example

To report on multiple criteria, pass a comma-separated list of arguments to this method.The following code generates a site report on common problems found in <table> elements.

```
var mySiteRpt = new JSXSiteReport;
mySiteRpt.setAccessibility ("missingNoFramesTag", "frameWOTitleAttr",
                            "tableWOSummaryAttr", tableWOCaptionTag);
mySiteRpt.setUseAnyOrAllCriteria("any")
mySiteRpt.runReport();
```

Custom Site Reports

You can use the `registerCustomReport` method of the SiteReport object to register your own function that GoLive is to call when generating a site report for that object.

registerCustomReport

siteReportObj`.registerCustomReport(`*functionName*`,` *reportString*`)`

Causes the *siteReportObj* object to call your *functionName* function when generating a site report.

Availability

6.0

Parameters

functionName	*String*	Custom function to be called.
reportString	*String*	A very short summary of the report action performed by functionName. This string appears in the report criteria summary at the top of the **Report Results** window.

Returns

Boolean	`true` indicates success.

Example

When the report runs, GoLive passes to each custom site reporting function a SiteReference object that corresponds to a file within the website. If the object meets the function's criteria, the custom function returns `true`; otherwise, it returns `false`.

A typical custom site report function looks like the following example.

```
function sampleSiteReport(itemRef)
{
        // itemRef is a SiteReference object that represents a
        // file within the site. This function does operations upon
        // itemRef to determine if it meets a particular set of
        // criteria. If itemRef passes the test, the function
        // returns true; otherwise, the function returns false.

        <do some operations on itemRef here>

        if (itemRefMeetsSelectionCriteria)
           return true;
        else
           return false;
}
```

Generating Reports

These functions manage site reporting behavior of the called SiteReport object.

setUseAnyOrAllCriteria

siteReportObj`.setUseAnyOrAllCriteria(`*any_or_all_condition*`)`

Specifies whether files included in site report must meet all or any *siteReportObj* criteria.

Availability

6.0

Parameters

any_or_all_condition	*String*	How the `runReport` method is to interpret the *siteReportObj* file-selection criteria. Specify this value by passing one of the following strings: `all` — Report only files which meet all criteria the *siteReportObj* specifies. `any` — Report files meeting any criterion the *siteReportObj* specifies.

Returns

Boolean	`true` indicates success.

runReport

siteReportObj`.runReport(`*show_window*`)`

Generate a report on the current site using the *siteReportObj* criteria.

Availability

6.0

Parameters

show_window	*String*	Pass `true` to display the **Site Report Results** window when this method completes.

Returns

SiteRefIterator	Each element of this collection is a SiteReference object representing a file that met the reporting criteria specified by the *siteReportObj* object.

NOTE: In order to get useful results, a website must be opened within GoLive when the report is run.

resetReport

siteReportObj`.resetReport()`

Discard all *siteReportObj* site-reporting criteria and custom site-reporting functions in order to reuse *siteReportObj* for another report.

Availability

6.0

Returns

SiteReport	The called SiteReport object, stripped of all previously-set criteria and all previously-registered custom site-reporting functions.

Description

When this function returns, the *siteReportObj* specifies no site reporting criteria and no custom site-reporting functions are registered with it.

saveQuery

siteReportObj`.saveQuery([`*path*`])`

Save the current *siteReportObj* reporting criteria to a site report query file at the *path* location.

Availability

6.0

Parameters

path	*String*	The location in which to create the site report query file, specified as a relative or fully-qualified path with filename.

Returns

Boolean	`true` indicates success.

Description

To display a dialog box in which the user can browse to specify a path, omit the *path* argument.

loadQuery

siteReportObj.`loadQuery([`*path*`])`

Loads report criteria from a previously-stored query file; optionally, displays a file-get dialog.

Availability

6.0

Parameters

path	*String*	The location from which to load the site report query file, specified as a relative or fully-qualified path with filename. Omit this argument to display a dialog box in which the user can browse to select this file.

Returns

Boolean	`true` indicates success.

addColumn

siteReportObj.`addColumn(`*name*`, `*width*`, `*alignment*`)`

Add a custom column to the **Report Results** window, using the specified *name*, *width*, and *alignment* (`"center"`, `"right"`, or `"left"`).

Availability

6.0

Parameters

name	*String*	Name of the custom column.
width	*String*	Width of the custom column, as a number of pixels.
alignment	*String*	Alignment of items in the custom column. Specify this value as one of the following strings:

`center`	Center-justify items that appear in this column.
`right`	Right-justify items that appear in this column.
`left`	Left-justify items that appear in this column.

Returns

Number	Column ID of the new custom column.
Null	Could not create custom column.

removeColumn

siteReportObj.`removeColumn(`*columnName*`)`

siteReportObj.`removeColumn(`*columnID*`)`

Remove the specified custom column from the **Report Results** window.

Availability

6.0

Parameters

columnName	*String*	String that is the name of the column to remove.
columnID	*Number*	Numeric ID of the column to remove. The `addColumn` method of the `SiteReport` object returns this value when it creates the custom column.

textArea Object

Availability

6.0

The `textArea` object manipulates text or HTML text in Layout view, allowing you to set/get selection, and get/set style information.

Acquiring TextArea Objects

Layout view always contains at least one `textArea` object. Whenever you insert text or boxes in Layout View, GoLive creates a TextArea object to encapsulate the text. The SDK also uses the TextArea object to represent the text content of a table cell.

- `mainTextArea` property of document Object

  ```
  document.mainTextArea // when document window is open
  ```

- `textArea` property of Markup Object

  markupObj.`textArea`

- `textArea` property of layout.tableCell Object

  markupObj.`layout.table.captionCell.textArea`

  markupObj.`layout.tableCell.textArea`

textArea Object Properties

`length`	*number*	Number of chars. Read-only.
`text`	*string*	The plain text of the text object. (Note: embedded boxes are displayed the placeholder string). Read-only.
`boxPlaceHolder`	*string*	The placeholder string used in the text property. Note that this string affects the value of the `length` property. The `findString` and `replaceString` methods operate on it as well.
`selectionStart`	*number*	Offset from the start of selection. Read-only
`selectionLength`	*number*	Number of selected chars. Read-only.
`styleSet`	*HTMLStyleSet*	The style set at the beginning of the current selection.Read-only.

textArea Object Functions

insert

markupObj`.textArea.insert(`*text*`)`

Inserts non-HTML *text* at the current position or replaces the current selection with it.

Availability

6.0

Parameters

text	*String*	Non-HTML text string this method inserts in the called TextArea object.

Returns

Boolean	`true` indicates success.

insertHTML

markupObj.textArea.insertHTML(*text*)

Insert the HTML *text* at the current position or replace the current selection with *text*.

Availability

6.0

Parameters

text	*String*	Text string to insert. This string may contain any mix of HTML tags and text.

Returns

Boolean	`true` indicates success.

insertChar

markupObj.textArea.insertChar(*aChar*)

Inserts the *aChar* character at the current position or replaces the current selection with *aChar*.

Availability

6.0

Parameters

aChar	*String*	Text character to insert.

Returns

Boolean	`true` indicates success.

insertNewLine

markupObj.textArea.insertNewLine()

Ends the current paragraph and starts a new one.

Availability

6.0

Returns

Boolean	`true` indicates success.

insertBox

markupObj.textArea.insertBox(*boxName*)

Adds a box object to Layout View at the current cursor position and sets its name property to the *boxName* value.

Availability

6.0

Parameter

boxName	*String*	JavaScript name of the box to insert. See this method's **Description** section for additional details.

Returns

Boolean	true indicates success.

Description

The *boxName* argument specifies the JavaScript name under which the new box appears in the GoLive environment. To insert a custom element, pass the value of its classid attribute. To insert an element the standard GoLive palettes provide, use one of the values Table 4.4 provides. This table also indicates the name that describes this element in the GoLive user interface, as well as the palette group that provides similar elements.

TABLE 4.4 Valid values of boxName argument

JavaScript Name	Palette Group
anchor	**Basic**
applet	
comment	
floatingbox	
image	
layoutgrid	
layouttextbox	
linebreak	
line	
marquee	
plugin	
swf	
quicktime	

JavaScript Name	Palette Group
real	**Basic** *(continued)*
svg	
javascript	
spacer	
table	
w3c	
form	**Forms**
submit	
reset	
button	
inputimage	
label	
textfield	
password	
textarea	
checkbox	
radiobutton	
popup	
listbox	
filebrowser	
hidden	
keygenerator	
fieldset	

remove

markupObj`.textArea.remove()`

Removes the current selection.

Availability

6.0

Returns

Boolean	`true` indicates success.

select

markupObj`.textArea.select(`*selBegin*`,` *numChars*`)`

Selects *selOffset* number of characters in the text, beginning with the character at the *selBegin* index position.

Availability

6.0

Parameters

selBegin	*Number*	The index position at which the selection begins; similar to the `document.selection.start` property. The first character is at index position 0.
numChars	*Number*	The number of characters to select.

Returns

Boolean	`true` indicates success.

selectParagraph

markupObj`.textArea.selectParagraph(`*num*`)`

Selects the specified paragraph.

Availability

6.0

Parameters

num	*Number*	Number of the paragraph to select. Paragraph 0 is the first paragraph in the called TextArea object.

Returns

Boolean	`true` indicates success.

selectall

markupObj`.textArea.selectall()`

Select all text.

Availability

6.0

Returns

Boolean	`true` indicates success.

deselect

markupObj`.textArea.deselect()`

Deselect the text.

Availability

6.0

Returns

Boolean	`true` indicates success.

cut

markupObj`.textArea.cut()`

Cut the currently-selected text.

Availability

6.0

Returns

Boolean	`true` indicates success.

copy

markupObj`.textArea.copy()`

Copy the currently-selected text.

Availability

6.0

Returns

Boolean	`true` indicates success.

paste

markupObj`.textArea.paste()`

Paste the current clipboard content at the current position.

Availability

6.0

Returns

Boolean	`true` indicates success.

applyStyle

markupObj`.textArea.applyStyle(`*styleName*`)`

markupObj`.textArea.applyStyle(`*styleSet*`)`

markupObj`.textArea.applyStyle(`*style*`)`

Applies the specified style to the text.

Availability

6.0

Parameters

styleName	*String*	Name of the style to apply.
styleSet	*HTMLStyleSet*	One or more styles to apply.
style	*HTMLStyle*	One style to apply.

Returns

Boolean	`true` indicates success.

Example

This method wraps the called textArea's text in a style tag. The Style need not be a known style. For example, evaluating the following

```
textArea.applyStyle("abc def=ghi")
```

returns

```
<abc def="ghi">selected text</abc>
```

GoLive quotes attribute values in the returned text automatically.

deapplyStyle

markupObj`.textArea.deapplyStyle(`*style*`)`

If the specified *style* is present in the current selection, this method removes the style's effect.

Availability

6.0

Parameters

style	*String*	Name of the style to deapply.

Returns

Boolean	`true` indicates success.

hasStyle

markupObj`.textArea.hasStyle(`*style*`)`

Indicates whether the specified *style* exists in the current selection.

Availability

6.0

Parameters

style	*String*	Name of the style for which this method tests.

Returns

number	**no**	The *style* is not in the current selection
number	**partly**	The *style* exists in the current selection, but does not terminate in the selection area.
number	**full**	The current selection contains the *style* completely.

addCurrentStyle

markupObj.textArea.addCurrentStyle([*styleSetName*])

Extracts the specified HTMLStyleSet object from the beginning of the current selection and adds it to the global list of HTML styles.

Availability

6.0

Parameters

styleSetName	*String*	Optional. Name of the style set to retrieve. If this argument is omitted, the method adds the current style to the global HTML style collection.

Returns

Boolean	`true` indicates success.

findString

markupObj.textArea.findString(*searchString* [, *startPosition*, *ignoreCase*])

markupObj.textArea.findString(*regExpression* [, *startPosition*, *ignoreCase*])

Returns the specified string's character offset from the beginning of the text.

Availability

6.0

Parameters

searchString	*String*	String to match.
regExpression	*RegExp*	Regular expression to match. This method overwrites the `lastIndex` and `globalFlag` properties of this object.
startPosition	*Number*	Optional. Character offset at which to begin searching. When this argument is omitted, the method begins searching at the beginning of the text.
ignoreCase	*Boolean*	Optional. Pass `true` to make case-insensitive comparisons. Pass `false` or omit this argument to make case-sensitive comparisons.

Returns

Number	Character offset of the found string or expression. A value of -1 means the string or expression was not found.

replaceString

markupObj`.textArea.replaceString(`*searchString*`,` *replaceString* `[,` *startPosition*`,` *ignoreCase*`])`

markupObj`.textArea.replaceString(`*regExpression*`,` *replaceString* `[,` *startPosition*`,` *ignoreCase*`])`

Replaces the *findString* text with the *replaceString* text.

Availability

6.0

Parameters

searchString	*String*	String to replace.
regExpression	*RegExp*	Regular expression object specifying string to replace. This method overwrites the `lastIndex` and `globalFlag` properties of this object.
replaceString	*String*	String to substitute in place of *searchString* or *regExpression*.
startPosition	*Number*	Optional. Character offset at which to begin searching. When this argument is omitted, the method begins searching at the beginning of the text.
ignoreCase	*Boolean*	Optional. Pass `true` to make case-insensitive comparisons. Pass `false` or omit this argument to make case-sensitive comparisons.

Returns

Number	Number of characters from the beginning of the called TextArea object at which the *findString* begins. A value of `-1` indicates that *findString* is not present in the called TextArea object.

scrollToText

markupObj`.textArea.scrollToText()`

Scroll **Layout View** as necessary to display the cursor's current position.

Availability

6.0

Returns

Boolean	`true` indicates success.

setCursor

markupObj`.textArea.setCursor(`*offset*`)`

Set the cursor's position to *offset* characters from the beginning of the text.

Availability

6.0

Parameters

offset	*Number*	Cursor's new position. Character `0` is the first character in the buffer.

Returns

Boolean	`true` indicates success.

setCursorParagraph

markupObj`.textArea.setCursorParagraph(`*paraNum*`)`

Position the cursor just before the *paraNum* paragraph.

Availability

6.0

Parameters

paraNum	*Number*	Number of the paragraph before which this method places the cursor. Paragraph `0` is the first paragraph in the buffer.

Returns

Boolean	`true` indicates success.

Translator Object

Availability

6.0

GoLive 6.0 creates a Translator object when it interprets a <jsxtranslator> tag in an extension's `main.html` file. The Translator object mimics the behavior of ASP code, using regular expressions to identify code it replaces with different code, either temporarily or permanently.

Your extension must implement the `translate` and `inspectTranslation` callback functions to support the document translation mechanism.

For more information, see "Document Source Translation" on page 204 of the *Extend Script SDK Programmer's Guide*.

Acquiring the Translator Object

The Translator object is available from the `translator` property of the Application object:

```
app.translator
```

Translator Object Functions

You can call these methods to replace, rewrite, or translate snippets on demand. The `translate` and `inspectTranslation` global functions also supply translation functionality.

translatorObj.`replaceSnippet` (*offsetNum*, *lengthNum*, *textString*)
: Replace the current source text at the given offset and length with the new text.

translatorObj.`getTranslatedSnippet` (*original*, *translation*, *urlList*, *classID*)
: Return *original* as correctly-wrapped *translation* data.

translatorObj.`translateSnippet` (*offset*, *length*, *text* `[,` *urlList*`]`))
: Replace the current source text at the given offset and length with translated text.

translatorObj.`rewriteSnippet` (*newCode*, *undoText*)
: Rewrite *xlatorObj* source code using *newCode* string as the *undoText* undoable action.

replaceSnippet

translatorObj.`replaceSnippet(`*offset*`,` *length*`,` *text*`)`

Replace the current source text at the given offset and length with the new text.

Availability

6.0

Parameters

offset	*Number*	Position at which to begin replacing characters in the source buffer with characters from the text string. Index number 0 is the first character in the text buffer.
length	*Number*	Number of characters to replace.
text	*String*	Replacement text string.

Description

The method does a simple text replacement without any insertion of extra tags. Use this method to insert or remove additional HTML before letting GoLive parse the source.

This function gives low-level access to the translation API. Its usage is flexible but error-prone, since you have to build the GoLive housekeeping elements around your translation by hand (you can use the utility function *getTranslatedSnippet* though). We recommend using the `translateSnippet` method.

getTranslatedSnippet

translatorObj.`getTranslatedSnippet(`*original*`,` *translation*`,` *urlList* `[,` *classID*`])`

This is a helper function that eases the work for constructing the correct translation wrapping information when not using *translateSnippet* but *replaceSnippet* instead.

Availability

6.0

Parameters

original	*String*	Original, pre-translation HTML source text.
translation	*String*	Your translation of *original* text.
urlList	*String*	Comma-separated list of URLS detected in translation.
classID	*String*	Optional. Class ID of the `<jsxtranslator>` element to use; if omitted, use the classid specified by the current Translator element.

Description

This is a helper function that eases the work for constructing the correct translation wrapping information when not using *translateSnippet* but *replaceSnippet* instead. You pass in the original snippet, your translation, the detected urls (comma separated) and (optional) your class id (when the class id is omitted the one used in the `<jsxtranslator>` element will be used instead). You get back the wrapped translation.

translateSnippet

translatorObj.translateSnippet(*offset*, *length*, *text* [, *urlList*])

Replace the current source text at the *offset* and *length* with translated *text*.

Availability

6.0

Parameters

offset	*Number*	Position at which to begin replacing characters in the source buffer with characters from the translated text string. Index number 0 is the first character in the text buffer.
length	*Number*	Number of characters to replace.
text	*String*	Replacement text string.
urlList	*String*	Optional. Comma-separated list of URLS detected in translation.

Description

The method inserts an *agl:translated* tag that contains the original source text in its *orig* attribute. The new, replaced text becomes the content of the *agl:translated* tag and is displayed by GoLive as a protected region. This tag is normally invisible; you can see it in the source code view in layout mode, however. Switching away from layout mode replaces the *agl:translated* tag with the original content again. The translated text is also never written into a file if direction=twoway.

Use the optional *urlList* argument it if the untranslated snippets your translator works on contain URLs that you detect and wish to bring under the control of GoLive. If you do, *urlList* contains a comma separated list of URLs; to inspect and change links, the items in this list must be in order.

Example

The following sample illustrates the replacement of the tag <*mytag*> with the text "Mock Content":

```
<agl:translated classid="myTagID" orig="%3cmytag%3e">
   Mock Content
</agl:translated>
```

rewriteSnippet

translatorObj`.rewriteSnippet(`*newCode*`,` *undoText*`)`

Rewrite the *xlatorObj* source code using the specified *newCode* string and register this action as the *undoText* undoable action.

Availability

6.0

Parameters

newCode	*String*	Position at which to begin replacing characters in the source buffer with characters from the translated text string. Index number 0 is the first character in the text buffer.
undoText	*String*	Number of characters to replace.

Description

The method calls the JavaScript method *translate()* afterwards to give the SDK developer a chance to update the replacement code according to the changes of the original code.

TreeNode Object

Availability

6.0

The TreeNode object provides an individual node in the hierarchy of nodes a Tree control (`<jsxcontrol type = "tree" ...>`) manages.

Whenever the user changes the selection or the value of the selected tree node, GoLive calls the `controlSignal` function. The `selection` property of the control passed to this function contains the TreeNode object that changed. If your `controlSignal` function changes the node it must call the node's `refresh` method to update the node's appearance.

Acquiring TreeNode Objects

dialogObj.*treeControlObj*.value.children[*n*]

controls[*treeName*].value.children[*n*]

TreeNode Object Properties

children	*TreeNodeCollection*	A Collection object to retrieve the child nodes one level below this node.
parent	*TreeNode* *TreeRoot*	The TreeNode or TreeRoot one level above this node.
checkedStyle	*Boolean*	A Boolean that if true, means the value property of this node is a Boolean and clicking the icon toggles the value and the icon displayed.
value	*Boolean* *String* *Null*	If checkedStyle is true, a Boolean corresponding to the check state. If checkedStyle is false, the key of the icon for this node or null if it has no icon.
text	*String*	The text displayed for this node.
userData	*String* *Number*	You can programmatically set this to anything you want. If is ignored in the operation of the control. It exists to allow you to attach any arbitrary data you like to this node.
uncheckedUpIcon	*String*	The key for the icon to display when value is false and the mouse is not depressed. Has no effect when checkedStyle is false.
uncheckedDownIcon	*String*	The key for the icon to display when value is false and the mouse is depressed. If null, the uncheckedUpIcon is used. Has no effect when checkedStyle is false.
checkedUpIcon	*String*	The key for the icon to display when value is true and the mouse is not depressed. Has no effect when checkedStyle is false.
checkedDownIcon	*String*	The key for the icon to display when value is true and the mouse is depressed. If null, the checkedUpIcon is used. Has no effect when checkedStyle is false.

TreeNode Object Functions

Functions of the TreeNode object

- manipulate its icon.
- edit its popup menu.
- add and remove child nodes.

addChoice

treeNodeObj`.addChoice(`*key, text*`)`

Appends the *text* choice to the *treeNodeObj* popup and associates the *key* value with this choice.

Availability

6.0

Parameters

key	*String*	The value to which treeNodeObj is set when the user selects this popup item.
text	*String*	Text the popup item displays to the user.

Description

This method associates a key with a value and an icon to be shown for that value. The popup menu will display text next to that icon. If any choices have been added, clicking on the icon of this node will display a popup menu. If this node has no choices, the value can only be set programmatically. This assumes that checkedStyle is false. If it is true, the list of choices is not used and the value can be toggled by clicking the icon.

removeChoices

treeNodeObj`.removeChoices()`

Clears the list of choices.

Availability

6.0

Description

If the *treeNodeObj*`.checkedStyle` property is `false`, the value of the node can only be set programmatically.

insertChild

treeNodeObj.`insertChild(`*child, index*`)`

Places the *child* node in *treeRootObj* at the *index* position.

Availability

6.0

Parameters

child	*TreeNode*	Child node to add to *treeRootObj*.
index	*Number*	Optional. Number of the node under which this method inserts the child node. Index position 0 is the first child node in the tree. An index of -1 (or if the argument is missing or invalid) specifies the last child.

Description

Index position 0 specifies the first child node. An index of `-1` (or if the argument is missing or invalid) specifies the last child.

There are never any empty indexes between children. If child already has a parent, it is removed from its parent before being inserted, even if the new parent and old parent are identical. If child was a node in another tree, the icon that corresponds to its value or to any of its checked-state icon keys is determined by the icons that were added to the new tree.

appendChild

treeNodeObj.`appendChild(`*child*`)`

Appends the *child* node to the bottom of the *treeNodeObj* node.

Availability

6.0

Parameters

child	*TreeNode*	Child node to append to the bottom of the *treeNodeObj* tree.

Description

Same as

```
treeNodeObj.insertChild(child, -1);
```

removeChildren

treeNodeObj.`removeChildren (`*startIndex*`, `*endIndex*`)`

Removes from *treeNodeObj* all child nodes having index values in the range from *startIndex* to *endIndex*, inclusive.

Availability

6.0

Parameters

startIndex	*Number*	Lower boundary of range of index values on which this method operates.
endIndex	*Number*	Upper boundary of range of index values on which this method operates.

Description

If the *endIndex* argument is missing, `-1`, or a number greater than the index of the last child, this method removes all children having indexes of *startIndex* or above. If no arguments are provided, all children are removed. If *startIndex* > *endIndex* or *startIndex* is greater than the index of the last child, no children are removed. Arguments that cannot be converted to non-negative integers are invalid; such arguments generate a runtime error. After removing nodes, this method removes empty indexes between the remaining child nodes and re-indexes the remaining children over a contiguous range of index values.

removeChild

treeNodeObj.`removeChild(`*index*`)`

Removes from *treeNodeObj* the node at the specified *index* position.

Availability

6.0

Parameters

index	*Number*	Optional. Number of the node under which this method removes the child node. Index position 0 is the first child node in the tree. An index of -1 (or if the argument is missing or invalid) specifies the last child.

Description

Equivalent to

treeNodeObj.`removeChildren(`*index*`, `*index*`)`

refresh

treeNodeObj`.refresh ()`

Requests a repaint of this node and its children.

Availability

6.0

Description

To reduce flickering, the tree control does not redraw itself automatically when you insert child nodes, remove child nodes or change the values of a node's properties. After performing such operations, you must call the affected `TreeNode` object's `refresh` method yourself to make the control's appearance reflect its current state.

TreeRoot Object

Availability

6.0

The `TreeRoot` object provides access to the individual nodes of a Tree Control.

Acquiring TreeRoot Objects

The `Tree` object's `value` property provides the `TreeRoot` object.

```
var treeRootObj = myTreePaletteName.myTreeControlName.value;
treeRootObj = myTreePaletteName.myTreeControlName.value;
```

TreeRoot Object Properties

`children`	*TreeNodeCollection*	Child nodes representing the top-level items in the Tree control.
`parent`	`null`	Always null. When traversing the nodes of the TreeRoot object, you can use this property to determine when you are at the root without needing to check the type of the object.
`editableText`	*Boolean*	If true, the user can select and edit the text.

TreeRoot Object Functions

Functions of the `TreeRoot` object can add, remove, and manipulate the individual nodes of a Tree control and the icons that represent them. There are never any empty indexes between children; when you remove nodes, GoLive re-indexes the remaining nodes contiguously.

To reduce flickering, the tree control does not redraw itself automatically when you insert child nodes, remove child nodes or change the values of a node's properties. After performing such operations, you must call the affected `TreeNode` object's `refresh` method yourself to make the control's appearance reflect its current state.

addIcon

treeRootObj`.addIcon(`*key*`,`*picture*`)`

Creates from *picture* an icon in *treeRootObj* which can be referenced by the *key* name.

Availability

6.0

Parameters

key	*String*	JavaScript name of the icon this method adds.
picture	*String*	URL to picture from which to create the icon.
	Picture	Picture from which to create the icon.

Description

When the *picture* argument is a `String`, it is a URL to a picture to use as the icon. The control disposes of this picture when it is no longer used.

When the *picture* argument is a `Picture`, this method copies the picture and disposes of the copy. The control does not hold a reference to the object passed as this argument, nor does it dispose of this object.

The key this method associates with the icon is arbitrary and can be any value, including any numeric values for which the `toString` method returns a unique String.

createNode

treeRootObj`.createNode()`

Creates a new, "blank" `TreeNode` object

Availability

6.0

Returns

TreeNode	After setting this object's properties, you can add it to the children of the called *treeRootObj* or any `TreeNode` object.

insertChild

treeRootObj.insertChild (*child* [, *index*])

Places the *child* node in *treeRootObj* as a child of the *index* node.

Availability

6.0

Parameters

child	*TreeNode*	Child node to add to *treeRootObj* at the specified *index* position.
index	*Number*	Optional. Number of the node under which this method inserts the child node. Index position 0 is the first child node in the tree. An index of -1 (or if the argument is missing or invalid) specifies the last child.

Description

Index position 0 specifies the first child node.An index of `-1` (or if the argument is missing or invalid) specifies the last child.

There are never any empty indexes between children. If child already has a parent, it is removed from its parent before being inserted, even if the new parent and old parent are identical. If child was a node in another tree, the icon that corresponds to its value or to any of its checked-state icon keys is determined by the icons that were added to the new tree.

appendChild

treeRootObj.appendChild(*child*)

Appends the *child* node to the bottom of the *treeRootObj* tree.

Availability

6.0

Parameters

child	*TreeNode*	Child node to append to the bottom of the *treeRootObj* tree.

Description

Same as

```
treeRootObj.insertChild(child, -1);
```

removeChildren

treeRootObj`.removeChildren (`*startIndex*`, `*endIndex*`)`

Removes from *treeRootObj* all child nodes having index values in the range from *startIndex* to *endIndex*, inclusive.

Availability

6.0

Parameters

startIndex	*Number*	Lower boundary of range of index values on which this method operates.
endIndex	*Number*	Upper boundary of range of index values on which this method operates.

Description

If the *endIndex* argument is missing, `-1`, or a number greater than the index of the last child, this method removes all children having indexes of *startIndex* or above. If no arguments are provided, all children are removed. If *startIndex* > *endIndex* or *startIndex* is greater than the index of the last child, no children are removed. Arguments that cannot be converted to non-negative integers are invalid; such arguments generate a runtime error. After removing nodes, this method removes empty indexes between the remaining child nodes and re-indexes the remaining children over a contiguous range of index values.

removeChild

treeRootObj`.removeChild (`*index*`)`

Removes from *treeRootObj* the node at the specified *index* position.

Availability

6.0

Parameters

index	*Number*	Optional. Number of the node under which this method removes the child node. Index position 0 is the first child node in the tree. An index of -1 (or if the argument is missing or invalid) specifies the last child.

Returns

Boolean	`true` indicates success.

Description

Equivalent to

treeRootObj`.removeChildren(`*index*`, `*index*`)`

Undo Object

Availability

5.0, 6.0

GoLive provides built-in undo support for dropping and resizing boxes. The user can undo or redo such operations by choosing the **Undo** or **Redo** item from the **Edit** menu. To undo any other operations involving your custom element, you must implement code that provides such behavior.

You can add undo support for any operation that does not cause GoLive to reparse. To undo an operation, you use a saved value, such as a name, to retrieve a previously-changed markup object. Reparsing the document discards the markup object the saved value represents. Thus, any operation that reparses the document cannot be undone.When Layout view is open,

- Operations involving only “pure SDK” elements can be undone completely, because they do not cause automatic reparsing.
- Most operations involving non-SDK elements cannot be undone.

  Even if your particular extension does not cause Layout view to reparse the document, the user or another extension may cause a reparse.

Acquiring Undo Objects

The body of the method that performs an operation to undo creates an undo object, initializes it, and returns it to GoLive. The SDK passes this undo object to the extension’s `undoSignal` method whenever the user issues the command to do, undo, or redo the operation associated with this undo object.

```
undoSignal (undo, action)
undoLink (undo, action)
```

Undo Object Properties

The undo object provides no properties of its own. You must add as its properties any data your extension requires to perform an operation or reverse its effects.

yourPropertyName *String* Add as strings any properties required to do, undo, or redo this action.

You can add properties to the Undo object simply by declaring and assigning them, as the following example does:

```
// create empty Undo object
var undo = document.createUndo ("operationName")
// add properties
undo.myProperty = myValue;
undo.myOtherProerty = myOtherValue;
undo.myThirdProp = yetAnotherValue;
// submit the undo object to GoLive
undo.submit();
```

Once you've finished adding properties to the undo object, pass it to GoLive by calling the undo object's `submit` method.

Undo Object Functions

The Undo object provides only one function of its own: the `submit` method, which passes an undo object to GoLive. Important Undo behaviors are provided by the `undoSignal` event-handling function that the Extend Script SDK calls as needed to process undo actions.

NOTE: To support undo functionality, your extension must provide the global event-handling methods that the Undo section of Chapter 6, "Event-Handling Functions," describes.

submit

undoObj`.submit()`

Returns an initialized undo object to GoLive.

Availability

5.0, 6.0

Description

Returns an initialized undo object to GoLive. GoLive adds this object to the document's History list and immediately passes the undo object to the `undoSignal` function with an action code of `1`. This action code specifies that the `undoSignal` method is to perform the operation for the first time.

website Object

Availability

6.0

Manipulates a site that is currently open in the GoLive design environment. The corresponding site window need not be frontmost, but it must be open. You can use this object and its methods to clean up a site; create new pages and folders in a site; rescan a site; get/set selection in site window; add custom columns to the site window; and export a site.

NOTE: The `document.site` property is not a `Website` object, it is a reference to the root folder of a site.

Acquiring website Objects

The `website` and `websites` Global Properties provide access to `Website` objects.

```
website
websites[number]
```

The website property of the document Object provides a `website` object if a site uses the document.

```
document.website
documentObj.website
```

For example, all three of the following lines of code retrieve the same `website` object, which provides access to the currently-active site.

```
var theActiveSite = website;
theActiveSite = document.website;
theActiveSite = document.websites[number];
```

Website Object Properties

`sitename`	*String*	The name of the site without the .site extension. Read-only.
`root`	*SiteReference*	The root directory of the site. (The same object `document.site` returns.) Read-only.
`homePage`	*SiteReference*	The home page of the site. Assigning a SiteReference object to this property makes the document it references into the new home page.
`document`	*Document*	Returns the Document object that represents the site.
`cleanupSettings`	*cleanupSettings*	Argument to the `website.cleanup()` method.
`exportSettings`	*ExportSiteSettings*	Argument to the `website.exportSiteSettings()` method.

Website Object Functions

Methods of the `website` object manipulate files, selections, and custom column content in an open site window. Whenever possible, this object performs its tasks as undoable actions; to undo or redo the most recent site-window change, simply call the website object's `undo` or `redo` methods.

websiteObj.`selectAllFiles()`
Selects all files in the files tab of the *websiteObj* site window.

websiteObj.`deselectAllFiles()`
Deselects all files in the **Files** tab of the *websiteObj* site window.

websiteObj.`selectFiles`(*siteReferenceObj*)
websiteObj.`selectFiles`(*siteRefIteratorObj*)
websiteObj.`selectFiles`(*1stURL [, 2ndURL, ... , nthURL]*)
Selects the specified files in the **Files** tab of the *websiteObj* site window.

websiteObj.`getSelectedFiles()`
Returns a `SiteRefIterator` that holds references to all files selected in the **Files** tab of the *websiteObj* site window.

websiteObj.`createFolder`(*URL*, [*fixDuplicateNames*, *showInSiteWindow*])
Creates a folder in the *URL* location.

websiteObj.`createFile`(*URL*, [*fixDuplicateNames*, *showInSiteWindow*)
Creates a generic HTML file in the *URL* location.

websiteObj.createFileFromStationery (*fileURL*, *stationeryURL*
[, *fixDuplicateNames*,
showInSiteWindow])

Use stationery at *stationeryURL* to create a HTML file at the *fileURL* location.

websiteObj.createFileFromTemplate (*fileURL*, *templtURL*
[, *fixDuplicateNames*,
showInSiteWindow])

Creates an HTML file from *templtURL* at the *fileURL* location.

websiteObj.refreshFileView()
websiteObj.refreshFileView(*siteRefObj*)
websiteObj.refreshFileView(*SiteRefIterator*)
websiteObj.refreshFileView(*URL [, anotherURL, ... , nthURL]*)

Updates the Files tab of the *websiteObj* site window.

websiteObj.cleanup([*cleanupSettings*])

Equivalent to choosing the **Site>Cleanup Site** menu item in GoLive.

websiteObj.exportSite(*folderURL [, exportSettings]*)

Exports the *websiteObj* site into the *folderURL* folder.

websiteObj.undo()

Undo the most recent change to the *websiteObj* site window.

websiteObj.redo()

Redo the most recent change to the *websiteObj* site window.

websiteObj.canUndo()

Returns `true` if the SDK can undo the last change to the *websiteObj* site.

websiteObj.canRedo()

Returns `true` if the SDK can redo the last change to the *websiteObj* site.

websiteObj.undo (*name*, *prefName* [, *width*, *alignment*])

Adds the *name* column to display the *prefName* named property without callbacks.

websiteObj.addColumn(*name* [, *width*, *alignment*])

Adds a custom column to the Files tab in the Site window of the *websiteObj* site.

websiteObj.insertPrefColumn(*name*, *prefName* [, *index*, *width*, *alignment*])

Inserts the *name* column to display the *prefName* named property without callbacks.

websiteObj.insertColumn(*name* [, *index*, *width*, *alignment*])

Inserts a custom column in the Site window.

websiteObj.removeColumn(*name*)
websiteObj.removeColumn(*columnID*)

Removes the specified column from the Files tab of the *websiteObj* Site window.

selectAllFiles

websiteObj.`selectAllFiles()`

Selects all files in the files tab.

Availability

6.0

Returns

Boolean	`true` indicates success.

deselectAllFiles

websiteObj.`deselectAllFiles()`

Deselects all files in the **Files** tab of the *websiteObj* window.

Availability

6.0

Returns

Boolean	`true` indicates success.

selectFiles

websiteObj.`selectFiles(`*fileRef1* `[,` *fileRef2*`, ... ,` *fileRefN*`])`

websiteObj.`selectFiles(`*SiteRefIterator*`)`

websiteObj.`selectFiles(`*URL* `[,` *anotherURL*`, ... ,` *nthURL*`])`

Selects the specified files in the files tab of the *websiteObj* window.

Availability

6.0

Parameters

fileRef	*SiteReference*	A file to select.
SiteRefIterator	*SiteRefIterator*	Collection of files to select.
URL	*String*	Fully-qualified or site-relative URL to a file to select.

Returns

SiteRefIterator	Group of SiteReference objects representing files that were selected successfully in the Files tab of the site window of the *websiteObj* site.

getSelectedFiles

websiteObj.`getSelectedFiles()`

Returns references to of all selected files in the files tab.

Availability

6.0

Parameters

relURL	*String*	Relative path to "`aglfmi`" files.

Returns

SiteRefIterator	Group of SiteReference objects representing files currently selected in the Files tab of the site window of the *websiteObj* site.

Example

```
var selectedFiles = website.getSelectedFiles();
var len = selectedFiles.length();
if (len == 0){
   return;
}
var aFile = selectedFiles.first();
myFunction(aFile);
if (len > 1){
   for (var i = 1; i < len; i++)
   {
      aFile = selectedFiles[i];
      myFunction(aFile)
   }
```

createFolder

websiteObj.createFolder(*URL* [, *fixDuplicateNames*, *show*]);

Creates a folder in the *URL* location; optionally, appends a numeric suffix to duplicate folder names or displays the new folder and its ancestors in the site window.

Availability

6.0

Parameters

URL	*String*	Location in which to create the new folder, specified as a fully-qualified or site-root relative URL.
fixDuplicateNames	*Boolean*	Optional. Pass `true` to append a numeric value to the end of a duplicate filename to make it unique. Omitting this argument is the same as passing `true`.
show	*Boolean*	Optional. Pass `true` to disclose all parent folders and select the new folder in the site window. Omitting this argument is the same as passing `true`.

Returns

SiteReference	Reference to the new folder.
Nothing	Could not create the folder.

createFile

websiteObj.`createFile(`*URL* `[,` *fixDuplicateNames*`,` *show*`])`

Creates a generic HTML file in the *URL* location.

Availability

6.0

Parameters

URL	*String*	Fully-qualified or relative path to new file's location. Terminate this path with ".html" or another appropriate extension to avoid a warning icon in the site window.
fixDuplicateNames	*Boolean*	Optional. Pass `true` to append a numeric value to the end of a duplicate filename to make it unique. Omitting this argument is the same as passing `true`.
show	*Boolean*	Optional. Pass `true` to disclose all parent folders and select the new folder in the site window. Omitting this argument is the same as passing `true`.

Returns

SiteReference	Reference to the new file.
Nothing	Could not create the file.

createFileFromStationery

websiteObj`.createFileFromStationery(`*fileURL*`,` *stationeryURL*
`[,` *fixDuplicateNames*`,` *show*`])`

Uses the GoLive stationery file at *stationeryURL* to create a HTML file at the *fileURL* location.

Availability

6.0

Parameters

fileURL	*String*	Fully-qualified or relative path to new file's location.
stationeryURL	*String*	Fully-qualified or relative path to GoLive stationery file's location.
fixDuplicateNames	*Boolean*	Optional. Pass `true` to append a numeric value to the end of a duplicate filename to make it unique. Omitting this argument is the same as passing `true`.
show	*Boolean*	Optional. Pass `true` to disclose all parent folders and select the new folder in the site window. Omitting this argument is the same as passing `true`.

Returns

SiteReference	Reference to the new file.
Nothing	Could not create the file.

createFileFromTemplate

websiteObj.`createFileFromTemplate`(*fileURL*, *templtURL*
`[`, *fixDuplicateNames*, *show*`]`)

Uses the GoLive template file at *templtURL* to create a HTML file at the *fileURL* location.

Availability

6.0

Parameters

fileURL	*String*	Fully-qualified or relative path to new file's location.
templtURL	*String*	Fully-qualified or relative path to GoLive template file's location.
fixDuplicateNames	*Boolean*	Optional. Pass `true` to append a numeric value to the end of a duplicate filename to make it unique. Omitting this argument is the same as passing `true`.
show	*Boolean*	Optional. Pass `true` to disclose all parent folders and select the new folder in the site window. Omitting this argument is the same as passing `true`.

Returns

SiteReference	Reference to the new file.
Nothing	Could not create the file.

cleanup

websiteObj.cleanup([*cleanupSettings*])

Equivalent to choosing the **Site>Cleanup Site** menu item in GoLive 6.Uses the site's preference settings when called without an argument.

Availability

6.0

Parameters

cleanupSettings	*cleanupSettings*	Boolean properties of this object specify the actions this method takes when it cleans up a site. Set a property to `true` to enable the feature it represents. A newly-constructed `cleanupSettings` object not associated with a site holds the following default values:

`rescanRoot`	`true`
`removeFiles`	`false`
`showRemoveList`	`true`
`removeExternals`	`false`
`removeColors`	`false`
`removeFontsets`	`false`
`addFiles`	`true`
`showAddList`	`true`
`addExternals`	`true`
`addColors`	`true`
`addFontsets`	`true`
`showDialog`	`false`

For more information, see "website.cleanupSettings Object" on page 254.

Returns

Boolean	`true` indicates success.

Example

The following code provides all default site cleanup behaviors except the display of lists of items the `cleanup` method added or removed:

```
// don't display add/remove lists
website.cleanupSettings.showRemoveList = false;
website.cleanupSettings.showAddList = false;
website.siteSettings(website.cleanupSettings);
```

exportSite

websiteObj.`exportSite(`*folderURL* `[,` *exportSettings*`])`

Exports the *websiteObj* site into the *folderURL* folder.

Availability

6.0

Parameters

URL	*String*	Folder in which to place the exported site, specified as a fully-qualified URL.
exportSettings	*ExportSiteSettings*	Boolean properties of this object specify the actions this method takes when it exports the site. Set a property to `true` to enable the feature it represents. A newly-constructed ExportSiteSetttings object not associated with a site holds the following default values:

`honorPublishGroups`	*Boolean*	`true`
`honorPublishPages`	*Boolean*	`true`
`publishOrphanFiles`	*Boolean*	`false`
`publishOnlyReferenced`	*Boolean*	`true`
`showDialog`	*Boolean*	`true`
`siteStructure`	*Enum*	`siteStructure.SITE`
`pagesFolder`	*String*	current Site prefs value
`mediaFolder`	*String*	current Site prefs value
`orphanFilesFolder`	*String*	current Site prefs value
`stripGoLiveTags`	*Boolean*	`false`
`stripComments`	*Boolean*	`false`
`stripSpaces`	*Boolean*	`false`

For more information, see "website.exportSettings Object" on page 256.

Returns

Boolean	`true` indicates success.

Description

The URL has to be fully qualified for this call, as the folder most likely will not be part of the site. Uses the site's preference settings when called without a second parameter.

Example

If you want to export the site with all options set to the current site settings except the site

structure option, you would do the following:

```
var mySettings = website.exportSettings;
mySettings.siteStructure = siteStructure.FLAT;
website.exportSite(mySettings);
```

If you want to export the site with all options set to default except the show dialog option, you would do the following:

```
var mySettings = new JSXExportSiteSettings();
mySettings.showDialog = false;
website.exportSite(mySettings);
```

addPrefColumn

websiteObj.`addPrefColumn`(*name*, *prefName* [, *width*, *alignment*])

Adds to the GoLive site files tab a *name* column that displays the *prefName* named property of a `SiteReference` object.

Availability

6.0

Parameters

name	*String*	Label to display at the top of the new column.
prefName	*String*	Named `SiteReference` property displayed as column content.
width	*Number*	Optional. Width of the column, as a number of pixels.Valid values with which this method can create a column range from 50 to 320. If this argument is omitted, GoLive uses a 50-pixel default width.
alignment	*String*	Optional. Alignment of the column's content. Valid values are the `left`, `right`, and `center` strings. If this argument is omitted, GoLive defaults to left alignment.

Returns

Integer	Column ID of the newly-inserted column or `-1` to indicate failure.

Description

Preferred over the `addColumn` method for better performance due to the avoidance of callbacks.

addColumn

websiteObj.`addColumn(`*name* `[,` *width*`,` *alignment*`])`

Adds to the GoLive site files tab a *name* column that uses the column callback mechanism to display its custom content.

Availability

6.0

Parameters

name	*String*	Label to display at the top of the new column.
width	*Number*	Optional. Width of the column, as a number of pixels.Valid values with which this method can create a column range from 50 to 320. If this argument is omitted, GoLive uses a 50-pixel default width.
alignment	*String*	Optional. Alignment of the column's content. Valid values are the `left`, `right`, and `center` strings. If this argument is omitted, GoLive defaults to left alignment.

Returns

Integer	Column ID of the newly-added column or `-1` to indicate failure.

Description

The custom column this method creates calls your extension's onSiteColumnContentCallback function to retrieve its content. If the user selects the column, this column calls the onSiteColumnCompareCallback function to sort its data.

insertPrefColumn

websiteObj.`insertPrefColumn(`*name*`,` *prefName* `[,` *index*`,` *width*`,` *alignment*`])`

Inserts in the GoLive site files tab at position *index* a *name* column that displays the *prefName* named property of a `SiteReference` object.

Availability

6.0

Parameters

name	*String*	Label to display at the top of the new column.
prefName	*String*	Named `SiteReference` property displayed as column content.
index	*Number*	Optional. Position at which to insert the new column. Valid values range from 1 to the number of columns present in the Site window. If this argument is omitted, GoLive uses 1 as the default index position, meaning that the column is inserted just after the *name* column. This value must be greater than 0, because the *name* column, which occupies position 0, cannot be moved.
width	*Number*	Optional. Width of the column, as a number of pixels.Valid values with which this method can create a column range from 50 to 320. If this argument is omitted, GoLive uses a 50-pixel default width.
alignment	*String*	Optional. Alignment of the column's content. Valid values are the `left`, `right`, and `center` strings. If this argument is omitted, GoLive defaults to left alignment.

Returns

Integer	Column ID of the newly-inserted column or `-1` to indicate failure.

Description

Using the `insertPrefColumn` method usually results in better performance than using the `insertColumn` method, because the `insertPrefColumn` method does not call back into your JavaScript code.

insertColumn

websiteObj.`insertColumn(`*name* `[,` *index*`,` *width*`,` *alignment*`])`

Availability

6.0

Parameters

name	*String*	Label to display at the top of the new column.
index	*Number*	Optional. Position at which to insert the new column. Valid values range from 1 to the number of columns present in the Site window. If this argument is omitted, GoLive uses 1 as the default index position, meaning that the column is inserted just after the *name* column. This value must be greater than 0, because the *name* column, which occupies position 0, cannot be moved.
width	*Number*	Optional. Width of the column, as a number of pixels.Valid values with which this method can create a column range from 50 to 320. If this argument is omitted, GoLive uses a 50-pixel default width.
alignment	*String*	Optional. Alignment of the column's content. Valid values are the `left`, `right`, and `center` strings. If this argument is omitted, GoLive defaults to left alignment.

Returns

Integer	Column ID of the newly-inserted column or `-1` to indicate failure.

Description

The custom column this method creates calls your extension's onSiteColumnContentCallback function to retrieve its content. If the user selects the column, this column calls the onSiteColumnCompareCallback function to sort its data.

removeColumn

websiteObj`.removeColumn(`*name*`)`

websiteObj`.removeColumn(`*columnID*`)`

Removes the specified column from the Files tab of the *websiteObj* Site window.

Availability

6.0

Parameters

name	*String*	JavaScript name of column to remove.
columnID	*Number*	Numeric ID of the column to remove. The `addColumn` method returns this value when it creates the custom column.

Description

This method only removes custom columns added by the SDK. It cannot remove columns built into the Site window.

After removing the column, GoLive calls the `onSiteColumnIDCallback` method. If your extension creates more than one custom column, you must implement this method, which updates your extension's references to the column IDs of the remaining columns.

canUndo

websiteObj`.canUndo()`

Returns `true` if GoLive can undo the last change to the *websiteObj* site.

Availability

6.0

Returns

Boolean	`true` indicates that the SDK can reverse the most recent change to the *websiteObj* site.

canRedo

websiteObj.canRedo()

Returns `true` if the SDK can reverse the effects of the last Undo command in the *websiteObj* site.

Availability

6.0

Returns

Boolean	`true` indicates that the SDK can reverse the effects of the most recent *websiteObj*.undo() command.

undo

websiteObj.undo()

Undoes the most recent change to the *websiteObj* site window.

Availability

6.0

Returns

Boolean	`true` indicates success.

redo

websiteObj.redo()

Reverses the effects of the most recent *websiteObj*.undo() method.

Availability

6.0

Returns

Boolean	`true` indicates success.

refreshFileView

websiteObj.refreshFileView()

websiteObj.refreshFileView(*siteReference*)

websiteObj.refreshFileView(*siteRefIterator*)

websiteObj.refreshFileView(*URL* [, *anotherURL*, ... , *nthURL*])

Updates folders or lists of folders in the Files tab of the *websiteObj* site window.

Availability

6.0

Parameters

URL	*String*	Relative path to a specific file to update.
siteReference	*SiteReference*	*SiteReference* to a specific file to update
siteRefIterator	*SiteRefIterator*	Group of files to update

Returns

Boolean	`true` indicates success.

Description

Re-scans the specified folder for single user sites, gets the actual data from the server for workgroup sites. Rescans the root if called without an argument. If your extension creates new files, the Site window may not display them until after you call this method.

website.cleanupSettings Object

Availability

6.0

The `cleanupSettings` object encapsulates Boolean values that enable or disable features of the `cleanup` method of the website Object.

The `cleanupsettings` object serves as a parameter block you use to set one or more behaviors of the `cleanup` method. Upon its creation, the properties of this object hold the current values that govern the actions of the `cleanup` method; thus, to configure this object, set the values of properties you need to alter, and allow the others to retain their default values.

Acquiring cleanupSettings Objects

You can obtain a `cleanupSettings` object from the `cleanupSettings` property of the Website object.

```
website.cleanupSettings // from the site itself
document.website.cleanupSettings// from frontmost doc's site
```

cleanupSettings Object Properties

When created without a site, this object provides the default values Table 4.5 shows. You can set a property's value to `true` to request the behavior it represents, or set it to `false` to suppress that behavior.

TABLE 4.5 Default values of cleanupSettings properties

Property	Default Value	Description
`rescanRoot`	`true`	If `true`, the `cleanup` method rescans the site's root folder before cleaning up the site.
`removeFiles`	`false`	If `true`, the `cleanup` method removes files the site does not reference.
`showRemoveList`	`true`	If `true`, the `cleanup` method displays a list in which the user confirms or denies file deletions before they occur.
`removeExternals`	`false`	If `true`, the `cleanup` method removes references to files outside the site folder.
`removeColors`	`false`	If `true`, the `cleanup` method clears the site's color preferences.
`removeFontsets`	`false`	If `true`, the `cleanup` method clears the site's fontset preferences.
`addFiles`	`true`	If `true`, the `cleanup` method adds to the site window all files present in the site folder.
`showAddList`	`true`	If `true`, the `cleanup` method displays a list in which the user confirms or denies the addition of new files to the site.
`addExternals`	`true`	If `true`, the `cleanup` method copies external files into the site folder and updates links to reference the local copies.
`addColors`	`true`	If `true`, the `cleanup` method adds to the site's color preferences any color the site uses.
`addFontsets`	`true`	If `true`, the `cleanup` method adds to the site's fontset preferences any fontset the site uses.

website.exportSettings Object

Availability

6.0

The `exportSettings` object encapsulates Boolean properties that configure features of the `website.exportSite()` method.

Acquiring an exportSettings Object

There are several ways to obtain an `exportSettings` object:

- You can construct a new `exportSettings` object whenever you need one.

```
new JSXExportSiteSettings; // new operator
JSXExportSiteSettings(); // constructor function
```

- The `exportSettings` property of the `website` object also provides an `exportSettings` object.

```
website.exportSettings;
```

- Using a document object to gain access to the Website object, you can get an `exportSettings` object as the following line of code does.

```
document.website.exportSettings;
```

exportSettings Object Properties

When created without a site, this object provides the default values Table 4.6 shows. You can set a property's value to `true` to request the behavior it represents, or set it to `false` to suppress that behavior.

TABLE 4.6 Default values of exportSettings properties

honorPublishGroups	*Boolean*	true	
honorPublishPages	*Boolean*	true	
publishOrphanFiles	*Boolean*	false	
publishOnlyReferenced	*Boolean*	true	
showDialog	*Boolean*	true	
siteStructure	enum	siteStructure.SITE	Other valid values are: siteStructure.SPLITPAGESANDMEDIA siteStructure.FLAT
pagesFolder	*String*	Site prefs	Fully-qualified URL to folder that is to contain exported pages. Used only when the siteStructure property is set to siteStructure.SPLITPAGESANDMEDIA
mediaFolder	*String*	Site prefs	Fully-qualified URL to folder that is to contain exported media. Used only when the siteStructure property is set to siteStructure.SPLITPAGESANDMEDIA
orphanFilesFolder	*String*	Site prefs	
stripGoLiveTags	*Boolean*	false	
stripComments	*Boolean*	false	
stripSpaces	*Boolean*	false	

You can set the siteStructure property's value to any of the siteStructure.SITE, siteStructure.SPLITPAGESANDMEDIA, or siteStructure.FLAT values; for example,

```
// specify pages separate from media
website.exportSettings.siteStructure =
                        "siteStructure.SPLITPAGESANDMEDIA";
website.exportSettings.pagesFolder = "mydisc/mysite/mypages/"
website.exportSettings.mediaFolder = "mydisc/mysite/mymedia/"
```

The SDK uses the pagesFolder and mediaFolder folder names only when the siteStructure property is set to siteStructure.SPLITPAGESANDMEDIA.

For a code example, see "exportSite" on page 247.

website.exportSettings.siteStructure Object

Availability

6.0

The `siteStructure` object encapsulates information about the structure of the site, such as whether it is flat, whether it is based on a site file, and whether it uses pages or media not contained in the site folder.

Acquiring the siteStructure Object

Get the `siteStructure` object from the `siteStructure` property of the ExportSettings object. The ExportSettings object is available from the `website` and `websites` global variables.

```
website.exportSettings.siteStructure
websites[number].exportSettings.siteStructure
```

siteStructure Object Properties

The `siteStructure` object holds one of only three enumerated values:

`siteStructure.SITE`	Site structure is defined by a GoLive `.site` file.
`siteStructure.SPLITPAGESANDMEDIA`	Site retrieves pages and media from designated folders outside of the site folder.
`siteStructure.FLAT`	No structure. All contents of site reside in one folder.

You can set the `siteStructure` property to any of the `siteStructure.SITE`, `siteStructure.SPLITPAGESANDMEDIA`, or `siteStructure.FLAT` values; for example,

```
// specify flat site structure
website.exportSettings.siteStructure = siteStructure.FLAT;
```

When the value of the `siteStructure` property is `siteStructure.SPLITPAGESANDMEDIA`, the `pagesFolder` and `mediaFolder` properties of the website.exportSettings Object are meaningful.

$ Object (Debugger Object)

Availability

5.0, 6.0

The debugger object ($) provides properties and methods you can use to debug your JavaScript code. This object is for debugging use only; commercially-distributed extensions should not rely on it.

This object is not available until at least one extension module enables debugging services. An extension module enables debugging services by providing a <jsxmodule> element that has the valueless debugger attribute, as the following example does.

```
// main.html file
<jsxmodule name=myCoolExtension debug>
```

$ Object Properties

flavor	*String*	Sets or returns the language flavor in use by GoLive Extend Script. Possible values are Javascript for the Netscape flavor, JScript for the Microsoft flavor or ECMAScript for the ECMA-262 flavor. Javascript is the default value of this property. **NOTE:** The flavor property applies only to GoLive. It does not apply to Web browsers.
os	*String*	Operating system version. **NOTE:** In production code, use app.osVersion to get this information without using the $ object.
version	*String*	Javascript engine version.

$ Object Functions

The $ object provides one function, which causes GoLive to break into the debugger.

bp

$.bp()

Executes a breakpoint.

Availability

5.0, 6.0

Description

This method is equivalent to the JavaScript debugger statement.

5 Tags

This chapter contains reference descriptions of all markup tags the GoLive Extend Script SDK provides. GoLive interprets these tags to create the JavaScript objects that Chapter 4, "Objects" describes.

Module

When GoLive interprets the `Main.html` file that defines an Extend Script extension, it creates a Module Object, which provides access to the extension module from JavaScript. The `<jsxmodule>` tag sets this object's JavaScript name and activates special features.

jsxmodule

`<jsxmodule name="`*moduleName*`" timeout="`*seconds*`" locale="`*countryCode*`" debug>`

Sets this extension module's JavaScript name and activates optional features.

Attributes

`name`	Optional. Specifies this module's JavaScript name. In the JavaScript name space, the `module.name` expression evaluates to this value. If the `<jsxmodule>` tag is missing or if it supplies no `name` attribute, GoLive supplies a default `module.name` value which is the name of the folder that holds the extension's `main.html` file.
`debug`	Optional. When any module supplies this attribute, the JavaScript Output palette is enabled and the integrated debugger is active for all modules. The `debugger;` JavaScript statement acts as a breakpoint when the integrated debugger is active.
`timeout`	Optional. Specifies the number of seconds the JavaScript engine waits for a response from the currently-running JavaScript code. If this amount of time elapses without a response from the currently-running JavaScript code, GoLive generates a timeout runtime error, stops waiting for a response, and exits the script. By default, the GoLive engine waits indefinitely for JavaScript code to return. Your module can specify the amount of time GoLive waits for its JavaScript code to return by setting this attribute's value to a number of seconds between `0` and `9999`. (9999 seconds is 2 hours, 46 minutes 39 seconds.) Setting this attribute to `0` seconds or a boolean value of `false` causes GoLive to wait indefinitely for this module's code to return. GoLive always waits indefinitely for its calls to your Event-Handling Functions to return, regardless of whether you have specified a timeout value. The code that sets the global default timeout value runs once when GoLive initializes all the modules in the **Extend Scripts** folder at startup time.

<jsxmodule> attributes (*continued from preceding page*)

locale	Defines a temporary locale used to translate the module's strings for testing purposes. Within the scope of this module, this value overrides the locale ID that GoLive uses. Valid values are two-letter ISO-3166-1 country codes as used for top-level Internet domains, such as DE for Germany or IT for Italy. The US country code represents all versions of English. Valid codes are those representing the Roman-language subset of all codes the ISO-3166-1 standard defines.You can disable all of the module's automatic translation features by setting NONE as this attribute's value; however, this feature is intended for development and testing purposes only. Do not use this feature to disable automatic localization in commercial extensions.

Description

GoLive always creates a Module object when it loads an extension's Main.html file, even if the extension does not define a <jsxmodule> element. This tag's attributes activate debug mode, automatic localization of JavaScript strings, and a specified JavaScript execution timeout for this extension's module object only.

Scripts

Normally, scripts reside inside the <body> element, but you can place them inside an optional <head> element.

script

<script [src=*localURLOnly*] > *JavaScriptCode* </script>

Place your extension's JavaScript code between the <script> and </script> tags.

Attributes

src	Optional. Relative URL specifying the path from the main.html file that contains this <script> tag to an external JavaScript file on the local file system. This attribute does not support URLs to remote files.

Description

You can use the optional src attribute to include the contents of an external JavaScript file in the extension's main.html file. Gratuitous use of this feature is not recommended; normally, the main.html file defines all of an extension's code.

When used in a `main.html` file, a `<script>` element that provides a `src` attribute value cannot provide any other content. The following code demonstrates correct use of the `src` attribute.

```
<script src="abc.js"></script>
<script>
initializeModule() {
}
parseBox() {
}
</script>
```

When using the `src` attribute in a `main.html` file, note the following differences from the JavaScript 1.1 standard for Web browser use of this attribute:

- Included files must reside on the local file system. A URL reference to a file residing on a remote server is not a valid `src` attribute value.
- JavaScript code appearing between the `<script>` and `</script>` tags in the `main.html` file executes after code provided by the `src` attribute. JavaScript in the `main.html` file overwrites same-named entities defined by the external file.

NOTE: A GoLive 5 extension's `Main.html` file must hold exactly one set of `<script></script>` tags; in GoLive 6, the `Main.html` file can hold multiple `<script>` tags.

Menus

You can use the following tags to create menus:

- The <jsxmenu> tag defines a custom menu that appears to the left of the **Window** menu in the GoLive menu bar.
- The <jsxitem> tag defines a custom menu item that appears in a custom menu.
- One <jsxmenubar> binary tag per `Main.html` file surrounds all the other tags that add menus to the GoLive menu bar.

A menu's name appears as a property of the global JavaScript namespace and as an element of the `menus` global array.

Each menu item is accessible as a property of the Menu object within which it is defined or as part of that Menu object's `items` array.

When GoLive is about to display a menu item, it passes the item to the `menuSetup` method of the extension that provided the menu item. Your implementation of this method can enable or disable the item and set or clear its check mark.

When the user chooses a menu item, GoLive passes the item to the `menuSignal` method of the extension that provided the menu item.

jsxmenubar

```
<jsxmenubar>
     <!-- <jsxmenu> element goes here. -->
</jsxmenubar>
```

The <jsxmenubar> tag wraps all of the tags that define an extension's custom menus and custom menu items:

- The <jsxmenubar> opening tag precedes all of the tags that define an extension's custom menus and custom menu items.
- Its companion </jsxmenubar> closing tag must close all the HTML that defines custom menus and custom menu items,

The <jsxmenubar> tag wraps the GoLive menu bar; hence, an extension's source code must not contain more than one <jsxmenubar></jsxmenubar> tag set.

The <jsxmenubar> tag has no attributes.

jsxmenu

```
<jsxmenu name="name" title="Menu text">
     <!-- <jsxitem> elements go here -->
</jsxmenu>
```

The <jsxmenu> tag creates a custom menu that appears left of the **Window** menu in the GoLive menu bar. The binary <jsxmenu> tag wraps one or more <jsxitem> tags, which define individual menu items.

A <jsxmenu> element must be contained by a <jsxmenubar> element. Only one <jsxmenubar> element is allowed per module, so all of the extension module's <jsxmenu> elements must be contained by this <jsxmenubar> element.

Attributes

name	The Javascript name of the menu. This name appears in the JavaScript global namespace and in the menus global array. Set this attribute's value to "special" to append this menu to the **Special** menu instead of installing it in the top level of the GoLive menu bar. If this value duplicates the name attribute of any other <jsxmenu> element in any other module, this element's menu items are appended to the previously-defined menu.
title	The menu's title in the GoLive menu bar or in the Special menu.

jsxitem

```
<jsxitem name="objectName" title="menuItemText" dynamic>
```

The <jsxitem> tag defines a single menu item inside a menu.

A menu item is accessible both as a property of its enclosing Menu object and as an element of that Menu object's items array. For example, if the myItem menu item appears in the myMenu custom menu, the myMenu.myItem expression provides access to this menu item in JavaScript. Another way to retrieve the item would be to pass its name property as the index into the

myMenu object's items array; for example, if the <jsxitem> element definition supplies Fred as the value of its name attribute, you could use the following line of JavaScript to retrieve this menu item:

```
var menuItemHolder = myMenu.items["Fred"];
```

Attributes

name	The Javascript name of the menu item. This name appears as a property of this menu item's enclosing Menu object and as an element of that Menu object's items array.
title	The menu item's text as it appears in the menu. Set this attribute's value to a single dash ("-") creates a menu separator item.
dynamic	When this attribute is present, GoLive passes this menu item to its extension's menuSetup method before displaying it. The menuSetup method sets the item's checked and enabled states. Menu items that do not include this attribute do not generate calls to the menuSetup method.

Custom Menu Example

The following example defines two menus. The first menu appends itself to the **Special** menu, while the second menu installs itself in the GoLive menu bar to the left of the **Window** menu. The first menu item initializes itself dynamically—its checked state changes each time the user chooses it. Selecting either menu item opens an alert window that displays the menu item text.

```
<jsxmenubar>
     <jsxmenu name="special">
        <jsxitem name="one" title="Special One" dynamic>
        <jsxitem name="two" title="Special Two">
        <jsxitem name="three" title="Special Three">
     </jsxmenu>
     <jsxmenu name="more" title="More">
        <jsxitem name="more1" title="Alert One">
        <jsxitem name="separator" title="-">
        <jsxitem name="more2" title="Alert Two">
     </jsxmenu>
</jsxmenubar>
<script>
     function menuSetup (item) {
        // simply invert the check mark for "one"
        if (item.name == "one")
           item.checked = !item.checked;
     }
     function menuSignal (item) {
        if (item.name == "one")
           alert ("Selected: " + item.title);>
     }
</script>
```

Dialogs and Palettes

A dialog is a container element (<jsxdialog>, <jsxpalette>, or <jsxinspector>) that provides the dialog window. Like an HTML form, this container may hold any number of controls, each defined as <jsxcontrol> elements, as the following example shows.

```
<!-- main.html file ->
<html>
     <body>
        <jsxdialog name="myModalDialog" title="Burger Order Form"
                  width="280" height="400" >
          <jsxcontrol  type="static" name="trimmingsPrompt"
                       value="Which trimmings would you like on your burger?"
                       posx="10" posy="10" width="240" height="18">
          <jsxcontrol  type="checkbox" name="lettuceBox" value="Lettuce"
                       posx="10" posy="30" width="60" height="18">
          <!-- additional jsxcontrols as needed... -->
          <jsxcontrol  type="button" name="panicButn" value="Panic"
                       posx="150" posy="30" width="60" height="18">
        </jsxdialog>
     </body>
</html>
```

There are three tags you can use to create different kinds of dialogs. You can use any of the following in place of the <jsxdialog> tags in the preceding example:

- The <jsxdialog> tag defines a modal dialog window. This dialog requires a response before it allows the user to continue.
- The <jsxpalette> tag defines a palette window, also known as a modeless dialog window. This dialog never requires a response. JavaScript code or the user can show or hide this palette.
- The <jsxinspector> tag creates a special-purpose palette used to implement the inspector for a custom tag the <jsxelement> tag defines. The SDK activates this palette automatically when the custom element is the selected element in a GoLive document window. Because an Inspector is always associated with a <jsxelement> element, information on using the <jsxinspector> tag is in the "Objects Palette Icons and Customized Content" section of this chapter.

The dialog, palette, or inspector is the container for a collection of <jsxcontrol> elements that provide the rest of the dialog's user interface. The <jsxcontrol> tag's type attribute specifies the kind of control it creates: pushbutton, static text, checkbox, radio group, or any of numerous others. For a complete description of the kinds of controls the <jsxcontrol> tag can define, see its reference description on page 269.

jsxdialog

```
<jsxdialog name="objectName" title="Title Of Dialog" width="anInteger"
     height="anInteger" >
     <!-- <jsxcontrol> elements here -->
</jsxdialog>
```

The <jsxdialog> tag defines a modal dialog window that acts as a container element for the tags and attributes that define the dialog's content. The content of a dialog is a form that GoLive implements as a table. The <jsxdialog> tag wraps this form.

The width and height attribute values specify the size of the dialog window in pixels. You can place buttons, checkboxes, static text and other controls in the dialog by adding <jsxcontrol> element definitions to the body of the <jsxdialog> element definition.

When GoLive interprets a <jsxdialog> element, it creates a Dialog object to represent it. The name attribute of this element defines the JavaScript name of the Dialog object; thus, you can retrieve this dialog object by name from the dialogs global array as the following example does.

```
var myDlog = dialogs["objectName"];
```

This dialog object also appears as a property of the global namespace. You can retrieve this dialog object by name from the JavaScript global namespace as the following example does.

```
var myDlog = objectName;
```

The runModal method of the Dialog object causes it to execute as a modal dialog, meaning that it requires a response before allowing the user to proceed. There are two ways to create a button that closes the dialog window:

- Name the button one of the special names dialogOK, dialogCancel, or dialogOther. When the user clicks a button having one of these names, the runModal method closes the dialog and returns a numeric value corresponding to the name of the button.
- Call the endModal method from the controlSignal method that handles that button's events. Pass as the endModal method's argument a value the runModal method is to return as the reason the dialog closed.

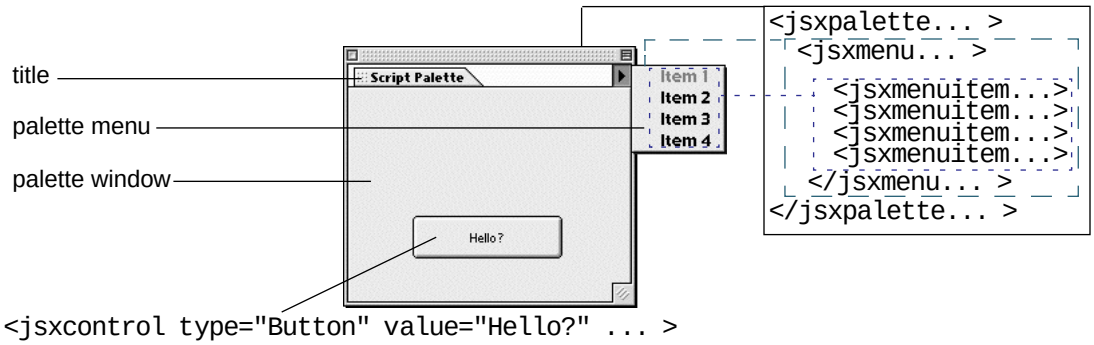
Attributes

name	The dialog's name in the global JavaScript namespace. The dialog may be accessed via this name in the global dialogs array, as well. For reliable name-based access, this value must be unique in the universe of all currently-running extensions.
title	The title of the dialog window (Windows only).
width	The width of the dialog, expressed in pixels. If you use a layout grid to design the dialog in GoLive, the SDK embeds a <table> tag that overrides this value.
height	The height of the dialog, expressed in pixels. If you use a layout grid to design the dialog in GoLive, the SDK embeds a <table> tag that overrides this value.

jsxpalette

```
<jsxpalette name="objectName" title="Title Of Palette" order="anInteger"
                  width="anInteger" height="anInteger" >
    <!-- <jsxcontrol> elements that define palette content -->
    <!-- <jsxmenu> and <jsxitem> elements that define palette menu -->
</jsxpalette>
```

FIGURE 5.1 Modeless dialog, also known as a floating window, or palette



The <jsxpalette> tag creates a modeless dialog, also known as a floating window or palette window. The palette's title is displayed in the **Window** menu so the user can open and close the window like any other palette window. The upper right corner of the palette window provides a pull-down menu.

When GoLive interprets a jsxpalette element, it creates a Palette object. You can retrieve this object by name from the JavaScript global namespace or from the global dialogs array, as in the following examples.

```
var myDlog = objectName; // palette name is global property
var myDlog = dialogs["objectName"]; // palette name is index selector
```

Although you can set the value of this object's `visible` property to show or hide the palette window, you cannot obtain similar results by specifying a `visible` attribute in your jsxpalette element's source representation. That is, you cannot force a jsxpalette window to be visible upon creation by specifying a `visible=true` attribute—the attribute and its value are included in the element's source representation but they have no effect on the palette's visibility.

Attributes

`name` The palette's name in the global JavaScript namespace. The palette may be accessed via this name in the global `dialogs` array, as well. For reliable name-based access, this value must be unique in the universe of all currently-running extensions.

`title` The title of the palette window. This text also appears in the **Window** menu.

<jsxpalette> attributes (*continued from preceding page*)

order	The sort order of this palette's name in the **Window** menu. Higher values place this item closer to the bottom of the menu. Values greater than 9999 cause undefined results. For additional details, see "Window Menu Items" on page 317 in Appendix A, "Sort Order Tables."
width	The palette's width, expressed in pixels.
height	The palette's height, expressed in pixels.

Controls

The <jsxcontrol> tag creates the controls in a dialog, palette, or Inspector.

jsxcontrol

```
<jsxcontrol type="KindOfControl" name="JavaScriptName" value="InitialValue"
            group="groupID" posx="NumOfPixels" posy="NumOfPixels"
            width="NumOfPixels" height="NumOfPixels" halign="SeeReference"
            valign="SeeReference" >
```

The <jsxcontrol> tag creates a variety of controls that appear as the content of modal or modeless dialogs. Its attribute values define the control's type, its initial position, and its alignment behavior.

NOTE: Controls dropped on a layout grid are implemented as table elements. Such controls ignore the width and height attributes their enclosing dialog specifies; instead, they use the table's width and height attributes as the size of their enclosing container.

Attributes

name	The JavaScript name of the control. The control appears under this name as a property of the dialog or palette that contains it; for example, the JavaScript expression myDialog.myControl returns the control named myControl that is contained by the dialog container named myDialog. Alternatively, the control's name value may be used to retrieve the control from the global controls array. Each control's name must be unique among all controls that share the same dialog container. Controls that do not share the same dialog container can use non-unique names. For example, a <jsxcontrol> element having the name myButton can be used in multiple dialog containers without causing namespace clashes. However, two controls having the name myButton cannot appear in the same dialog or palette. **NOTE:** When a control's name is not unique among all currently-running extensions, the control cannot be retrieved by name from the global controls array reliably.

<jsxcontrol> attributes (*continued from preceding page*)

type	The type of control this element defines. Valid values are:	
	button	Pushbutton
	custom	Custom control
	checkbox	Checkbox
	check	Checkbox
	radiobutton	Radio button
	radio	Radio button
	edit	Edit field that signals a change when the input focus changes and the edit field's contents have changed; that is, changing focus without changing content does not signal a change.
	buttonedit	Edit field that signals a change when the input focus changes and the contents of the field have changed. Optionally, this control can display an Enter button. To enable the enter button, uncheck the **Direct input for textfields** checkbox in the **Edit>Preferences...>General>User Interface** panel.
	editarea	Multiline edit field
	static	Static text field
	color	Color select field
	frame	Horizontal line, vertical line, or frame box. The value string is displayed for boxes and horizontal lines. A horizontal line is created when the height attribute is omitted; similarly, a vertical line is created when the width attribute is omitted. You cannot click on a frame control, and such controls do not call the controlSignal function. In the extension's main.html file, the <jsxcontrol type= frame...> element must precede the controls it frames; if not, the frame draws on top of the framed controls, instead of behind them.
	password	Password-entry field.
	popup	Popup menu
	list	List box
	tree	Hierarchical view of TreeNode Objects. See "Tree Control" on page 180 of the *Extend Script SDK Programmer's Guide*.
	line	Pane separator. GoLive 6 only.
	urlgetter	URL entry field

NOTE: You cannot initialize a urlgetter control by assigning a URL to its value property. Doing so results in a disabled control that displays the assigned value. To change the URL this control displays, call the setLink method.

`<jsxcontrol>` attributes (*continued from preceding page*)

`value` The control's initial value. For popup and list box controls this value is a comma-separated list of items to display; in such lists, a dash creates a menu item separator. For example, `<jsxcontrol type=popup value= "Red, - , Blue" ... >` creates a popup list in which a separator appears between the **Red** and **Blue** list items.

`group` The group ID for radio buttons. All buttons with the same ID are automatically turned off when one button is selected. If the attribute is omitted, the button is not affected by other buttons. The group ID is an arbitrary string value. It is valid for all controls within the same extension; it would affect controls within other dialogs and the same group ID, so choose these IDs carefully.

`posx` The x-coordinate of the upper-left corner of the control in the dialog window. When the user drops the control on a table grid in layout mode, GoLive updates these attributes to reflect the position of the control on the page. This makes it easy to create f.ex. an inspector by dropping the controls on a table grid and moving them into position.

`posy` The y-coordinate of the upper-left corner of the control in the dialog window. When the user drops the control on a table grid in layout mode, GoLive updates these attributes to reflect the position of the control on the page. This makes it easy to create f.ex. an inspector by dropping the controls on a table grid and moving them into position.

`width` The control's width, expressed as a number of pixels.

`height` The control's height, expressed as a number of pixels.

`halign` The horizontal alignment of the control when the size of the window changes. Valid values are:

`left`	Left-aligned
`center`	Centered
`right`	Right-aligned
`scale`	Scaled automatically

`valign` The vertical alignment of the control when the size of the window changes. Valid values are:

`top`	Top-aligned
`center`	Centered
`bottom`	Bottom-aligned
`scale`	Scaled automatically

Objects Palette Icons and Customized Content

The **Objects** palette holds icons the user can drag to Layout view to add predefined content to the page. This predefined content is known as a **palette entry**.

You can use the following tags to install a custom palette entry in the **Objects** palette:

- The `<jsxpaletteentry>` tag defines the palette entry itself:
  - It provides the content this palette entry adds to the page. This content can consist of any combination of standard HTML tags, SDK-defined tags, custom tags defined by the <jsxelement> tag, and associated attribute values.
  - It supplies the drag-and-drop icon that represents this content in the Objects palette.

    You'll use the `<img>` tag to supply palette icons and the placeholder graphic that represents a custom element in Layout view.The SDK-provided version of this tag provides a `name` attribute that the standard HTML version of the tag does not provide. This additional attribute enables your extension's JavaScript code to retrieve and manipulate `<img>` elements by name.
- The `<jsxelement>` tag defines the name of a custom markup tag to associate with the palette entry's content; for example, you can use this tag to define your own `<myTag>` tag. An associated `<jsxpaletteentry>` element provides content that the `<myTag>` tag adds to the page.

  NOTE: The name of a custom tag must consist of lowercase characters only.
- The `<jsxinspector>` tag creates the inspector window used to interact with the content the palette entry adds to the page.
- The `<jsxpalettegroup>` tag specifies the Objects palette tab that is to hold the palette entry's drag-and-drop icon. This tag can create a new, named tab in the Objects palette or it can specify the use of any existing tab except the **Custom** tab. Optionally, this tag can supply a custom icon to display on the tab itself.

GoLive uses the `classid` attributes of the `<jsxelement>`, `<jsxpaletteentry>`, `<jsxinspector>`, and `<jsxpalettegroup>` tags to associate the palette entry with a custom tag and an inspector window. All of the tags that define a palette entry must specify the same `classid` attribute value.

jsxpalettegroup

```
<jsxpalettegroup name="objectName" display="tabName" tabOrder="anInteger"
                 order=anInteger picture="tabIcon" >
                 <!-- <jsxpaletteentry> elements -->
</jsxpalettegroup>
```

The `<jsxpalettegroup>` tag specifies the Objects palette tab that holds a group of palette entries. This tag can define a new tab in the Objects palette or select any existing tab except the

Custom tab. Optional attributes specify the tab's label text, icon, and location with respect to other tabs in the Objects palette.

Attributes

`name` The JavaScript name of this palette group. This case-sensitive value is either your unique identifier or one of five predefined values. Specifying a unique value creates a new tab in the Objects palette; specifying a predefined name installs this palette group in one of the built-in Objects palette tabs. The predefined names are:

`Basic`	Basic elements (images, plug-ins, scripts, and so on.)
`Forms`	Form elements
`Project`	Site elements
`CSObjects`	Actions.
`WO`	WebObjects

`display` The text that identifies this palette group and its tab to the user. This text appears in the palette menu of the Objects palette. This text also appears in the lower-left corner of the Objects palette when the mouse cursor pauses over this palette group's tab. The SDK ignores this value when the name attribute specifies a predefined tab name.

`taborder` A number that describes the position of this tab amongst all tabs appearing in the Objects palette. Higher values place the tab further to the right. The SDK ignores this value when the name attribute specifies a predefined tab name. For a listing of the taborder values of built-in tabs, see "Objects Palette Entries" on page 318.

`order` A number that describes the position of this palette group's entries amongst all entries appearing in the **Objects** palette's built-in tabs. Higher values place the tab further to the right. The highest valid value is 32767.

This attribute is used only to add palette entries to one of the built-in tabs, such as the **Basic** tab. When you add palette entries to your own tab, the SDK ignores this attribute because it always adds entries to a custom palette in the order the extension's source code defines them; thus, you can rearrange the icons in your own palette by simply reordering their definitions in your extension's source code. You can use this approach to add your palette group to a built in tab too: if the value of your palette entry's order attribute is less than that used by a built-in palette entry, GoLive adds your icon to the palette before it adds the built-in icon. For a listing of the order values of built-in tabs, see "Objects Palette Entries" on page 318.

`picture` The JavaScript name of the `<IMG>` element that provides the tab's icon. GoLive scales this image to 12 pixels high by 12 pixels wide automatically when it installs the palette group's icon. The SDK ignores this value when the name attribute specifies a predefined tab name.

jsxpaletteentry

```
<jsxpaletteentry display="DescriptionOfTag" classid="uniqueIdemtifier"
                 picture="paletteIcon">
     <!-- HTML content of custom tag goes here -->
</jsxpaletteentry>
```

The <jsxpaletteentry> tag defines a palette entry within a palette group. Attributes of the <jsxpaletteentry> tag specify

- The icon and descriptive text that represent the palette entry in the GoLive user interface.
- An identifier that associates the palette entry with a <jsxelement> custom markup tag and a <jsxinspector> inspector window.

The body of the <jsxpaletteentry> element defines the HTML content this palette entry adds to the page when the user drags the palette entry's icon from the Objects palette to Layout view. Your <jsxpaletteentry> element replaces the <*HTMLContentOfCustomTagHere*> placeholder with the exact HTML content your palette entry is to insert into the GoLive document. This content can consist of any combination of HTML, SDK-defined tags, and custom tags defined by the <jsxelement> tag.

The <jsxelement> tag defines only the name of a custom tag; the default attributes the custom tag provides, as well as their values, are defined by the <jsxpaletteentry> element having the same classid attribute value. If the <jsxelement> tag that defines the custom tag name specifies width and height attributes, Go Live uses them to set the size of the box object it creates to represent a palette entry dropped into Layout view.

IMPORTANT: *The name of a custom tag must consist of lowercase characters only.*

Attributes

display	The name that appears in the lower-left corner of the Objects palette when the mouse pointer pauses over this palette entry.
classid	The class name of the markup element. This value must match the classid attribute of the <jsxelement> this <jsxpaletteentry> installs in the Objects palette; its <jsxinspector> window also has this classid value. An individual classid value must be unique among all currently-running extensions, and must be used by one extension only.
picture	The the icon which is displayed in the palette. GoLive scales this picture to 24 pixels high by 24 pixels wide automatically when it installs the palette entry's icon.

img

<img name= *JavaScriptName* macsrc=*urlOrPathname* winsrc=*urlOrPathname*>

The Extend Script SDK provides its own version of the familiar <img> tag. In addition to providing a name attribute, this tag provides attributes that support platform-specific paths to the image source file.

Attributes

name	The JavaScript name of the markup element GoLive generates when it parses this tag and its attributes.
src	The URL of the source image file. To provide platform-specific pathnames, specify winsrc and macsrc attributes instead of a src attribute.
macsrc	The URL of the image source file on Mac OS platforms.
winsrc	The URL of the image source file on Windows platforms.

jsxelement

```
<jsxelement tagName="nameOfCustomTag" classid="jsxpaletteentryID"
            type="typeOfTag" tagStart="delimiterChars" glue="mkupLang"
            leftMargin="numPixels" rightMargin="numPixels"
            topMargin="numPixels" bottomMargin="numPixels"
            invisible fixedWidth=numPixels fixedHeight=numPixels >
</jsxelement>
```

This tag defines the name of a custom markup tag.

IMPORTANT: *The name of a custom tag must consist of lowercase characters only.*

This tag's attributes specify characteristics of the Box object that provides this element's visual representation in **Layout** view.

Do not confuse the <jsxelement> tag's attributes with those of the custom tag it defines. The <jsxelement> tag defines ONLY the custom tag's name. The custom tag's attributes and any

other content it adds to the page is defined by the <jsxpaletteentry> element that has the same classid attribute value as the <jsxelement> tag that defines the custom tag's name

Attributes

tagName	The name of the tag that this <jsxelement> element defines. Valid values are composed of lowercase characters only.
classid	Unique identifier associated with the tagname tag. This value must match the classid attribute of the <jsxpaletteentry> and <jsxinspector> tags associated with the custom tag this <jsxelement> tag defines. An individual classid value must be unique among all currently-running extensions, and must be used by one extension only.
type	The tag type, which is one of the following:

binary	A binary tag that can contain any HTML code. The code is not displayed and your extension must draw and maintain this box's visual representation in Layout view.
container	A binary tag that can contain any HTML code. The code is displayed inside the box associated with this element. This box should provide margins adequate to display this code. The drawBox function must draw within these margins. Drawing that occurs outside these margins is not displayed on the screen. By default, container boxes cannot be resized.
plain	A standard tag that draws a placeholder graphic in Layout view.
ssi	A server-side include; a comment that begins with a pound sign (#).
bracket	A tag delimited by straight brackets ([or]); for example, [hello] is such a tag.
percent	A tag delimited by double percent (%%) signs.

<jsxelement> attributes (*continued from preceding page*)

tagStart	The character or characters immediately following the left angle bracket (<) of a custom tag not delimited solely by angle brackets. Use this attribute to create custom tag delimiters and non-HTML tags. Valid values are: "" — Standard HTML tag. When the value of the tagStart attribute is the empty string, no characters following the left angle bracket (<) are part of the tag delimiter; in other words, tagStart="" indicates that this <jsxelement> element defines a standard HTML tag. !-- — Comment open tag. When !-- is the value of the tagStart attribute, it indicates that this tag begins with a left angle bracket (<) followed by the !-- characters; that is, this element defines the <!-- tag that begins a standard HTML comment. ! — SGML (Standard Generalized Markup Language) tag. When the value of the tagStart attribute is an exclamation point (!), this element defines the <!*chars*> SGML tag. The *tagName* characters are defined by the value of this element's tagName attribute. bracket — Tag delimited by square brackets.When the value of the tagStart attribute is the bracket string, this element defines the [*tagName*] tag. The *tagName* characters are defined by the value of this element's tagName attribute.
glue	String specifying the markup language environment in which this custom tag overloads the standard one; for example, you could use this attribute to define a custom img tag that GoLive uses only for source written in a particular markup language, such as WML or XML. When this attribute is not present, defining your own version of an existing tag replaces that tag throughout the GoLive environment.
leftMargin	The left margin of the generated container box.
rightMargin	The right margin of the generated container box.
topMargin	The top margin of the generated container box.
bottomMargin	The bottom margin of the generated container box.

<jsxelement> attributes (*continued from preceding page*)

`invisible`	Presence of this valueless attribute makes the element non-visible in a browser window; for example, when you view an HTML file in a Web browser, it does not make comments visible. By default, Layout view displays placeholder graphics that represent invisible elements. When such elements are visible in **Layout** view, the **Edit>Hide Invisible Items** menu item makes them non-visible in **Layout** view. Conversely, when such elements are not visible in **Layout** view, the **Edit>Show Invisible Items** menu item makes them visible in **Layout** view.
`fixedwidth`	If present, the box cannot be resized horizontally. This attribute can be specified with or without a value; if a value is supplied, it denotes the width of the box. By default, container boxes cannot be resized.
`fixedheight`	If present, the box cannot be resized vertically. This attribute can be specified with or without a value; if a value is supplied, it denotes the height of the box.By default, container boxes cannot be resized.

jsxinspector

```
<jsxinspector name="objectName" title="nameInWindowMenu"
              classid="yourUniqueID" width="anInteger" height="anInteger"
              minwidth="numPixels" minheight="numPixels" >
    <!-- <jsxcontrol> elements go here -->
</jsxinspector>
```

This tag defines the inspector window for the box object that GoLive creates when the user drops a palette entry into Layout view. Because an inspector is a special kind of dialog, its attributes, structure, and behavior are identical to those of the <jsxdialog> tag. All inspectors use the same window, so you can ignore the <table> tag that usually defines their window size. (The Inspector lays out its controls on a grid—controls dropped on a layout grid are implemented as table elements.)

When a custom box is selected, GoLive activates the inspector window having the same `classid` value as the selected box. Before displaying the window, GoLive calls its extension's `inspectBox` method, passing the selected box as its argument. Your implementation of this method initializes the elements of the inspector window to display the current data.

When the user manipulates controls in the inspector, GoLive calls the `controlSignal` method just as it would for any other dialog, passing the control that changed as the argument to this method. Your implementation of this method alters the corresponding elements of the box as well as the HTML representation of its code to reflect a new value returned by the control that changed.

The value of the <jsxinspector> tag's `name` attribute appears as a property of the JavaScript global namespace and as an element of the global `dialogs` array.

In GoLive 6, a Box object can have multiple inspectors. To create more than one inspector, define additional inspectors with alternate *classid* attributes, as the following example does:

```
<jsxinspector name="insp1" title="one" classid="myclass">
…
</jsxinspector>
<jsxinspector name="insp2" title="two" classid="alternate">
…
</jsxinspector>
```

At runtime, you can specify the inspector a box is to use by setting its classid property; for example, the following line of code specifies that the myBox object is to use the inspector created from the `<jsxinspector classid= "alternate" … >` element:

```
var myBox = boxes[0]; box.classid = "alternate";
```

In GoLive 6.0, the `minheight` and `minwidth` attributes specify the inspector window's minimum width and height in pixels.

Attributes

`name`	The Javascript name of the inspector window. This name appears in the `dialogs` global array.
`title`	The title of the inspector dialog
`classid`	An identifier that associates a custom element with its inspector. This value must match the `classid` attributes of the `<jsxpaletteentry>` and `<jsxelement>` tags that define the custom element to inspect. An individual `classid` value must be unique among all currently-running extensions, and must be used by one extension only.
`minwidth`	Minimum width of the palette, expressed as a number of pixels. GoLive 6 only.
`minheight`	Minimum height of the palette, expressed as a number of pixels. GoLive 6 only.

Locale

The `<jsxlocale>` tag supplies an HTML table that maps English-language versions of an extension module's strings to their localized counterparts.The SDK uses this table to replace English-language strings with localized versions when GoLive loads the extension module. The SDK replaces attribute value strings with localized versions automatically; thus, menu items and such are localized automatically. To enable the dynamic localization of strings that are arguments to JavaScript methods, replace them with calls to `module.localize(`*message*`)` as in the following example.

```
module.localize("EnglishStringToLocalize");
```

For a more detailed explanation, see "Dynamic Localization" on page 182 of the *GoLive 6.0 Extend Script SDK Programmer's Guide*.

jsxlocale

`<jsxlocale src="`*pathToFolder*`/locale.html"> `*translationTable*` </jsxlocale>`

The `<jsxlocale>` tag is a container for a table that defines localized versions of the strings an extension displays in the GoLive user interface: menu items, dialogs, and so on.

Attributes

`src`	Absolute or relative path to the `locale.html` file.

Description

The translation table itself is a standard HTML table that presents localization data in a specific format. A typical translation table looks like the one Table 5.1 depicts.

TABLE 5.1 Translation table example

US	DE	NO
`Are you sure?`	`Sind Sie sich sicher?`	`Er du sikker?`
`Search`	`Suchen`	`Start søk`

The translation table format is as follows:

- The topmost item in each column is a two-letter country code identifying the Roman language in which the rest of the column's entries are written. Valid codes are those representing the Roman-language subset of all codes the ISO-3166-1 standard defines.
- The leftmost item in each row is an English-language string to be localized. This string must appear in the table exactly as it appears in the extension's source code. The remaining cells in each row hold localized versions of the string that appears in the leftmost cell in the row.

Optionally, you can use the `src` attribute to load a localization table from an external HTML file named `locale.html` that you provide. The body of this HTML file holds the same `<jsxlocale>` element (Table 5.1) that would normally appear in the extension's `main.html` file. The encoding of this file (specified by the `<meta>` element) must match that of the `main.html` file. If the `locale.html` file's encoding does not match the `main.html` file's encoding, Dynamic Localization uses the `main.html` file's encoding.

You can use the `src` attribute to associate multiple translation tables with an extension, usually for purposes of providing additional translations. For example, the `<jsxlocale>` element in the `main.html` file could reference a `locale.html` file that translates English strings to European languages, while the <jsxlocale> element in the `locale.html` file might reference

another translation table that provides translations for other target languages and markets. You can use a scheme like this to localize an extension for different target markets without modifying the main.html file—all you need to do is supply an appropriate external localization table.

Do not embed a translation table in a `<jsxlocale>` element that provides the `src` attribute; instead, place the translation table in the `<body>` element of the `locale.html` file. When GoLive finds multiple definitions of the translation table, it uses the first definition it loads. If the `main.html` file provides a translation table, that one is used rather than the one the `src` attribute specifies.

If GoLive cannot find the `locale.html` file at the specified location, no additional translation beyond the default behavior of GoLive occurs.

Example

The following example code shows Table 5.1 installed as a module's translation table.

```
<jsxlocale>
   <table border="2" cellpadding="2">
      <tr>
         <th>US</th>
         <th>DE</th>
         <th>NO</th>
      </tr>
      <tr>
         <td>Are you sure?</td>
         <td>Sind Sie sich sicher?</td>
         <td>Er du sikker?</td>
      </tr>
      <tr>
         <td>Search</td>
         <td>Suchen</td>
         <td>Start s&oslash;k</td>
      </tr>
   </table>
</jsxlocale>
```

You may need to use special character sequences to express certain characters as HTML. For example, note that the `søk` string in the **NO** column of Table 5.1 becomes `s&oslash;k` in the table's HTML source. When you create your translation table in Layout view, GoLive expresses such special characters as correctly-formed HTML automatically.

For more information, see

- "Dynamic Localization" on page 182 of the *GoLive 6.0 Extend Script SDK Programmer's Guide*.
- "Translator Object" on page 222 of this Reference.

Document Translation

The <jsxtranslator> tag enables an Extend Script extension to parse the text of a source file before the GoLive parser even gets the opportunity to parse it. Because the DOM has not yet been generated, a translator uses regular expressions to identify a specific piece of code to replace with different code, either temporarily or permanently.

The goal of translation is to translate documents of foreign markup into a format GoLive accepts: HTML or XML. Once the document is translated, the user can manipulate the content of the original document in Layout View just as if it were an HTML or XML document.

A few helper functions are available to replace or to rewrite the text that the translator returns. These function add GoLive-specific tags to the code so GoLive is able to recognize the code and display the translated version correctly. For example, you can use a translator to mimic the output of server-side code in Layout view. When the document is saved, these tags also enable GoLive to write the modified document back to its file in its native format.

For more information, see "Document Source Translation" on page 204 of the *GoLive 6.0 Extend Script SDK Programmer's Guide*.

jsxtranslator

```
<jsxtranslator classid=uniqueIdemtifier direction=oneWayOrTwoWay
               haslinks=yesOrNo blend=yesOrNo>
    <!--- <extension> and <expression> elements go here -->
</jsxtranslator>
```

The `jsxtranslator` tag provides a container element that enables the preprocessing (translation) of source code as specified by `<extension>` or `<expression>` subelements.

Availability

6.0

Attributes

`classid`	An identifier that associates this translator element with its inspector.This inspector allows you to modify your source code just like any other inspector. This `classid` value must be unique among all currently-running extensions, and must be used by one extension only.	
`direction`	Specifies whether translation is permanent or temporary. Valid values are:	
	`twoway`	Display translated page content layout view without changing source code permanently. Use this option to simulate serverside replacement, and so on.
	`oneway`	Translate the page content permanently while still preserving the untranslated snippets in special elements. Translators of this style can mimic code generators.

<jsxtranslator> attributes (*continued from preceding page*)

haslinks	Optional. Specifies whether to use this translator when reparsing.	
	yes	GoLive uses this translator when reparsing files. Set this value to provide a translator that translates links, such as server-side includes.
	other values no value	Do not use this translator when reparsing files.
blend	Optional. Governs the use of color blending by this translator.	
	yes	GoLive default behavior, same as omitting this attribute.
	no	Use document's original color values.

Example

The following sample code defines a translator element that acts on all .html or .htm files that contain the "<time" Unicode source string.

```
<jsxtranslator classid="timeClass" direction="twoway">
   <extension value="html">
   <extension value="htm">
   <expression value="<time">
</jsxtranslator>
```

Here's another example. This translator provides the haslinks=yes attribute, indicating that GoLive is to use this translator to process links when it reparses the source document. The following sample code defines a translator element that acts on all .html or .htm files that contain the "#include" Unicode source string.

```
<!-- translates SSI includes in .html and .htm files -->
<jsxtranslator classid="SSIinclude" direction="twoway" haslinks="yes">
   <param name="extension" value="html">
   <param name="extension" value="htm">
   <param name="expression" value="#include">
 </jsxtranslator>
```

To improve reparsing performance, you might choose not to translate links; to do so, omit the haslinks attribute entirely, or set it to any value other than the yes value.

extension

`<extension value=` *filenameExtension*`>`

The `extension` tag specifies a file type to translate; to translate multiple file types, supply multiple `extension` subelements as necessary.

Availability

6.0

Attribute

`value`	Filename extension of the kind of file to translate; for example, `htm` and `html` are valid values.

Description

Like the `<expression>` element, the `<extension>` element is the content of a `<jsxtranslator>` element.

Example

The following sample code defines a translator element that acts on all `.html` or `.htm` files that contain the `"<time"` Unicode source string.

```
<jsxtranslator classid="timeClass" direction="twoway">
   <extension value="html">
   <extension value="htm">
   <expression value="<time">
</jsxtranslator>
```

expression

`<expression value=`*regExp*`>`

The `expression` tag defines a regular expression that selects text to be translated; to specify multiple expressions, supply multiple `expression` elements as necessary.

Availability

6.0

Attribute

`value`	Regular expression that selects text to be translated.

Description

Like the `<extension>` element, the `<expression>` element is the content of a `<jsxtranslator>` element.

Example

The following sample code defines a translator element that acts on all `.html` or `.htm` files that contain the `"<time"` Unicode source string.

```
<jsxtranslator classid="timeClass" direction="twoway">
   <extension value="html">
   <extension value="htm">
   <expression value="<time">
</jsxtranslator>
```

6 Event-Handling Functions

This chapter describes functions GoLive calls to announce events, such as a change in the state of a control or document. To respond to an event, your extension implements the corresponding JavaScript global function this chapter describes. You need not implement event-handling functions which correspond to features your extension does not use; for example, if your extension does not provide a custom control, it need not implement a `drawControl` method.

GoLive 6.0 calls all functions this chapter describes. GoLive 5 calls all of these except the `docSignal` function, which is available only in GoLive 6.0.

Additionally, GoLive 6.0 enables each extension to register custom callback functions; for more information, see "Using Callback Functions" on page 205 of the *Extend Script SDK Programmer's Guide*.

Initialization and Termination

GoLive calls these functions when your extension is loaded or unloaded.

initializeModule

```
initializeModule()
```

Your implementation of this method performs extension-specific startup tasks.

Availability

5.0, 6.0

Description

Called after the module has been loaded.Your implementation of this method performs extension-specific startup tasks, such as initializing your extension's global variables, connecting to a database or DAV server, and so on.

terminateModule

`terminateModule()`

Called before the module is being unloaded.

Availability

5.0, 6.0

Description

Your implementation of this method performs extension-specific shutdown tasks, such as releasing saved references to JavaScript objects; disconnecting from a database; or, in the case of extensions that use external C++ libraries, deallocating memory reserved by external library functions.

Inter-Module Messaging

The broadcasting mechanism enables an extension to send a message to another extension requesting that it perform a task; optionally, the called extension can return a result string.

When one extension module needs to communicate with another extension module, it calls the `broadcast` method of the app Object. The `broadcast` method passes its argument to the `broadcastSignal` method of all modules or to specified modules only.

An extension that responds to messages from other extensions implements the `broadcastSignal` method. Your implementation of the `broadcastSignal` function tests the argument the SDK passes to it, performs any tasks necessary to respond to the broadcast message, and, optionally, returns a result string.

Use of the broadcast mechanism is entirely optional. If your extension does not need to communicate with other extensions, it need not implement the optional `broadcastSignal` function.

broadcastSignal

Called by the `broadcast` function to send a message to this module.

`broadcastSignal (`*argString*`);`

Availability

5.0, 6.0

Arguments

arg	*String*	Data passed by the calling module.

Description

Your extension's implementation of this optional method can return a string value or `undefined` to the caller.

Boxes

This section lists event-handling methods GoLive calls in response to user interaction with a box object in Layout view. These methods initialize the box, redraw the box, or resize the box.

parseBox

`parseBox (`*box*`, `*reason*`)`

Your implementation of this method initializes the custom *box* for the specified *reason*.

Availability

5.0, 6.0

Parameters

box	*Box*	Box object to initialize.
reason	*Number*	Reason GoLive called this method:
		2 Custom element dropped from Objects palette or pasted from clipboard.
		3 Box was repositioned.

Description

GoLive calls this method only when Layout view is active. GoLive does not call this method when Layout view is not active, nor does it call this method when it parses an HTML tag listed in Appendix B, “Supported Layout Tags.” In GoLive 6, you may avoid the use of this method by adding your custom element to the page as a subelement of a layout grid object.

GoLive calls the `parseBox` function when it must read or reread the tag associated with a box object in Layout view. When GoLive 5 parses a custom tag, it creates a box object, inserts it into the Layout view of the associated document window, and calls this function, passing the box as its argument. Your implementation of this function must initialize the box GoLive passes to it. For example, it is common for this function to set the box’s height and width as specified by the `height` and `width` attributes of the custom HTML element the passed box represents.

If Layout view is active, GoLive calls this method when:

- The document that contains the tag is opened in Layout view.
- The user switches to the document window’s Layout view from some other view.
- The user or JavaScript code changes a non-SDK element of the document while Layout view is active. A non-SDK element is one that uses any markup tag not defined by the SDK.
- The document’s `reparse` method is called when Layout view is active.

  Your `parseBox` function should avoid operations that cause GoLive to reparse the document.

NOTE: Do not call *markupObj*`.setInnerHTML()`, *markupObj*`.setOuterHTML()`, `document.reparse()` or `document.reformat()` from within the `parseBox` function. Attempting to call these methods results in a runtime error.

When GoLive calls the `parseBox` method, it passes a `reason` value which indicates the circumstances under which this method was called.

Example

```
function parseBox(box, reason) {
   box.width = (box.element.width == undefined) ? 48 :
               box.element.width;
   box.height = (box.element.height == undefined) ? 48 :
               box.element.height;
   box.url = (box.element.src == undefined) ? "none" :
             box.element.src;
   box.link = box.createLink(box.url);
   box.oldWidth = box.width;
   box.oldHeight = box.height;
}
```

drawBox

`drawBox (box, draw)`

Availability

5.0, 6.0

Returns

No return value.

Called when a custom box is about to be drawn. Your implementation of this method draws your custom element's graphical representation in Layout view. Use the `draw` object to draw lines, rectangles, circles, or text. Your `drawBox` method can call the `draw` method of a Picture object, but it cannot create objects. If you require custom picture objects, you must create them elsewhere in your extension's code.

IMPORTANT: *Your* `drawBox` *function can call drawing functions ONLY. Attempting to perform memory-intensive tasks from within the* `drawBox` *function may cause GoLive to terminate abnormally. Do not create objects, reparse documents, or download files from within the body of the* `drawBox` *method. Do not set breakpoints within the body of the* `drawBox` *method.*

boxResized

```
boxResized (box, width, height)
```

Called when the size of the box has changed. Implement it to record the changes in the markup code and/or to recalculate the contents of the box.

IMPORTANT: *Your* `boxResized` *function must not call any functions other than drawing functions. Attempting to reparse the document or to download a file from within the* `boxResized` *method may cause GoLive to terminate abnormally.*

Availability

5.0, 6.0

Returns

No return value.

inspectBox

```
inspectBox (box)
```

Availability

5.0, 6.0

Returns

No return value.

Called when the inspector dialog for a custom box is about to be displayed. Use it to fill in the control of the inspector with the current values for the box.When an inspector is active for a box, the property Inspector.box of the inspector points to the box being inspected, so you can use control.parent.box within the controlSignal() function to access the box being inspected. On the other hand, when a box is being inspected, the property `Box.inspector` is set to the inspector dialog so you can use the expression `box.inspector.controlName` to access the inspector's controls.

IMPORTANT: *Your* `inspectBox` *function must not call any functions other than those necessary to update the inspector window. Attempting to reparse the document or to download a file from within the* `inspectBox` *method may cause GoLive to terminate abnormally.*

Controls

GoLive calls these functions to provide your extension an opportunity to respond when the state of a control changes.

controlSignal

`controlSignal (control)`

GoLive calls this function when the state of one of this extension's controls changes.

Availability

5.0, 6.0

Returns

No return value.

Description

Your implementation of this method extracts the state of the control passed as its argument and responds to the state change in an appropriate manner; for example, the `controlSignal` method for a URL Getter control would set the link used by an associated page element.

IMPORTANT: *This method must not perform operations that cause GoLive to reparse.*

Do not call the `reparse` or `reformat` methods from within the `controlSignal` function. When the `controlSignal` function executes as the result of changes in the state of a control displayed by an inspector window, do not call the `setInnerHTML` or `setOuterHTML` functions from within the `controlSignal` function if doing so would cause GoLive to reparse the document.

The Layout object in GoLive 6 can modify the HTML elements in Appendix B, "Supported Layout Tags," without reparsing. GoLive 5 must reparse the document after modifying any standard HTML element, including those the aforementioned appendix lists.

GoLive 5 reparses the document when either of the `setInnerHTML` or `setOuterHTML` functions modifies or replaces a tag not defined by the GoLive 5 SDK.Tags defined by the SDK include the `<jsx...>` tags described in this book and any custom tags you create using the `<jsxelement>` tag. Note that modifying or replacing non-SDK tags provided by the `<jsxpaletteentry>` associated with a custom tag causes GoLive to reparse the document.

Custom Controls

A custom control (`<jsxcontrol type="custom" ... >`) use the `drawControl` function to update its appearance separately from updating the state of the custom control object in its `controlSignal` method.

drawControl

```
drawControl (control, draw)
```

Global callback function called by GoLive when a custom control should be redrawn. The affected control is supplied along with a Draw object which is used for all drawing. Pay respect to the mouse state (is the control clicked?) when drawing. Use the supplied Draw object for the drawing operations.

IMPORTANT: *Your* `drawControl` *function must not call any functions other than drawing functions. Attempting to reparse the document or to download a file from within the* `drawControl` *method may cause GoLive to terminate abnormally.*

Availability

5.0, 6.0

mouseControl

```
mouseControl (control, x, y, mode)
```

Global callback function called by GoLive when mouse is over the `control` and the button has been pressed (`mode==0`), the mouse has been moved with the left button pressed (`mode==1`), or the mouse button has been released (`mode==2`).

x and y are the position of the mouse pointer relative to the upper left corner of the control. mode is one of the following values:

0 - the left button has been pressed

1 - the mouse has been moved with the left button pressed

2 – the mouse button has been released

When this function is not present, the SDK calls the `controlSignal` method instead when the mouse button is released over a custom control.

Availability

5.0, 6.0

Links

The `linkChanged` method provides your extension with an opportunity to update its values according to the new value specified by the user's choice in a control of type urlGetter.

linkChanged

`linkChanged (`*linkObj*`)`

The *linkObj* link changed; your implementation of this method updates your extension's JavaScript references to the *linkObj* link.

Availability

5.0, 6.0

Parameter

linkObj	*Link*	Undo object encapsulating data required to do, undo, and redo the task this method performs.

Description

GoLive calls this method when a link changes, passing the link object that changed as its argument. Your implementation of this method updates your extension's JavaScript references to this link.

A variety of circumstances may cause a link to change and cause GoLive to call this method; for example, the user may interact with a point-and-shoot link in Site View, or with a `URLgetter` control in an **Inspector** window.

Menus and Menu Items

The SDK calls your extension's optional `menuSetup` function before it displays a menu or menu item your extension provides. The SDK calls your extension's `menuSignal` function when the user chooses a menu item your extension provides.

menuSetup

`menuSetup (menuitem)`

GoLive calls this function before displaying a dynamically-initialized menu item (a `<jsxitem>` element that provides the valueless `dynamic` attribute.)

Availability

5.0, 6.0

Parameter

`menuitem`	*menuitem*	Menu item the user chose, causing the SDK to call this function.

Description

Before displaying dynamic menu items, the SDK calls this function, passing the menu item to display as its argument. Your implementation of the `menuSetup` method sets the menu item's check mark or disables the menu item.To do so, it sets the `enabled` and `checked` states of the menu item GoLive passed to it. GoLive then displays the menu and its items as specified by the current values of these properties.

Example

The following example `menuSetup` function sets the `enabled` and `checked` states of the menu item passed to it according to whether the current document's background color is red, green, or some other color. This function uses the passed menu item's `name` attribute as the case expression of a `switch` statement that customizes the function's actions according to the particular menu item passed to it.

```
function menuSetup(menuItem) {
        // begin with menu item unchecked
        menuItem.checked = false;
        // disable dynamic menu items if no document or no element
        menuItem.enabled = (document != null && document.element != null);
        // set enabled & checked states according to background color
        var tree = document.element;
        var bodyElement = tree.getSubElement("body");

        // this example code continues on the next page
        switch (menuItem.name) {
        case "bgcolred":
           if (bodyElement)
           {
           // allow user to change color to red if isn't red now
           menuItem.enabled = (bodyElement.bgcolor != "red")
           // place checkmark next to menu item if bgcolor is red now
           menuItem.checked = (bodyElement.bgcolor == "red")
           }
           break;
        case "bgcolgrn":
           if (bodyElement)
           {
           // allow user to change color to green if isn't green now
           menuItem.enabled = (bodyElement.bgcolor != "green")
           // place checkmark next to menu item if bgcolor is green now
           menuItem.checked = (bodyElement.bgcolor == "green")
           }
           break;
        }
}
```

menuSignal

`menuSignal (menuitem)`

Availability

5.0, 6.0

GoLive calls this function when the user chooses a menu item. Your implementation of this method must respond to the user's choice of any menu item your extension provides.

To determine which menu item the user chose, your `menuSignal` function can test the `name` property of the menuitem object that GoLive passes to it. The following example `menuSignal` function uses the `menuItem.name` expression as the case expression of a `switch` statement that varies its actions according to the menu item the user chose. Each case in this example `menuSignal` function displays a customized alert that names the menu item the user chose.

```
function menuSignal(menuItem)
{
     switch (menuItem.name)
     {
        case "doThis":alert("You chose the Do Something item.");
                      break;
        case "doThat":alert("You chose the Do Something Else item.");
                      break;
        default:alert ("Something went wrong...");
     }
}
```

Undo

This section lists event-handling methods the SDK calls to do, undo, and redo an extension-specific task that has been submitted to the undo mechanism.

undoSignal

`undoSignal (`*obj*`, Number` *action*`)`

Availability

5.0, 6.0

GoLive calls this method whenever the user issues the command to do, undo, or redo the operation the undo object encapsulates.

Parameters

obj	*String*	Undo object encapsulating data required to do, undo, and redo the task this method performs.
action	*Number*	Action code indicating whether this method should do the operation for the first time, undo the operation, or redo the operation. Values are: `0` Do. Called as soon as the object is submitted. `1` Undo. `2` Redo.

Description

IMPORTANT: *Do not call* `Markup.setInnerHTML()`, `Markup.setOuterHTML()`, `Document.reparse()`, *or* `Document.reformat()` *from within an undo action if doing so will cause GoLive to reparse.*

Example

```
function undoSignal (undo,action) {
     switch (undo.kind) {
        case "Link":undoLink (undo, action); break;
        // assume we've created these additional undo actions
        case "TextColor":undoTextColor (undo, action); break;
        case "What":undoSomethingElse (undo, action); break;
     }
}
```

Because each case in an `undoSignal` function must handle three cases of its own (Do, Undo, and Redo), you may improve the readability of your code by creating your own functions that the cases of the `undoSignal` function can call to perform tasks.The next code fragment illustrates such an approach. This `undoLink` function provides the data needed to change the `url` property of a custom box, undo the change, and redo the change.

```
function undoLink (undo, action) {
     var box = boxes[undo.boxName];// get the real box behind the name
     if (action == 1) {// undo
        box.url = undo.oldURL;
     else             // do or redo
        box.url = undo.newURL;
}
```

Document Events

Your extension can implement the optional `docSignal` event in order to respond to document-related events such as creating, opening, changing, saving, and closing documents, or changing the user interface display of a document s data.

docSignal

`docSignal(`*document*`, `*event*`)`

GoLive 6.0 calls this method when document-related events occur.

Availability

6.0

Returns

`null`

Parameters

document	*Document*	Document affected by the event.	
event	*String*	The event that caused the SDK to call this method. One of the following values:	
		`new`	a new document has been created
		`opened`	an existing document has been opened.
		`save`	the document is about to be saved.
		`saved`	the document has been saved.
		`close`	the document will be closed.
		`view`	the document view has changed.
		`activate`	the document window has been activated.
		`deactivate`	the document window has been deactivated.

Document Translation

The GoLive 6.0 document-translation mechanism calls the functions this section describes. For details, see "Document Source Translation" on page 204 of the *GoLive 6.0 Extend Script SDK Programmer's Guide*.

translate

`translate(`*docURL*`, `*siteURL*`, `*source*` [, `*links*`])`

Your implementation of this method translates the document. The translator calls this global function when it detects a match with its defined file types and one of its regular expressions.

Availability

6.0

Parameters

docURL	*String*	When GoLive calls the `translate` function, it passes the URL of the document as this value.
siteURL	*String*	When GoLive calls the `translate` function, it passes the URL of the site as this value.
source	*String*	When GoLive calls the `translate` function, it passes the Unicode string containing the entire document as this value.
links	*String*	Optional. A comma-separated list of links detected in the translated content.

Description

You must implement this function to support automatic document translation.The translator calls this global function when it detects a match with its defined file types and one of its regular expressions. It receives the document URL, the site URL and the Unicode source of the document.

This function is responsible for detecting the document parts that needs translation. Each part of the document is then translated by calling the method *app.translator.translateSnippet()*.

The `translate` method is called under many circumstances. Often, it only receives the original code instead of the entire source; for example, this is.the case when the original source is changed by a call to *rewriteSnippet()*. Because *rewriteSnippet()* only changes the original code, it calls *translate()* afterwards to give the extension a chance to update the replacement code according to the changes to the original code.

inspectTranslation

`inspectTranslation (`*inspector*`, `*source*`)`

GoLive calls this function whenever the protected region that displays the replacement text is activated.

Availability

6.0

Parameters

inspector	*Inspector*	The activated inspector object
source	*String*	The original untranslated code of one translation instance (not the whole doc).

Description

Implement this function if you define an inspector for your replaced code. Your implementation of this function uses data it extracts from the *source* argument to initialize the inspector. When the inspector calls the *controlSignal()* function, you can use *app.translator.rewriteSnippet()* to change the original code.

NOTE: If your translator reports URLs, the usage of urlgetter controls in the inspector differs from the way you use it in conjunction with custom boxes.

- You cannot choose a random name for the urlgetter control; instead you have to name it `"link0"` for the first link you report during translation, `"link1"` for the second, and so on.
- Your `controlSignal` function does not respond to the urlgetter control; instead, your `linkChanged` function responds to it. Your `controlSignal` function should handle all of your inspector's other controls as it normally would.

Returns

No return value.

translationLinkChanged

`translationLinkChanged (`*source*`, `*index*`, `*newURL*`)`

Availability

6.0

Parameters

source	*String*	The original untranslated code.
index	*Number*	The link to change, specified as an index into the `links` collection of a Box object. Index position 0 holds the first link in the collection.
newURL	*String*	The URL to which this method sets the link it translates.

When a link changes, GoLive calls this function to report the change to the document-translation mechanism. Your implementation of this method updates your extension's JavaScript references to this link.

A variety of circumstances may cause a link to change and cause GoLive to call this method; for example, the user may interact with a point-and-shoot link in Site View, or with a `URLgetter` control in an **Inspector** window.

Site Events

Extend Script extensions can add custom columns to the Site window in GoLive 6. This section describes notifications GoLive sends when Site Window events occur.

- Opening or closing a site
- Selecting files in the Site window
- Custom content management:
  - adding a custom column to the site window
  - removing a custom column from the site window
  - adding cells to a custom column
  - removing cells from a custom column
  - editing the content of a cell in a custom column
  - sorting the cells in a custom column

To respond to one of these events, you must

- Define a *functionName* custom function that responds to the event. For implementation details, see the event's listing in this section.
- Register your custom function to be called for that event.
  `registerCallback (`*evtName*`, `*fnName*`)`

For an example, see "Registering Your Content Callback Function" on page 304.

onWebsiteSwitch

When the active site changes, GoLive calls all custom *siteSwitchFn* methods registered for notification of the `onWebsiteSwitch` event. To request such notification, call the SiteReport object's `registerCustomReport`() method, which registers your *siteSwitchFn* as the one to execute in response to these notifications.

Your custom *siteSwitchFn* function is of the form

```
function siteSwitchFn(...) { … }
```

Your implementation of this function can perform any task you need to perform when a new site opens. When your *siteSwitchFn* is called, the new site is available from the global `website` property.

Availability

6.0

onFileSelectionChange

When the file selection in the Files tab of the site window changes, GoLive calls all custom *fileSelFn* methods registered for notification of the `onFileSelectionChange` event. To request such notification, call the SiteReport object's `registerCustomReport` method, which registers your *fileSelFn* as the one to execute in response to these notifications.

Your custom *fileSelFn* function is of the form

```
function fileSelFn(...) { … }
```

Your implementation of this function can perform any task you need to perform when the file selection in the site window changes.

Availability

6.0

onWebsiteOpen

When GoLive opens a website, it calls all custom *siteOpenFn* methods registered for notification of the `onWebsiteOpen` event. To request such notification, call the SiteReport object's `registerCustomReport` method, which registers your *siteOpenFn* as the one to execute in response to these notifications.

Your custom *siteOpenFn* function is of the form

```
function siteOpenFn(...) { … }
```

Your implementation of this function can perform any task you need to perform when a new site window opens. Your *siteOpenFn* function can retrieve the new site from the collection in the global `website` property.

Availability

6.0

onWebsiteClose

GoLive sends this notification just before a website gets closed. At this point, the global `website` property still provides access to the site to be closed.

When GoLive closes a website, it calls all custom *siteCloseFn* methods registered for notification of the `onWebsiteClose` event. To request such notification, call the SiteReport object's `registerCustomReport` method, which registers your *siteCloseFn* as the one to execute in response to these notifications.

Your custom *siteCloseFn* function is of the form

```
function siteCloseFn(...) { … }
```

Your implementation of this function can perform any task you need to perform before the site window closes. When your *siteCloseFn* function executes, the site has not yet closed; thus, it is still available from the collection in the global `website` property.

Availability

6.0

Custom Columns in the Site Window

This section describes callback methods you implement to update custom content you've added to a Site window.

GoLive 6.0 calls these methods when it needs your extension to

- Supply data to its custom column in the Site window.
- Sort data in its custom column in the Site window.

NOTE: To update custom content without using callback functions, see "Using Named Properties to Display Custom Column Content" on page 203 of the *Extend Script SDK Programmer's Guide*.

Arguments and Return Values of Column Callback Functions

To ease your data handling, GoLive sets certain global variables before calling your event-handling functions.

- Before calling your onSiteColumnContentCallback function, GoLive makes available in the `currentReference` global variable a *siteReference* object it uses to pass arguments to your methods.
- Before calling the onSiteColumnCompareCallback function, GoLive makes available two siteReference objects, in the `referenceToCompare1` and `referenceToCompare2` global variables.

Registering Your Content Callback Function

Your implementation of the initializeModule function registers your callback methods by calling the registerCallback function, as the following example does.

```
function initializeModule()
     {
     // ...
     registerCallback ("onFilesTabContentCallback","onSiteColumnContentCallback");
     registerCallback ("onFilesTabCompareCallback","onSiteColumnCompareCallback");
     registerCallback ("onFilesTabColumnIDCallback","onSiteColumnIDCallback");
     // ...
     }
```

onSiteColumnContentCallback

`onSiteColumnContentCallback (`*colID*`, `*url*`, `*data*`)`

Your implementation of this method provides String *data* that GoLive draws in the *colID* column of the *url* site's Site window.

Availability

6.0

Parameters

colID	*Integer*	[in]	ID of the column, as returned by the `addColumn` or `insertColumn` method of the Website object.
url	*String*	[in]	URL of a site object, used as an object identifier
result	*String*	[out]	Data GoLive is to draw in the custom column.

Returns

Use the *data* parameter to return the String data GoLive is to draw in the custom column.

Description

GoLive calls this function whenever a custom column requests its content. Most calls to this method are for drawing purposes.

Example

```
function onSiteColumnContentCallback (colID, url, data)
{
   // Access the global SiteReference "currentReference"
   // and set the data string according to your own purposes

   // In this example we display the reference's mime type in the SDK column

   switch (colID)
   {
      case "9":
         data = currentReference.mimeType + " through callback";
         break;

      default:
         break;
   }
}
```

onSiteColumnCompareCallback

`onSiteColumnCompareCallback (`*colID, url1, url2, result*`)`

Your implementation of this method compares *url1* and *url2* and returns an integer value that indicates the results of the comparison.

Availability

6.0

Parameters

colID	*Integer*	[in]	ID of the column, as returned by the `addColumn` or `insertColumn` method of the Website object.
url1	*String*	[in]	URL of a site object, used as an object identifier (obj1)
url2	*String*	[in]	URL of a site object, used as an object identifier (obj2)
result	*Integer*	[out]	Compare result; return the following: `-1` (obj1 smaller obj2); `0` (obj1 equals obj2); `1` (obj1 greater obj2)

Returns

One of the following integer values, which indicate the result of the comparison:

-1	obj1 < obj2
0	obj1 == obj2
1	obj1 > obj2

Description

GoLive calls this method whenever a custom column needs to compare two objects. For example, GoLive calls this method when the user clicks in the title of a Site window column, which causes GoLive to sort the contents of the site window according to the values in that

column. Your implementation of this method sorts the column content and returns a value that indicates the result of the comparison operation.

Each column to compare is identified by the columnID value that the `addColumn` or `insertColumn` method of the Website object returned when the column was added to the Site window.

Example

```
function onSiteColumnCompareCallback (colID, url1, url2, result)
{
   // Either identify the two objects given by "url1" and "url2" or access the
   // global SiteReference's "referenceToCompare1" and "referenceToCompare2"
   // and compare them according to the criteria of your SDK column

   // In this example we compare two references by their mime types

   switch (colID)
   {
      case 9:
         if (referenceToCompare1.mimeType < referenceToCompare2.mimeType)
            result= -1;
         else if (referenceToCompare1.mimeType > referenceToCompare2.mimeType)
            result= 1;
         else
            result = 0;
         break;

      default:
         result= 0;
         break;
   }
}
```

onSiteColumnIDCallback

`onSiteColumnIDCallback (`*oldID, newID*`)`

Your implementation of this method updates the columnID values of custom columns that remain in the Site window after the SDK removes a custom column.

Availability

6.0

Parameters

oldID	*Integer*	Column ID of an SDK column as used by GoLive before the removal.
newID	*Integer*	Column ID of an SDK column as used by GoLive after the removal; you need to use this ID from now on.

Description

GoLive calls this function when the user removes a custom column from the Site window at runtime. The column is identified by its ID as returned by the `addColumn` or `insertColumn` methods of the Website object.

When the user removes a custom column, each custom column remaining in the Site window must change its column ID as specified by the arguments to this function. When a column's ID changes, all internal functions, methods, and callback methods subsequently use the *newID* value to refer to that column.

NOTE: You can ingore this method if the Site window contains only one custom column or you are not using any custom column callback procedures.

Example

```
function onSiteColumnIDCallback (oldID, newID)
{
   // GoLive calls this fn when the SDK deletes a custom column from
   // the site window at runtime. Subsequently, GoLive calls this
   // function for each of the remaining custom columns.
   // Your implementation of this function must update your
   // extension's global columnID values.
}
```

Custom Columns in the Site Report Window

To supply data to a custom column in the site report window and to reorder custom columns, you can implement the callback methods this section describes.

onReportWinColumnCompare Event

This event indicates that GoLive needs to sort the content of a custom column in the Site Report window.

Availability

6.0

Parameters

colID	*Number*	ID of the column being sorted. The SiteReport object's `addColumn` method returned this value when it added this column.
url1	*String*	URL to the first site object to compare.
url2	*String*	URL to the second site object to compare.
result	*Number*	One of the following values indicating equality or inequality of *url1* and *url2*:

`-1`	*url1.prop* < *url2.prop*
`0`	*url1.prop* == *url2.prop*
`1`	*url1.prop* > *url2.prop*

Description

GoLive sends this notification when it needs to sort your *colID* custom column in the Report Window. The URL's represent two site objects that are to be compared in some way. Your custom function extracts values from these site objects, compares the extracted values, and returns a value that ranks them ordinally.

When this event is issued, GoLive calls all *colCompareFn* methods registered for notification of the `onReportWinColumnCompare` event. To request such notification, call the `registerCustomReport` method, which registers your *colCompareFn* as the one to execute in response to this event.

Your custom *colCompareFn* function is of the form

```
function colCompareFn(colID, url1, url2, result) { … }
```

Your implementation of the *customColCompareFn* must compare the *url1* and *url2* items in some way and return a value that indicates their ordinal relationship (greater than, less than, or equal).

Example

The `itemSorter` function is registered with GoLive as the function to execute in response to the `onReportWinColumnCompare` event.

```
function itemSorter (colID, url1, url2, result)
{
     // Get some value from each of the URLs supplied
     var value1 = // (get some value from url1 here);
     var value2 = // (get some value from url2 here);

     if (value1 < value2)
        result = -1;
     else
     if (value1 > value2)
        result = 1;
     else
        result = 0;
     }
```

onReportWinColumnContent Event

This event indicates that GoLive needs to add content to a custom column in the Site Report window.

Availability

6.0

Parameters

colID	*Number*	Column ID of the column to update. The SiteReport object's `addColumn` method returned this value when it added this column.
url	*String*	The site item from which to extract the requested content.
data	*String* *Number*	A string to display in the report results window at the column/row location the *colID* and *url* arguments specify/.

Description

To fill in the content of custom columns created within the report window, GoLive calls functions that registered for `onReportWinColumnContent` notifications. Such functions should follow the format of the following example function.

When GoLive needs to add data to a custom column in the Site Report window, it calls all *colDataGetterFn* methods registered for notification of the `onReportWinColumnContent` event. To request such notification, call the SiteReport object's `registerCustomReport` method, which registers your *colDataGetterFn* as the one to execute in response to these notifications.

Your custom function is of the form

`function` *colDataGetterFn*`(`*colID*`, `*url*`, `*data*`) { … }`

When called, it supplies content to custom columns.

Example

```
function contentFiller (colID, url, data)
{
// in colID The ID of the custom column to provide content for.
// How do you know the ID of your custom column? It's
// the value returned by GoLive when the column is added.
// in url The URL of the site object to supply content for
// out data The string data which should be shown under the
// column "colID" for the URL "url".

data = "";
if (colID == foo)
     {
        // do something with the url supplied
        data = "woof!";
     }
}
```

WebDAV

When interacting with a WebDAV server, GoLive calls the event-handling methods this section describes.

`signalConnect`(*store*)
Successful connection to *store* WebDAV server established; initialize as needed.

`inspectionBegins`(*resource*)
The *resource* file or folder is selected; initialize its Inspector.

`inspectionContinues`(*resource*)
Current selection in *resource* has been edited; update Inspector.

`inspectionEnds`(*resource*)
Selection has changed; impose changes on disk-based entity *resource* represents.

NOTE: To use these functions, your extension's `initializeModule` function must implement the `setupWebDAV` function as described in "setupWebDAV" on page 312.

Initialization

To use the WebDAV features of the SDK, your extension must implement the `setupWebDAV` function.

setupWebDAV

```
setupWebDAV()
```

Modules that implement this function receive the `inspectionBegins`, `inspectionContinues`, and `inspectionEnds` messages from GoLive. An extension that adds one or more custom items to the WebDAV browser window must implement this function; its presence registers the module to receive these messages. Because GoLive calls the `setupWebDAV` function at startup time, your implementation of this function can perform appropriate initialization tasks if necessary.

Availability

6.0

Returns

`Boolean`

Connecting To The WebDAV Server

The `signalConnect` method passes to your extension the objects it calls to manipulate resources on that server. It also provides an opportunity to perform initialization tasks which require a successful connection to a WebDAV server.

signalConnect

`signalConnect(`*store*`)`

Connection to the *store* WebDAV server established successfully. Your implementation of this method performs any initialization tasks which require a live connection to a DAV server.

Availability

6.0

Parameters

store	*DAV*	DAVStore object that manages resources on the currently-connected DAV server.

Returns

boolean	`true` indicates success.

Description

When the **Connect...** button makes a successful connection to a WebDAV server, the SDK calls the `signalConnect` method, passing as its argument a `DAV:ObjectStore` object that can make or break connections with this WebDAV server and fetch data from it.

The `root` property of the `DAV:ObjectStore` object contains a `DAVResource` object. The `DAVResource` object provides methods that manipulate a disk-based resource; for example, this object can create, copy, move, lock, or unlock a file on the WebDAV server.

A typical implementation of this method assigns these DAVStore and DAVResource objects to global variables for later use; additionally, it can perform any other initialization tasks that are appropriate.

```
myDAVRsrc = "";
function signalConnect(DAVObj){
    if (DAVObj.root != null){
        myDAVRsrc = DAVObj.root;
    }
    else{
        alert ("Could not locate DAV resources.");
    }
}
```

Inspectors for Custom WebDAV Resources

An extension that adds one or more custom items to the WebDAV browser window must implement the functions this section describes. When the user manipulates files or folders in the WebDAV browser, GoLive calls these functions to support the custom item's **Inspector** window.

`inspectionBegins`(*resource*)
: The *resource* file or folder is selected; initialize its Inspector.

`inspectionContinues`(*resource*)
: Current selection in *resource* has been edited; update Inspector.

`inspectionEnds`(*resource*)
: Selection has changed; impose changes on disk-based entity *resource* represents.

NOTE: To use these functions, your extension must implement the `setupWebDAV` method; for more information, see "Initialization and Termination" on page 287.

inspectionBegins

`inspectionBegins(`*resource*`)`

The *resource* file or folder is selected; initialize its Inspector window.

Availability

6.0

Parameters

resource	*DAVResource*	DAVResource object that manages the currently-selected file or folder.

Returns

boolean	`true` indicates success.

Description

Your implementation of this method initializes the Inspector window when the custom *resource* your extension adds to the WebDAV browser is selected.

GoLive calls this method when *resource* is selected in the WebDAV browser. When *resource* is your extension's custom item, your implementation of this method initializes the Inspector controls to display current values extracted from *resource*. Your method ignores all *resource* objects which do not represent your custom additions to the WebDAV browser.

inspectionContinues

`inspectionContinues(`*resource*`)`

The current selection in the *resource* file or folder has been edited; reinitialize its Inspector.

Availability

6.0

Parameters

resource	*DAVResource*	DAVResource object that manages the currently-selected file or folder.

Returns

boolean	`true` indicates success.

Description

Your implementation of this method updates the Inspector window when the *resource* `DAVResource` object is edited in the WebDAV browser. It does not change the disk file or folder that *resource* manages—that's the job of the `inspectionEnds` method.

GoLive calls the `inspectionContinues` method when the user edits your custom *resource* in the WebDAV browser. When *resource* is your extension's custom item, your implementation of this method updates the Inspector controls to display current values extracted from *resource*. Your method ignores all *resource* objects which do not represent your custom additions to the WebDAV browser.

inspectionEnds

`inspectionEnds(`*resource*`)`

Impose on the associated disk-based file or folder the changes to *resource* specified by the `inspectionContinues` method.

Availability

6.0

Parameters

resource	*DAVResource*	DAVResource object that manages the currently-selected file or folder.

Returns

boolean	`true` indicates success.

Description

Your implementation of this method commits to disk the changes made in the custom *resource* item's Inspector window.

GoLive calls this method when the current selection in the WebDAV browser changes from *resource* to something else. Your implementation of this method imposes current Inspector window values on the disk-based entity *resource* represents. For example, when *resource* is renamed by the `inspectionContinues` method, the disk file *resource* represents is not actually renamed until the `inspectionEnds` method actually performs this task.

A Sort Order Tables

This chapter lists the values GoLive uses to sort items in the Windows menu and the Objects palette.

Window Menu Items

The entries of the Window menu are sorted groupwise. The upper two digits denote a group, while the lower two digits are the sort order within the group. Groups are separated by menu separators, which means that f.ex. all order numbers which start with 90*xx* are grouped together and separated from the other groups with a menu separator. Table A.1 lists the values used to sort Window menu items.

TABLE A.1 Codes used to sort Window menu items

Objects	0101
Color	0110
Inspector	2001
View Controller	2001
Debug	3001
Transform	3001
Align	3002
Tracing Image	3003
Floating Boxes	5001
Table	5002
Link View	7001
Source Code	7002
Javascript Shell	7003
Markup Tree	9001
History	9003

Objects Palette Entries

The objects in the **Objects** palette are sorted in two levels. The first level is the order the tabs for each group appear in, while the second value is the sort order within one group of icons. The higher the value is, the further to the right the icon or tab appears. Please note that many built-in modules install more than one icon. You cannot place your entries' icons between these icons, only to the left or right of these icons. For example, all icons in the Head section exhibit this behavior.

NOTE: You cannot add items to the **Custom** tab of the **Objects** palette.

Table A.2 lists the values used to sort **Objects** palette entries.

TABLE A.2 Codes used to sort Objects palette entries

Name	taborder	order	Icons
Basic	0		
Layout Grid		10	2
Floating Box		15	1
Table		20	1
Image		30	1
Plugin		40	5
Java™ Applet™		50	1
Object		60	1
Line		70	1
Horizontal Spacer		80	1
JavaScript		90	1
Marquee		100	1
Comment		110	1

Name	taborder	order	Icons
Anchor		120	1
Line Break		130	1
Tag		140	1
(reserved for future use)		150	1
Smart	1		
Smart items		200	all
Forms	2		
Form items		10	all
Head	3		
Head items		20000	all
Frames	4		
Frame items		20000	all
WebObjects	5		
WebObjects items		1-4	all
Site	50		
Site items		1	all
Site Extras	51		
Site extras items		2	all
QuickTime	5000		
QuickTime items		1	all
Extensions (default)	30000		
Extension items (default)		30000	

B Supported Layout Tags

This Appendix lists the markup elements (in JavaScript represented by Markup objects) for which GoLive creates a Layout object. This object's methods test and manipulate the attributes of the HTML element it manages without needing to reparse the document. Particularly useful is the `insertBox` method, which inserts supported elements without reparsing.

The values these methods accept and return vary according to the kind of markup element the Layout object manages. For each kind of markup element a Layout object can manage, the listings in this chapter specify

- valid attribute names.
- read/write availability of each attribute.
- valid arguments to pass to methods of the Layout object.
- values the methods of the Layout object can return.

To set the value of an attribute of an element managed by the *layoutObj* object, you can call its `setAttribute` method, as the following example does.

layoutObj`.setAttribute(`*attributeName*`, `*value*`, ... )`

- To determine the values that are valid *attributeName* arguments, see this Appendix's listing for the kind of element the called *layoutObj* represents.
- To determine the type of data to pass as the *value* argument, see this Appendix's listing for the kind of element the called *layoutObj* represents.

For example, imagine that you're manipulating an image in Layout view. The markup element GoLive creates to represent the `<img>` element provides a `width` attribute. Under the listing for `<img>` in this Appendix, you can see that `width` is a valid *attributeName* value to pass to methods of the Layout object that manages this element. Thus, this listing indicates that the following JavaScript code would be considered valid by the methods of this object.

```
markupObj.layout.setAttribute("width", 100)
```

Using This Appendix

Some attributes have two components, a value component and a mode component. This is equivalent to HTML's ability to accept various data types as values for a single HTML attribute. In the above example, the meaning of the `width` value may be either a fixed pixel value or a percentage of the window width. This distinction is determined by the value of the `widthmode` attribute, which is specified as either `percent` or `pixel`.

All attributes are read/write unless otherwise noted. Some are write-only because of the way they are used. The action that results from changing these write-only attributes is explained in the notes.

Be sure to enter all attribute values in quotes, *except* the Boolean values `true` and `false`. These values must *not* be quoted, or they will be interpreted as string values, which are always `true` unless they are empty.

Values are only listed fore the special values supported by the GoLive SDK. Standard HTML values of text strings, numbers, etc. are not listed. If you need more information on these allowed values, consult an HTML reference.

Elements the Layout Object Manages

GoLive creates a `Layout` object to manage the Layout View appearance of each of the markup elements this section lists.

TABLE B.1 Elements the Layout Object Supports

`spacer`

`br`

`hr`

`img`

`applet`

`a`

`form`

`label`

`button`

`input type="submit"`

`input type="reset"`

`input type="button"`

`input type="text"`

`input type="password"`

`input type="file"`

`input type="checkbox"`

`input type="radio"`

`input type="hidden"`

`input type="image"`

`textarea`

`select`

```
keygen
fieldset
embed type="audio/x-pn-realaudio-plugin"
embed type="application/x-shockwave-flash"
embed type="video/quicktime"
embed type="image/svg-xml"
embed type="audio/"
embed
marquee
body
table
td
th
tr
script
meta
isIndex
base
link
```

spacer

Attributes

type	horizontal vertical block	
width		
height		
size		Read only.
align	top middle bottom left right absmiddle absbottom texttop baseline	

br

Attributes

clear	none false left right all	

hr

Attributes

noshade	false true	
width		
height	none false	
widthmode	full false percent % pixel	Set value of width to given measurement unit.
heightmode	full false percent % pixel	Set value of height to given measurement unit.

img

Attributes

src		
lowsrc		
iccprofile		
orgsize	true	Write only. Set width and height to the original values of the image.
hspace		
vspace		
bordermode	true false	Switch on/off the attribute border.
border		
alt		
usemap	false true	Adds/removes the usemap attribute. If a <map> does not immediately follow the <img>, a new <map> is created with a generated name attribute.
mapname		Sets the corresponding attributes of the <img usemap=#> and <map name=> tags.

Supported attributes of <img> tag (*continued from preceding page*)

widthmode	pixel percent % image	Set value of width to given measurement unit.
heightmode	pixel percent % image	Set value of height to given measurement unit.
width		
height		
uselowsrc	true false	Switch on/off the attribute lowsrc.
align	top middle bottom left right absmiddle absbottom texttop baseline	
name		

applet

Attributes

code		
codebase		
name		
width		
height		
hspace		
vspace		
widthmode	pixel %	Set value of width to given measurement unit.
heightmode	pixel %	Set value of height to given measurement unit.

Supported attributes of `<applet>` tag (*continued from preceding page*)

`align`
`alt`

a

Attributes

`name`

form

Attributes

`action`	
`name`	

`method`	`default`
	`get`
	`post`

`target`	
`enctype`	

label

Attributes

`for`		
`tabindex`		
`tabindexmode`	`true` `false`	Switch on/off attribute `tabindex`.
`accesskey`		

button

Attributes

name		
value		
type	normal submit reset	
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.

input type="submit"

Attributes

name		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.
value		
valuemode	true false	Switch on/off attribute value.
type	normal submit reset	

input type="reset"

Attributes

name		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.
value		
valuemode	true false	Switch on/off attribute value.
type	normal submit reset	

input type="button"

Attributes

name		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.
value		
valuemode	true false	Switch on/off attribute value.
type	normal submit reset	

input type="text"

Attributes

name		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.
value		
size		
maxlength		
password	true false	Write only.
readonly	true false	

input type="password"

Attributes

name		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.
value		
size		
maxlength		
password	true false	Write only.
readonly	true false	

input type="file"

Attributes

size		
name		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.

input type="checkbox"

Attributes

name		
value		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.
checked	true false	

input type="radio"

Attributes

name		
checked	true false	
value		
disable	true false	
accesskey		
tabindex		
tabindexmode	true false	Switch on/off attribute tabindex.

input type="hidden"

Attributes

name		
value		
disable	true false	

input type="image"

Attributes

src		
lowsrc		
iccprofile		
orgsize		Write only. Set width and height to the original values of the image.
hspace		
vspace		
bordermode	true false	Switch on/off the attribute border.
border		
alt		

Supported attributes of `<input type="image">` tag (*continued from preceding page*)

usemap	false true	Adds/removes the usemap attribute. If a <map> does not immediately follow the <img>, a new <map> is created with a generated name attribute.
mapname		Sets the corresponding attributes of the <img usemap=#> and <map name=> tags.
widthmode	pixel percent % image	Set value of width to given measurement unit.
heightmode	pixel percent % image	Set value of height to given measurement unit.
width		
height		
uselowsrc	true false	Switch on/off the attribute lowsrc.
align	top middle bottom left right absmiddle absbottom texttop baseline	
name		

textarea

Attributes

cols		
rows		
wrap	default off virtual physical	
content		Write only.
name		

Supported attributes of `<textarea>` tag (*continued from preceding page*)

`disable`	`true` `false`	
`accesskey`		
`tabindex`		
`tabindexmode`	`true` `false`	Switch on/off attribute `tabindex`.
`readonly`	`true` `false`	

select

Attributes

`multiple`	`true` `false`	
`rows`		
`newitem`	Write only. Inserts a new element `<option value="value">item</option>` after the select element.	
`selectitem`	Write only. Value is the index of the `<option...>` element. Switch on/off the attribute `selected` of the corresponding element.	

`itemlabel`	Write only. Sets the label of the `<option...>` element. E.g., you have HTML source like this:
	`<select name="selectName" size="4" multiple>`
	`  <option value="one">hello</option>`
	`  <option selected value="three">`
	`  blah</option>`
	`</select>`
	When you run the following:
	`call layoutObject.setAttribute`
	`  ("itemlabel", 1, "test");`
	The result is to change `blah` to `test`:
	`<select name="selectName" size="4" multiple>`
	`  <option value="one">hello</option>`
	`  <option selected value="three">`
	`  test</option>`
	`</select>`	

Supported attributes of `<select>` tag (*continued from preceding page*)

`itemvalue`		Write only. Same as `itemlabel`, but sets the value of the `<option...>` element. E.g., you have HTML source like this:
		`<select name="selectName" size="4" multiple>`
		`<option value="one">hello</option>`
		`<option selected value="three">`
		`blah</option>`
		`</select>`
		When you run the following:
		`call layoutObject.setAttribute`
		`("itemvalue", 1, "test");`
		The result is to change `three` to `test`:
		`<select name="selectName" size="4" multiple>`
		`<option value="one">hello</option>`
		`<option selected value="test">`
		`blah</option>`
		`</select>`

`disable`	`true` `false`	
`tabindex`		
`tabindexmode`	`true` `false`	Switch on/off attribute `tabindex`.

keygen

Attributes

`challenge`		
`name`		
`disable`	`true` `false`	
`tabindex`		
`tabindexmode`	`true` `false`	Switch on/off attribute `tabindex`.
`accesskey`		

fieldset

Attributes

legend	true false	Write only.
align	default top bottom left right center	
accesskey		

embed type="audio/x-pn-realaudio-plugin"

Attributes

name		
width		
height		
hspace		
vspave		
widthmode	percent % pixel	Set value of width to given measurement unit.
heightmode	percent % pixel	Set value of height to given measurement unit.
usefile	true false	Switch on/off attribute src.
usemime	true false	Switch on/off attribute type.
type		
hidden	true false	
usepagelink	true false	Switch on/off attribute pluginspage.
palette	default foreground background	

Supported attributes of `<embed type="audio/x-pn-realaudio-plugin">`
(*continued from preceding page*)

`newparameter`		Write only. Adds a new attribute to the element. For example, `call layoutObject.setAttribute ("newparameter", "newparametername")`
`chgparametername`		Write only. Change the name of the attribute which was inserted with `newparameter`. For example, `layoutObject.setAttribute ("newparameter", "mypara"); layoutObject.setAttribute ("chgparametername", "mypara", "yourpara");` changes the name of the attribute `mypara` to `yourpara`.
`chgparametervalue`		Write only. Change the value of a new inserted parameter.
`delparameter`		Write only. Deletes an attribute which was inserted with `newparameter`.

`align`	`top`
	`middle`
	`bottom`
	`left`
	`right`
	`absmiddle`
	`absbottom`
	`texttop`
	`baseline`	

`autostart`	`true` `false`	
`nolabels`	`true` `false`	

`controls`	`imagewindow`
	`controlpanel`
	`playbutton`
	`playonlybutton`
	`pausebutton`
	`stopbutton`
	`rwctrl`
	`mutectrl`
	`volumeslider`
	`tacctrl`
	`homectrl`
	`infovolumepanel`
	`infopanel`
	`statusbar`
	`statusfield`
	`positionfield`	

Supported attributes of `<embed type="audio/x-pn-realaudio-plugin">`
(*continued from preceding page*)

console	default _master _unique	

embed type="application/x-shockwave-flash"

Attributes

name		
width		
height		
hspace		
vspave		
widthmode	percent % pixel	Set value of width to given measurement unit.
heightmode	percent % pixel	Set value of height to given measurement unit.
usefile	true false	Switch on/off attribute src.
usemime	true false	Switch on/off attribute type.
type		
hidden	true false	
usepagelink	true false	Switch on/off attribute pluginspage.
palette	default foreground background	
newparameter		Write only. Adds a new attribute to the element. E.g.: `call layoutObject.setAttribute ("newparameter", "newparametername")`

Supported attributes of `<embed type="application/x-shockwave-flash">`
(*continued from preceding page*)

`chgparametername`		Write only. Change the name of the attribute which was inserted with `newparameter`. For example,
		`layoutObject.setAttribute ("newparameter", "mypara");`
		`layoutObject.setAttribute ("chgparametername", "mypara", "yourpara");`
		changes the name of the attribute `mypara` to `yourpara`.

`chgparametervalue`		Write only. Change the value of a new inserted parameter.
`delparameter`		Write only. Deletes an attribute which was inserted with `newparameter`.

`align`	`top`
	`middle`
	`bottom`
	`left`
	`right`
	`absmiddle`
	`absbottom`
	`texttop`
	`baseline`	

`play`		
`loop`		
`quality`		
`scale`		

embed type="video/quicktime"

Attributes

`name`
`width`
`height`
`hspace`
`vspave`

Supported attributes of `<embed type="video/quicktime">` (*continued from preceding page*)

`widthmode`	`percent`
	`%`
	`pixel`
		Set value of width to given measurement unit.

`heightmode`	`percent`
	`%`
	`pixel`
		Set value of height to given measurement unit.

`usefile`	`true` `false`	Switch on/off attribute src.
`usemime`	`true` `false`	Switch on/off attribute type.
`type`		

`align`	`top`
	`middle`
	`bottom`
	`left`
	`right`
	`absmiddle`
	`absbottom`
	`texttop`
	`baseline`	

`hidden`	`true` `false`	
`usepagelink`	`true` `false`	Switch on/off attribute `pluginspage`.

`palette`	`default`
	`foreground`
	`background`	

`newparameter`		Write only. Adds a new attribute to the element. E.g.
		`layoutObject.setAttribute`
		`("newparameter", "newparametername")`

`chgparametername`		Write only. Change the name of the attribute which was inserted with `newparameter`. For example,
		`layoutObject.setAttribute`
		`("newparameter", "mypara");`
		`layoutObject.setAttribute`
		`("chgparametername", "mypara", "yourpara");`
		changes the name of the attribute `mypara` to `yourpara`.

`chgparametervalue`		Write only. Change the value of a new inserted parameter.
`delparameter`		Write only. Deletes an attribute which was inserted with `newparameter`.

Supported attributes of `<embed type="video/quicktime">` (*continued from preceding page*)

autoplay		
loop		
controler		
playeveryframe		
cache		
scale		
volume		
bgcolor		
target		
usehref	true false	Switch on/off the attribute href.

embed type="image/svg-xml"

Attributes

name		
width		
height		
hspace		
vspave		
widthmode	percent % pixel	Set value of width to given measurement unit.
heightmode	percent % pixel	Set value of height to given measurement unit.
usefile	true false	Switch on/off attribute src.
usemime	true false	Switch on/off attribute type.
type		

Supported attributes of `<embed type="image/svg-xml">` (*continued from preceding page*)

`align`	`top`
	`middle`
	`bottom`
	`left`
	`right`
	`absmiddle`
	`absbottom`
	`texttop`
	`baseline`	

`hidden`	`true` `false`	
`usepagelink`	`true` `false`	Switch on/off attribute `pluginspage`.

`palette`	`default`
	`foreground`
	`background`	

`newparameter`		Write-only. Adds a new attribute to the element. Ex:
		`layoutObject.setAttribute`
		`("newparameter", "newparametername")`

`chgparametername`		Write only. Change the name of the attribute which was inserted with `newparameter`. For example,
		`layoutObject.setAttribute`
		`("newparameter", "mypara");`
		`layoutObject.setAttribute`
		`("chgparametername", "mypara", "yourpara");`
		changes the name of the attribute `mypara` to `yourpara`.

`chgparametervalue`		Write only. Change the value of a new inserted parameter.
`delparameter`		Write only. Deletes an attribute which was inserted with `newparameter`.
`use_svgz`	`true` `false`	

embed type="audio/"

Attributes

`name`
`width`
`height`
`hspace`

Supported attributes of `<embed type="audio/">` (*continued from preceding page*)

vspave		
widthmode	percent % pixel	Set value of width to given measurement unit.
heightmode	percent % pixel	Set value of height to given measurement unit.
usefile	true false	Switch on/off attribute src.
usemime	true false	Switch on/off attribute type.
type		
align	top middle bottom left right absmiddle absbottom texttop baseline	
hidden	true false	
usepagelink	true false	Switch on/off attribute pluginspage.
palette	default foreground background	
mastersound	true false	
autostart		
loop		
starttime		
endtime		
volume		
controls		
newparameter		Write only. Adds a new attribute to the element. E.g.: `call layoutObject.setAttribute ("newparameter", "newparametername")`

Supported attributes of `<embed type="audio/">` (*continued from preceding page*)

`chgparametername`	Write only. Change the name of the attribute which was inserted with `newparameter`. For example,
	`layoutObject.setAttribute ("newparameter", "mypara");`
	`layoutObject.setAttribute ("chgparametername", "mypara", "yourpara");`
	changes the name of the attribute `mypara` to `yourpara`.

`chgparametervalue`	Write only. Change the value of a new inserted parameter.
`delparameter`	Write only. Deletes an attribute which was inserted with `newparameter`.

embed

Attributes

`name`		
`width`		
`height`		
`hspace`		
`vspave`		

`widthmode`	`percent`
	`%`
	`pixel`
		Set value of width to given measurement unit.

`heightmode`	`percent`
	`%`
	`pixel`
		Set value of height to given measurement unit.

`usefile`	`true` `false`	Switch on/off attribute src.
`usemime`	`true` `false`	Switch on/off attribute type.
`type`		

`align`	`top`
	`middle`
	`bottom`
	`left`
	`right`
	`absmiddle`
	`absbottom`
	`texttop`
	`baseline`	

Supported attributes of <embed> tag (*continued from preceding page*)

hidden	true false	
usepagelink	true false	Switch on/off attribute pluginspage.
palette	default foreground background	
newparameter		Write only. Adds a new attribute to the element. E.g.: layoutObject.setAttribute ("newparameter", "newparametername")
chgparametername		Write only. Change the name of the attribute which was inserted with newparameter. For example, layoutObject.setAttribute ("newparameter", "mypara"); layoutObject.setAttribute ("chgparametername", "mypara", "yourpara"); changes the name of the attribute mypara to yourpara.
chgparametervalue		Write only. Change the value of a new inserted parameter.
delparameter		Write only. Deletes an attribute which was inserted with newparameter.

marquee

Attributes

text		Write only.
width		
height		
widthmode	pixel percent %	Set value of width to given measurement unit.
heightmode	pixel percent %	Set value of height to given measurement unit.
hspace		
vspace		
color		

Supported attributes of <marquee> tag (*continued from preceding page*)

behavior	default scroll slide alternate	
direction	left right	
loop		
loopforever	false true	
scrollamount		
scrolldelay		

body

Attributes

bgcolor		
usebgcolor	true false	Switch on/off attribute bgcolor.
text		
usetext	true false	Switch on/off attribute text.
link		
uselink	true false	Switch on/off attribute link.
alink		
usealink	true false	Switch on/off attribute alink.
vlink		
usevlink	true false	Switch on/off attribute vlink.
background		
usebackground	true false	Switch on/off attribute background.

table

Attributes

width		
widthmode	pixel percent % auto	Set value of width to given measurement unit.
height		
heightmode	pixel percent % auto	Set value of height to given measurement unit.
border		
cellpadding		
cellspacing		
usebgcolor	true false	Switch on/off attribute bgcolor.
bgcolor		
align	left right center	
usebackground	true false	Switch on/off attribute background.
background		

td

Attributes

valign	default top bottom middle	
halign	default left right center	
usebgcolor	true false	Switch on/off attribute bgcolor.

Supported attributes of <td> tag (*continued from preceding page*)

bgcolor		
width		
height		
widthmode	pixel percent % auto	Set value of width to given measurement unit.
heightmode	pixel percent % auto	Set value of height to given measurement unit.
nowrap	true false	
usebackground	true false	Switch on/off attribute background.
background		

th

Attributes

valign	default top bottom middle	
halign	default left right center	
usebgcolor	true false	Switch on/off attribute bgcolor.
bgcolor		
width		
height		
widthmode	pixel percent % auto	Set value of width to given measurement unit.

Supported attributes of <th> tag (*continued from preceding page*)

heightmode	pixel percent % auto	Set value of height to given measurement unit.
nowrap	true false	
usebackground	true false	Switch on/off attribute background.
background		

tr

Attributes

valign	default top bottom middle	
halign	default left right center	
usebgcolor	true false	Switch on/off attribute bgcolor.
bgcolor		
height		
heightmode	pixel percent % auto	Set value of height to given measurement unit.

script

Attributes

name
src
language

meta

Attributes

http-equiv	
content	
mode	http http-equivalent name

isIndex

Attributes

prompt

base

Attributes

href
target

link

Attributes

href
title
name
urn
methods
rel
rev

C DAV Objects

This chapter describes objects that provide access to WebDAV features of GoLive 6.

DAV Object

Availability

6.0

The DAV object can make or break connections with a specific WebDAV server and fetch data from it, without invoking the WebDAV browser window.

Acquiring a DAV Object

When the **Connect...** button makes a successful connection to a WebDAV server, the SDK calls the signalConnect method, passing as its argument a DAV object that can make or break connections with that WebDAV server and fetch data from it.

```
signalConnect(DAVObj)
```

You can also create a new, empty DAV object by using the new operator on the DAV constructor.

```
var myDAV = new DAV;
var myDAVRsrc = myDAV.root;
```

DAV Object Properties

The root property of the DAV object provides the DAVResource Object that represents and manipulates resources on the DAV server the DAV object represents.

root	*DAVResource*	Object that manipulates a resource on the DAV server.

The root is empty until you populate it with DAVResource objects; if the DAVResource object has a live connection to the network store it represents, you can call its fetch() method to populate*DAVResourceObj*.root with DAVResource objects representing the contents of the store.

DAV Object Functions

Functions of the `DAV` object manage the connection to the DAV server and retrieve `DAVResource` objects from the server; while connected, you call functions of the DAVResource Object to work with resources on the DAV server.

connect

DAVObj`.connect(`*server, userName*`)`

Attempt to connect to the *server* store as the *userName* user.

Availability

6.0

Parameters

server	*String*	URL to the DAV server.
userName	*String*	User name.

Returns

boolean	`true` indicates success.

disconnect

DAVObj`.disconnect()`

Disconnect from the DAV server that provides the *DAVObj* volume.

Availability

6.0

Returns

boolean	`true` indicates success.

isConnected

DAVObj`.isConnected()`

Return `true` if *DAVObj* is connected.

Availability

6.0

Returns

boolean	`true` indicates that a live connection to the *DAVObj* store exists. `false` indicates that a connection could not be made or the user cancelled.

fetch

DAVObj`.fetch()`

Populate the *DAVObj*`.root` collection with `DAVResource` objects representing the contents of the *DAVObj* store.

Availability

6.0

Returns

boolean	`true` indicates success.

Description

This method requires *DAVObj* to represent an active connection to a resource on a DAV server. Before you call the `fetch` method, log into the appropriate WebDAV server if necessary. You can call the *DAVObj* object's `connect` method to do so.

DAVactivelock Object

Availability

6.0

The DAVactivelock object encapsulates locking information for a DAV resource. The presence of this object indicates that the resource is unavailable for checkout until the lock's owner (or the WebDAV server administrator) releases it.

Acquiring DAVactivelock Objects

The DAVactivelock object is not a direct DAV resource property. You use the lockdiscovery property of the DAVResource object to access the DAVactivelock object.

```
lockdiscov = DAVResourceObj["DAV:lockdiscovery"];
activelock = lockdiscov[0];
```

DAVactivelock Object Properties

Properties of the `DAVactivelock` object hold values that govern the lock status of a DAV resource. These values specify the scope, type, depth, owner, and timeout of the lock.

`lockscope`		Specifies how the lock affects individual users, groups, and the administrator.
`locktype`		Specifies whether the lock has an expiration date.
`depth`		Specifies whether the lock affects this folder only or affects subfolders too.
`owner`		The owner of the lock; only a lock's owner or the server administrator can release a lock before its expiration date.
`timeout`	*Date*	The time at which this lock releases automatically.

DAVactivelock Object Functions

describe

DAVactivelockObj`.describe()`

Returns a text description of the lock status of the resource the *DAVactivelockObj* manages.

Availability

6.0

Returns

String	Description of the lock status of the resource this object manages.

toString

DAVactivelockObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace. For example, the `toString` method of the `DAVactivelock` object returns the `"DAV:activelock"` string.

DAVcreationdate Object

Availability

6.0

The `DAVcreationdate` object encapsulates a text string that is the creation date of a file or folder on a DAV server. This string is guaranteed to be convertible to a JavaScript Date object.

Acquiring DAVcreationdate Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVcreationdate` object from a `DAVResource` object.

```
DAVResourceObj["DAV:creationdate"];
```

DAVcreationdate Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*Number*	Text content of the current document's `creationdate` property.: nominally, the number of seconds elapsed since 1/1/1970. You can set this value to change this content in the source document.

DAVcreationdate Object Functions

toString

DAVcreationdateObj`.toString()`

Returns a string representation of the object's creation date.

Availability

6.0

Returns

String	String representation of the object's creation date. This string is suitable for initializing a JavaScript Date object.

DAVdisplayname Object

Availability

6.0

The DAVdisplayname object encapsulates the display text that labels this resource in the WebDAV browser.

Acquiring DAVdisplayname Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVdisplayname` object from a `DAVResource` object.

DAVResourceObj`["DAV:displayname"];`

DAVdisplayname Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the current document's `displayname` property. You can set this value to change this content in the source document.

DAVdisplayname Object Functions

toString

DAVactivelockObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace. For example, the `toString` method of the `DAVactivelock` object returns the `"DAV:activelock"` string.

DAVgetcontentlanguage Object

Availability

6.0

The `DAVgetcontentlanguage` object encapsulates a String that describes the spoken language (English, Deutsche, and so on) in which the content of the resource is written.

Acquiring DAVgetcontentlanguage Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVgetcontentlanguage` object from a `DAVResource` object.

```
DAVResourceObj["DAV:getcontentlanguage"];
```

DAVgetcontentlanguage Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the current document's `getcontentlanguage` property. You can set this value to change this content in the source document.

DAVgetcontentlanguage Object Functions

toString

DAVactivelockObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace, such as the `"DAV:activelock"` string.

DAVgetcontentlength Object

Availability

6.0

The `DAVgetcontentlength` object encapsulates a value describing the length of the resource.

Acquiring DAVgetcontentlength Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVgetcontentlength` object from a `DAVResource` object.

```
DAVResourceObj["DAV:getcontentlength"];
```

DAVgetcontentlength Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the current document's `getcontentlength` property. You can set this value to change this content in the source document.

DAVgetcontentlength Object Functions

toString

DAVactivelockObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace. For example, the `toString` method of the `DAVactivelock` object returns the `"DAV:activelock"` string.

DAVgetcontenttype Object

Availability

6.0

The `DAVgetcontenttype` Object encapsulates a String that describes the type of content this resource provides (SWF, RA, and so on.)

Acquiring DAVgetcontenttype Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVgetcontenttype` object from a `DAVResource` object.

DAVResourceObj`["DAV:getcontenttype"];`

DAVgetcontenttype Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the current document's `getcontenttype` property. You can set this value to change this content in the source document.

DAVgetcontenttype Object Functions

toString

DAVactivelockObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace, such as the `"DAV:activelock"` string.

DAVgetetag Object

Availability

6.0

The DAVgetetag Object retrieves the resource's digital signature, also known as its eTag.

Acquiring DAVgetetag Objects

Use the DAVResource object's index operator [] to retrieve other DAV objects by name; for example, the following line of code retrieves a DAVgetetag object from a DAVResource object.

```
DAVResourceObj["DAV:getetag"];
```

DAVgetetag Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

description	*String*	A proprietary description of the property, suitable for display to the user.
value	*DAVPCdata*	Text content of the current document's getetag property. You can set this value to change this content in the source document.

DAVgetetag Object Functions

toString

DAVactivelockObj.toString()

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace. For example, the toString method of the DAVactivelock object returns the "DAV:activelock" string.

DAVgetlastmodified Object

Availability

6.0

The `DAVgetlastmodified` object encapsulates a text string that is the modification date of a file or folder on a DAV server. The date this object encapsulates is guaranteed to be convertible to a JavaScript Date object.

Acquiring DAVgetlastmodified Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVgetlastmodified` object from a `DAVResource` object.

```
DAVResourceObj["DAV:getlastmodified"];
```

DAVgetlastmodified Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*Number*	Text content of the current document's `getlastmodified` property.: nominally, the number of seconds elapsed since the document was last modified. You can set this value to change this content in the source document.

DAVgetlastmodified Object Functions

toString

DAVgetlastmodifiedObj`.toString()`

Returns a String representation of the document's modification date.

Availability

6.0

Returns

String	String representation of the object's getlastmodified date. This string is suitable for initializing a JavaScript Date object.

DAVlockdiscovery Object

Availability

6.0

The `DAVlockdiscovery` object retrieves the locked status of a resource on the DAV server, returning `true` if the resource is locked.

Acquiring DAVlockdiscovery Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVlockdiscovery` object from a `DAVResource` object.

DAVResourceObj`["DAV:lockdiscovery"];`

DAVlockdiscovery Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`[num]`	*String*	The DAVlockdiscovery object is a container for DAVactivelock objects that can be retrieved by ordinal index number.
`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the current document's `lockdiscovery` property. You can set this value to change this content in the source document.

DAVlockdiscovery Object Functions

toString

DAVlockdiscoveryObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace. For example, the `toString` method of the `DAVactivelock` object returns the `"DAV:activelock"` string.

DAVlockentry Object

Availability

6.0

Presence of the `DAVlockentry` object makes its resource read-only (i.e. locked for writing, but not for reading).

Acquiring DAVlockentry Objects

Use the `DAvsupportedlock` object's index operator `[]` to retrieve DAVlockentry objects by ordinal index

```
var DAVResourceObj = new DAV();
DAVResourceObj = DAVResourceObj.root;
var lock = DAVResourceObj["DAV:supportedlock"];
var lockentry = lock[num];
```

DAVlockentry Object Properties

`lockscope`	*DAVPCdata*	Possible values are `"unset"`, `"shared"`, and `"exclusive"`.
`locktype`	*DAVPCdata*	Possible values are `"unset"`, and `"write"`.
`value`	*DAVPCdata*	Text content of the XML document's `lockentry` property.

DAVpcdata Object

Availability

6.0

The `DAVpcdata Object` encapsulates String content that appears between an element's start and end tags. You can set this value to change this content in the source document.

To encapsulate all other data types, GoLive creates a DAVUnknownProperty Object.

Acquiring DAVpcdata Objects

DAV objects that encapsulate text data provide a `value` property that returns the text content `(#PCDATA)` of the XML element the `DAV`*xx* object encapsulates.

DAVpcdata Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the XML element that this `DAVpcdata` object references You can set this value to change this content in the source document.

DAVResource Object

Availability

6.0

The `DAVResource` object manipulates a disk-based resource on a DAV server; for example, this object can create, copy, move, lock, or unlock a file on the WebDAV server. This object also manages this resource's appearance in the WebDAV browser, allowing you to add your own custom properties for display in the WebDAV browser.

Acquiring a DAVResource Object

The root property of a DAV object provides the DAVResource object. The SDK passes a DAV object as the argument to your signalConnect event-handling method, which can retrieve this object as the following example does.

```
// global var to hold resource
var myDAV;
var myDAVRsrc;

function signalConnect(DAVObj){
   if (DAVObj.root != null){
      myDAVRsrc = DAVObj.root;
   }
   else{
      alert ("Could not connect to DAV server.");
   }
}
```

You can also create your own DAV object by using the new operator on the DAV constructor; subsequently, you can retrieve a DAVResource object from the DAV object's root property, as the following example does.

```
var myDAV= new DAV;
var myDAVRsrc = myDAV.root;
if (myDAVRsrc.connect("someserver", "name");
   myDAVRsrc.fetch();
){
```

The root is empty until you populate it with DAVResource objects; if the DAVResource object has a live connection to the network store it represents, you can call its fetch() method to populate*DAVResourceObj*.root.collection with DAVResource objects representing the contents of the store.

DAVResource Object Properties

The DAVResource object's description property provides a text description of the site resource the object represents.

To determine whether a DAVResource object represents a file or a folder, test its collection property; the value of the collection property is true if the DAVResource object represents a folder. The length property specifies the number of items in the folder. You can pass string or

numeric arguments to the [ ] operator to retrieve objects from this collection by type or by position, and call their methods to manipulate the contents of the resources they represent.

```
If (myDAVRsrc.collection == true){
     for (i =0; i < myDAVRsrc.length; i++){
        res = myDAVRsrc.collection[i];
        if (res["DAV:getcontenttype"].value == "image/jpeg"){
           dav.root.createFile(i + ".html",
                               "<html>…<img href=" + res.url + "…</html>");
        }
     }
}
```

[*n*]	*Number*	Numeric index position 0 is the first element in the collection.
[*propName*]	*String*	Named property of a DAV object.
[*filename*]	*String*	Filename, expressed as a relative or fully-qualified URL
`collection`	*DAVResourceCollection*	Collection of `DAVResource` objects representing contents of a folder. Undefined or 0 if the referenced disk resource is not a folder. Like all collections, this object's `length` property indicates the number of elements in the collection.
`description`	*String*	XML representation of all properties as in WebDAV.
`href`	*String*	URL to display in the anchor that places the DAV Resource on the page.
`props`	*String*	Description of the properties of this resource, suitable for display to the user. Also known as the resource's **metadata**.
`url`	*String*	URL to the DAV Resource this object manipulates.

DAVResource Object Functions

DAVResourceObj.`appendProperty`(*propName*, *nameSpace*)
Add *propName* property to *DAVResourceObj* in specified XML *nameSpace*

DAVResourceObj.`copy`(*fileToCopy*)
Create a new copy of the *fileToCopy* file at the *DAVResourceObj* location.

DAVResourceObj.`createFile`(*filename*, *content*)
Creates the *filename* file on the *DAVResourceObj* store.

DAVResourceObj.`createFolder`(*foldername*)
Creates the *foldername* folder on the *DAVResourceObj* store.

DAVResourceObj.`destroy`()
Delete all files in the *DAVResourceObj* folder.

DAVResourceObj.`download`(*fileObj*)
Download *DAVResourceObj* file on a DAV server to *fileObj* on the local file system.

DAVResourceObj.`fetch`()
Populates *DAVResource*`.root.collection` with `DAVResource` objects representing the contents of the store.

DAVResourceObj.getDocument()
Creates a document object from the *DAVResourceObj* resource.

DAVResourceObj.lock()
Locks the *DAVResourceObj* resource.

DAVResourceObj.move(*collection*)
Moves the *DAVResourceObj* file into the *collection* folder.

DAVResourceObj.openDocument()
Opens the *DAVResourceObj* resource in a new document window.

DAVResourceObj.refresh()
Replace the cached *DAVResourceObj* with a new copy read from the server.

DAVResourceObj.rename(*newName*)
Attempt to rename the *DAVResourceObj* resource to *newName.*

DAVResourceObj.saveProperties()
Save current properties of *DAVResourceObj*.

DAVResourceObj.saveProperty(*nameSpaceExpandedPropName*)
Save *nameSpaceExpandedPropName* property.

DAVResourceObj.unlock()
Attempts to unlock the *DAVResourceObj* resource.

DAVResourceObj.upload()

appendProperty

DAVResourceObj.`appendProperty`(*propName*, *nameSpace*)

Add *propName* property to *DAVResourceObj* in specified XML *nameSpace*.

Availability

6.0

Parameters

propName	*String*	Name of property to append.
nameSpace	*String*	XML name space in which to add the property.

Returns

boolean	`true` indicates success.

copy

DAVResourceObj.`copy`(*fileToCopy*)

Create a new copy of the *fileToCopy* file at the *DAVResourceObj* location.

Availability

6.0

Parameters

fileToCopy	*String*	Path to the file to copy, as a URL.

Returns

boolean	`true` indicates success.

createFile

DAVResourceObj.`createFile`(*filename, content*)

Creates the *filename* file on the *DAVResourceObj* store.

Availability

6.0

Parameters

filename	*String*	Name of file to create, expressed as a URL ending in a filename.
content	*String*	Content of file to create, expressed as a string.

Returns

boolean	`true` indicates success.

Example

```
// connect to a WebDAV server
dav = new DAV;
dav.connect(“someserver”, “name”);
dav.fetch();
// get a document from a DAV resource
doc = dav.root[“/info.xml”]. getDocument();
// Do something with the document’s DOM
outerXML = doc.element.getOutHTML();
// create a new file with the new data
dav.root.createFile(“file.html”, outerXML);
// Show the newly-created file
dav.root[“file.html”].openDocument();
```

createFolder

DAVResourceObj.`createFolder`(*foldername*)

Creates the *foldername* folder on the *DAVResourceObj* store.

Availability

6.0

Parameters

foldername	*String*	Name of folder to create, expressed as a URL that ends in a slash (indicating a directory name, rather than a file name.) Partial URLs are interpreted as relative to the current working directory.

Returns

boolean	`true` indicates success.

destroy

DAVResourceObj`.destroy()`

Delete from the server the entity the *DAVResourceObj* represents.

Availability

6.0

Returns

boolean	`true` indicates success.

download

DAVResourceObj`.download(`*fileObj*`)`

Download the *DAVResourceObj* source file from the DAV server to *fileObj* on the local file system.

Availability

6.0

Parameters

fileObj	*File*	Newly-constructed file object to hold the contents of the downloaded file.

Returns

boolean	`true` indicates success.

Example

```
// connect to WebDAV server and fetch root resources
dav = new DAV;
dav.connect("someserver", "name");
dav.fetch();

// Download the selected resource to the C:/index.html file.
res = dav.root[“/index.html”]
file = new JSXFile("C:");
res.download(file);
```

fetch

DAVResourceObj`.fetch()`

Populate*DAVResource*`.root.collection` with `DAVResource` objects representing the contents of the store.

Availability

6.0

Returns

boolean	`true` indicates success.

getDocument

DAVResourceObj`.getDocument()`

Creates a document object from the *DAVResourceObj* resource.

Availability

6.0

Returns

Document	Document object created from the *DAVResourceObj* disk resource.

Description

This method does not display the document in a window; in this regard, it's similar to the `openMarkup` method of the `Document` object.

lock

DAVResourceObj`.lock()`

Locks the *DAVResourceObj* resource.

Availability

6.0

Returns

boolean	`true` indicates success.

move

DAVResourceObj.move(*collection*)

Moves the *DAVResourceObj* file into the *collection* folder.

Availability

6.0

Parameters

collection	*Type*	Root resource (DAV.root.collection) of the folder into which this method moves the *DAVResourceObj* file.

Returns

boolean	true indicates success.

openDocument

DAVResourceObj.openDocument()

Opens the *DAVResourceObj* resource in a new document window.

Availability

6.0

Returns

boolean	true indicates success.

refresh

DAVResourceObj.refresh()

Replace the cached *DAVResourceObj* with a new copy read from the server.

Availability

6.0

Returns

boolean	true indicates success.

rename

DAVResourceObj`.rename(`*newName*`)`

DAVResourceObj`.rename(`*newName*`)`

Attempt to rename the *DAVResourceObj* resource on disk to *newName.*

Availability

6.0

Parameters

newName	*String*	New name of the *DAVResourceObj* file or folder.

Returns

boolean	`true` indicates success.

Description

Requires a live connection to the *DAVResourceObj* store.

saveProperties

DAVResourceObj`.saveProperties()`

Save current properties of *DAVResourceObj*.

Availability

6.0

Returns

boolean	`true` indicates success.

saveProperty

DAVResourceObj`.saveProperty(`*nameSpaceExpandedPropName*`)`

Save *nameSpaceExpandedPropName* property.

Availability

6.0

Parameters

nameSpaceExpandedPropName	*String*	Property name specifying the XML namespace in which it resides.

Returns

boolean	`true` indicates success.

unlock

DAVResourceObj`.unlock()`

Attempts to unlock the *DAVResourceObj* resource.

Availability

6.0

Returns

boolean	`true` indicates success.

upload

DAVResourceObj.upload(*fileURL*, *askToOverwrite*)

Upload the *fileURL* file to the *DAVResourceObj* server, optionally prompting the user to confirm file overwrites.

Availability

6.0

Parameters

file	*String*	URL of the file to upload.
askToOverwrite	*Boolean*	Behavior of file-overwrite user prompts.
	`true`	Get user confirmation before replacing an existing file.
	`false`	Overwrite files without prompting the user.

Returns

boolean	`true` indicates success.

Example

```
// connect to WebDAV server and fetch root resources
myDav = new DAV;
myDav.connect("someserver", "name");
myDav.fetch();

// Upload C:\\index_new.html file to root of DAV server,
// overwrite dupes without user prompt
myFile = new JSXFile("C:\\index_new.html");
myDav.root.upload(myFile, false);
```

DAVresourcetype Object

Availability

6.0

The `DAVresourcetype` object describes whether a DAV resource is a file or a folder.

Acquiring DAVresourcetype Objects

Use the `DAVResource` object's index operator `[]` to retrieve other DAV objects by name; for example, the following line of code retrieves a `DAVresourcetype` object from a `DAVResource` object.

```
DAVResourceObj["DAV:resourcetype"];
```

DAVresourcetype Object Properties

This object's properties provide access to the value it encapsulates and a text description of that value.

`description`	*String*	A proprietary description of the property, suitable for display to the user.
`value`	*DAVPCdata*	Text content of the current document's `resourcetype` property. You can set this value to change this content in the source document.

DAVresourcetype Object Functions

toString

DAVactivelockObj`.toString()`

Returns a namespace-qualified representation of the XML property the *DAVactivelockObj* encapsulates.

Availability

6.0

Returns

String	Description of the *DAVactivelockObj* property in the XML namespace.

Description

This method returns a description of the *DAVactivelockObj* property qualified in the XML namespace. For example, the `toString` method of the `DAVactivelock` object returns the `"DAV:activelock"` string.

DAVStatus Object

Availability

6.0

The DAVStatus Object provides status reporting information returned from the DAV server. This object also manipulates the Status Reporting window.

Acquiring a DAVStatus Object

The WebDAV SDK stores the state of the last action in a global status object. To access this object you have to create a DAVStatus object:

```
status = new DAVStatus;
```

DAVStatus Object Properties

Properties of the DAVStatus object provide information about the state of the last action. For example if the last action ended in an error state, `status.isError` evaluates to `true`. If so, you might call the status object s `writeLog` method to log the error to the console:

```
if (status.isError == true)
   status.writeLog();
```

The `errorNumber` property returns the error number of the last action, and you can use its value to condition your script s actions.

```
if (status.errorNum == 404)
   doSomething();
```

`errorNumber`	HTTP Error code returned from the DAV server.
`errorText`	Short error description; appears in the Network Status window
`errorTextLong`	Long error description.
`isError`	`true` indicates that the last action returned an error.
`isTypeError`	true indicates that the last action ended due to a type error
`isTypeNotFoundError`	true indicates that the WebDAV resource cannot be found; usually, because the resouce has been deleted, moved, or renamed.
`isTypeStatus`	Status message
`isTypeSuspicius`	true indicates that the last action should be verified (WebDAV server replied a multistatus)
`isTypeUserAbort`	true indicates that the user aborted the last action
`isTypeWarning`	This is a type warning

DAVsupportedlock Object

Availability

6.0

The `DAVsupportedlock Object` provides a listing of the lock capabilities supported by the resource.

DAVUnknownProperty Object

Availability

6.0

This object represents a DAV resource property of a type not defined in the WebDAV DTD. GoLive also returns this object to represent a newly-created “empty” property that has no value.

DAVUnknownProperty Object Properties

You can use the `DAVUnknownProperty` object’s index operator `[]` to retrieve named properties this object encapsulates.

```
var myTest = DAVUnknownPropertyObj["c:test"]
```

DAVUnknownProperty Object Functions

The appendPCData method appends a simple text strings to a `DAVUnknownProperty` object; to add new properties to the object, call its `appendProperty` method. You can use these methods to construct trees of properties programmatically.

DAVUnknownPropertyObj.`appendProperty`(*propName*, *nameSpace*)
: Create the *propName* property in the *DAVUnknownPropertyObj* resource.

DAVUnknownPropertyObj.`appendPCDATA`(*dataString*)
: Assign the *dataString* to the *DAVUnknownPropertyObj* property.

D C and C++ APIs For Use In External Binary Libraries

This chapter describes C-language macro functions that the `JSA.h` interface (header) file provides.

The functions in a compiled external library call these functions to

- obtain arguments from the GoLive environment
- pass return values back to GoLive.

For a guide to the use of these functions, see Appendix A, “Using External Libraries.”

Included Files

The `JSA.h` file required to access the functions and data types this chapter describes is available in the **Binary API** code sample folder the SDK provides. You can copy this file into your development directory or configure your development environment to include files residing in the **Binary API** folder.

Determining Whether An External Library Is In Memory

The SDK makes currently-running modules available by name in the JavaScript global namespace; thus, the JavaScript name of a module that has not been loaded is undefined. You can test this condition as the following line of JavaScript does.

```
If (typeof moduleName == "undefined")
   alert ("Cannot find external library.");
```

For example, if your function were built into an external file named `JSASample`, this test would look like the following line of JavaScript.

```
if (typeof JSASample == "undefined")
```

C and C++ API Synopsis

This section summarizes use of the C API. The C++ API is a superset of the C API. A C++ library is written, built, and used in exactly the same manner as a C library. For more detail, see "Implementing External Binary Libraries" on page 234.

```
// include JSA header for C or C++
#include "JSA.h" // use "JSA++.h" for C++ implementations

// include any other libs your extension requires
#include <math.h>

// include platform-specific drawing support
#ifdef WIN32
   #include <windows.h>
#else
   #include <QuickDraw.h>
#endif

// Call this macro once to initialize the JavaScript environment.
JSA_INIT

// define your functions
// use argc/argv pairs to pass args
// use returnValue to return this fn's result
static void power(int argc, JSValue *argv, JSValue returnValue)
{
   double a, b, c;
   // call JSAXxx fns to extract args passed by JavaScript callers
   a = JSAValueToDouble(argv[0]);
   b = JSAValueToDouble(argv[1]);

   c = pow (a, b);

   // call JSAXxx fns to pass values back to JavaScript callers
   JSADoubleToValue(returnValue, c);
}

// you must implement this fn, which registers your lib's fns with GoLive
void JSAMain() {
   JSARegisterFunction ("power", power);

}
/*************************************************************************/
/* Assuming you've built this fn into a library named JSASample.dll (or */
/* just JSASample for Mac OS platforms) which resides in the            */
/* "Modules/Extend Scripts/Common" folder, your JavaScript code calls   */
/* the external library function as follows:                             */
/*************************************************************************/
      JSASample.power (2,3);
```

Data Types

This section describes the C-language data types GoLive provides for use by external binaries.

JSValue pointer

The `JSValue` type is an opaque `void` pointer that is a data element in the *argv* vector GoLive passes to an externally-defined binary function. Internally, GoLive casts this pointer's type as necessary to hold each element's data.

```
typedef void *JSValue;
```

JSAValueType Scalar Types

The following macros define `JSAValueType` scalar data types as returned by the `JSAGetValueType` function.

Undefined

`JSA_UNDEFINED`

The undefined or empty value.

Boolean

`JSA_BOOL`

A `Boolean` value, expressed as an integer. `0` represents a value of `false`, and `1` represents a value of `true`.

Integer

`JSA_INTEGER`

A 32-bit signed integer quantity.

Floating Point

`JSA_DOUBLE`

An 8-byte, double-precision, floating-point value.

String

`JSA_STRING`

A null-terminated ASCII string.

JSANativeMethod Type

The GoLive external binary API encodes your library's function definitions as `JSANativeMethod` structures. This data type implements the following data structures, which your functions use to implement their parameters.

argc	*Integer*	Number of array elements in the **argv* vector.
**argv*	*JSValue*	Vector of argument values
return	*JSValue*	Empty pointer passed to your function by GoLive when it is called. C functions call the appropriate `JSA`*Xx*`ToValue` function to store a return value in this pointer. C++ functions can access this pointer directly or they can call a `JSA`*Xx*`ToValue` function.

JSADrawInfo Struct

The `JSADrawInfo` struct is used to implement C or C++ drawing functions that can be called from JavaScript. For more information, see "Drawing Function Examples" on page 238.

```
typedef struct _JSADrawInfo {
     long context;// a DC (Windows) or a GrafPort (Mac)
     long left, top;// upper left corner of the drawing rect
     long right, bottom;// lower right corner of the drawing rect
} JSADrawInfo;
```

Initialization and Termination Functions

This section describes functions used to initialize or terminate an external binary library.

JSA_INIT

```
JSA_INIT // macro
```

The `JSA_INIT` macro must appear exactly once in the implementation of a binary extension. This macro call must appear after the `JSA.h` or `JSA++.h` include statement and before the required call to the `JSAMain` function.

The `JSA_INIT` macro inlines the `JSAEntry` function. GoLive tests for the presence of the `JSAEntry` function to determine whether an external binary is intended for use by Extend Script extensions; thus, if the external binary does not call the `JSA_INIT` macro, GoLive does not make the external binary available to Extend Script extensions.

The `JSAEntry` function

- sets an environment pointer containing references to the various `JSA`*Xx* functions this Appendix describes.
- calls the `JSAMain` function.

JSAMain

```
JSAMain()
```

Every external binary must implement the `JSAMain` function.

Your implementation of this function must register your external binary's functions with the GoLive JavaScript engine. To do so, it calls the `JSARegisterFunction` function once for each function it registers. Optionally, your implementation of the `JSAMain` function can perform any additional initialization tasks your external binary requires.

JSARegisterFunction

`JSARegisterFunction(name,foo)`

Register the function *foo* under the name specified by the value of the *name* parameter.

The `JSARegisterFunction` function makes an external binary function available under a specified JavaScript name. Only registered `JSANativeMethod` functions are available to Extend Script extension modules.

To register a function, call the JSARegisterFunction function from within the body of the external library's `JSAMain` function. You must call `JSARegisterFunction` function once for each function to be registered.

Parameters

name	*String*	The name under which the foo function is to appear in the JavaScript namespace. This name must observe JavaScript naming conventions.
foo	*Token*	The token that the function definition associates with the function's implementation. For example, the code `myFunk () {return;}` associates the `myFunk` token with the `{return;}` implementation.
return	*Integer*	Optional. An integer value to return instead of the result the call to this function actually returns.

JSAExit

`JSAExit()`

GoLive calls your external binary's JSAExit function when the extension module that uses it about to be unloaded. Your implementation of this optional method perform housekeeping tasks required to exit the extension, such as setting the values of your pointers and variables to `null`.

Parameters

None.

Returns

`Void`.

Accessor Functions

C-language library functions must use these accessors to extract arguments from GoLive and return values to GoLive. C++ functions can access the C++ objects in the GoLive engine directly, or they can use these accessors in the same way that C-language functions do.

JSAGetValueType

`JSAGetValueType(arg)`

Returns an integer value that indicates the type of the *arg* argument.

Synopsis

```
#include "JSA.h"
. . .
JSAGetValueType(arg, return)
```

Parameters

arg	*JSAValue*	The JSA value to test
return	*Integer*	Optional. An integer value to return instead of the result the call to this function actually returns.

Returns

One of the following integer values indicating the type of the JSA object passed as the value of the *arg* parameter:

0	Undefined
1	Boolean
2	Integer
3	Double (double-precision floating point)
4	Text

Example

```
static void myFn(int argc, JSValue *argv, JSValue returnValue)
{
     int a, b, c;
     a = JSAGetValueType(argv[0]);
     JSAIntToValue(returnValue, a);
}
void JSAMain() {
     JSARegisterFunction ("myFn", myFn);
```

JSAValueToInt

`JSAValueToInt(arg)`

Returns the value of the *arg* parameter as a 32-bit integer.

Synopsis

```
#include "JSA.h"
. . .
JSAValueToInt(arg)
```

Parameters

arg	*JSAValue*	The JSA value to test
return	*Integer*	Optional. An integer value to return instead of the result the call to this function actually returns.

Returns

32-bit Integer

JSAValueToBool

`JSAValueToBool(arg)`

Return the argument as a boolean (an integer, either zero or nonzero).

Parameters

arg	*JSAValue*	The JSA value to test
return	*Integer*	Optional. An integer value to return instead of the result the call to this function actually returns.

JSAValueToString

`JSAValueToString(arg)`

Return the argument as a zero-terminated ASCII string.

Parameters

arg	*JSAValue*	The JSA value to test
return	*Integer*	Optional. An integer value to return instead of the result the call to this function actually returns.

JSAValueToDouble

`JSAValueToDouble(arg)`

Return the argument as an eight-byte floating point value.

Parameters

arg	*JSAValue*	The JSA value to test
return	*Integer*	Optional. An integer value to return instead of the result the call to this function actually returns.

JSAIntToValue

`JSAIntToValue(arg, val)`

Store the long value *val* into *arg*.

Parameters

arg	*JSAValue*	The `JSAValue` object that is to hold the *val* argument.
val	*Integer*	32-bit integer value to store in the *arg* object.

JSABoolToValue

`JSABoolToValue(arg, val)`

Store the integer value *val* into *arg*. Non-zero values of *val* specify a `Boolean` value of `true`.

Parameters

arg	*JSAValue*	The `JSAValue` object that is to hold the *val* argument.
val	*Integer*	32-bit integer value to store in the *arg* object. Non-zero values of *val* specify a `Boolean` value of `true`.

JSAStringToValue

`JSAStringToValue(arg, val)`

Store the value stored in the zero-terminated ASCII string variable pointed to by *val* into *arg*.

Parameters

arg	*JSAValue*	The `JSAValue` object that is to hold the *val* argument.
val	*String*	Zero-terminated ASCII string to store in the *arg* object.

JSADoubleToValue

`JSADoubleToValue(arg, val)`

Store the double value *val* into *arg*.

Parameters

arg	*JSAValue*	The `JSAValue` object that is to hold the *val* argument.
val	*Number*	Double-precision floating-point value to store in the *arg* object.

JSAUndefinedToValue

`JSAUndefinedToValue(arg)`

Set the value of *arg* to undefined.

Parameters

arg	*JSAValue*	The `JSAValue` object that is to hold the *val* argument.

Other Functions

These functions enable an external C or C++ function to evaluate JavaScript expressions in the calling extension's JavaScript scope; provide processor time to other extensions during lengthy operations; and generate its own JavaScript runtime errors.

JSAEval

`JSAEval(scriptlet, returnValue, timeout)`

Evaluate a specified JavaScript scriptlet in the current execution context (inside the JavaScript function that called the external C function that calls the `JSAEval` function).

Parameters

scriptlet	*String*	The expression the JavaScript engine is to interpret.

Parameters to `JSAEval()` function (*continued from preceding page*)

returnValue	*JSValue*	A `JSValue` pointer that can hold a value to return to the caller. Typically, the `JSAEval` function is built into a `foo` function in an external binary that JavaScript code in an Extend Script module calls. When the `foo` function is called in this way, GoLive supplies a valid *returnValue* pointer to the `foo` function. The body of the `foo` function can pass this pointer as the *returnValue* argument when calling the `JSAEval` function. Passing `NULL` as this value causes the `JSAEval` function to discard the result of scriptlet evaluation. When `JSAEval` completes, this pointer contains the result of evaluating the *scriptlet* argument if `foo` did not pass `NULL` as the *returnValue* argument. The body of the `foo` method can extract this data by passing the `returnValue` pointer to a `JSAValueTo`*Xx* function. If you do not intend the result of scriptlet evaluation to be the `foo` function's return value, the `foo` function must clear the *returnValue* pointer the `JSAEval` function returns before the `foo` function exits. To set the `foo` function's result, pass this pointer to a `JSA`*Xx*`ToValue` function; for example, passing this pointer to the `JSAUndefinedToValue` function causes the `foo` function to return an undefined result.
timeout	*Integer*	Positive values specify the number of milliseconds to wait for the call to complete. If the timeout elapses, the engine generates a runtime error. A value of 0 causes the engine to run the scriptlet asynchronously; that is, the call returns immediately, regardless of whether the scriptlet completes execution successfully. A *timeout* value that is less than zero causes the caller to wait unconditionally.

Returns

No return value.

Optionally, the function that calls the `JSAEval` function uses its *returnValue* parameter to specify the result of a called external function. For more information, see the description of the *returnValue* parameter, as well as the following example.

Example

External library functions use the `JSAEval()` function to evaluate a text string as a JavaScript expression. The SDK evaluates the expression in the execution scope of the JavaScript module that calls the external library. For a discussion of scope, see "JavaScript Scope of Name Attribute Values" on page 396.

C and C++ functions in external libraries can use the `JSAEval` function to

- Retrieve the values of JavaScript properties from within the scope of the calling module.
- Execute a JavaScript function call within the scope of the calling module.

  You can't pass arguments of any kind, but the scriptlet is evaluated in the context of the JavaScript variables and properties available to the Extend Script extension module that called the external binary library.
- Retrieve or discard the results of the script evaluation.

For example, the following example function `foo` uses the `JSAEval` function to call the `getOuterHTML` method of the markup object that represents the `<mytag>` custom element in the current document's markup tree, presumably because a method of that object called the `foo` function.an external binary library provides.

```
static void foo(int argc, JSValue *argv, JSValue returnValue)
{
     char* pTheSource;

     JSAEval("document.element.getSubElement("mytag").getOuterHTML()",
           returnValue, -1);
     pTheSource = JSAValueToString(returnValue);
     // sets the foo function's return value to undefined
     JSAUndefinedToValue(returnValue);
}
```

When an **ExtendScript** extension calls the external `foo` function, the SDK supplies the `JSValue` pointer passed as the argument to the `returnValue` parameter. To retrieve the results of script evaluation, the body of the `foo` function passes this pointer as the `returnValue` argument to the `JSAEval` function. To discard the results of script execution, pass `NULL` as the `returnValue` argument to the `JSAEval` function.

```
JSAEval("document.element.getOuterHTML()", returnValue, -1);
```

When the `JSAEval` function completes, the `returnValue` pointer holds the result of the script evaluation (unless you passed `NULL` as this argument.) To return the script evaluation result as the result of the `foo` function call, the `foo` method need not do anything further—the SDK uses the `returnValue` pointer as it is.

To operate on the data passed as the script evaluation's result, the `foo` function must extract it from the `returnValue` pointer in a format that suits your development goals. To do so, pass the `returnValue` pointer to the `JSAValueTo`*`Xx`* method that returns your preferred data type. In the current example, the result of the evaluation is a character string that is the HTML representation of the entire document. To extract this character string from the returnValue pointer, the body of the `foo` function passes the `returnValue` pointer to the `JSAValueToString` function.

```
char* pTheSource;
JSAEval("document.element.getOuterHTML()", returnValue, -1);
pTheSource = JSAValueToString(returnValue);
```

If the result of scriptlet evaluation is not to be the `foo` function's return value, the `foo` function must clear this result from the *returnValue* pointer before exiting. To set the `foo` function's result, pass the *returnValue* pointer to the `JSA`*`Xx`*`ToValue` function that implements your return value's data type as a `JSValue` pointer. In the current example, the `foo` function passes the *returnValue* pointer to the `JSAUndefinedToValue` function, which causes the `foo` function to return an undefined result; to return integer, Boolean, string, or double values, call the appropriate `JSA`*`Xx`*`ToValue` function.

```
JSAUndefinedToValue(returnValue);
```

JSAPoll

`JSAPoll()`

Polls all scriptlets scheduled for timed execution and executes those that are ready for execution. An external C function that carries out a lengthy procedure must call this function regularly to prevent timed scriptlets from timing out.

Parameters

None.

Returns

`Void`.

JSASetError

`JSASetError(text)`

Generate a Javascript runtime error with the specified text as explanation.

Parameters

text	*String*	The text of the error message this function generates.

E Additional Topics

This Appendix provides details on compatibility with previous versions of GoLive., and utilizing the rules of scope to share JavaScript control names.

Compatibility Information

This SDK requires the use of Adobe GoLive version 5.0 or later. Some features of the SDK are available only in GoLive 6.0.

This version of GoLive supports version 1.4 of the JavaScript language. By default, GoLive uses the Netscape flavor of JavaScript 1.4; for debugging purposes, you can specify temporary use of the Microsoft® JScript flavor or the ECMA-262 flavor. For details, see the description of the `flavor` property in "$ Object Properties" on page 259.

GoLive 6.0 provides full support for Unicode strings. The GoLive 5.0 JavaScript interpreter does not. GoLive 5.0 implements JavaScript strings as 8-bit ASCII

In GoLive 5.0, the collection object in the `controls` global variable provides a numeric index to its elements. In GoLive 6.0, this object does not provide numeric indexes or a `length` property, nor do the `controls` properties of dialog, palette, and inspector objects.

All Collection objects in GoLive 6.0 implement name-based access; to maintain compatibility with all versions of GoLive, do not use numeric indexes to `the controls collection`; instead, retrieve elements by name. For example, the following lines retrieve the `jwpRadio1` control from the `ControlCollection` object that the `controls` global variable provides.

```
var myCtl = controls["jwpRadio1"] // versions 5 and 6 of GoLive
var myOtherCtl = controls[0] // GoLive 5 only
```

If you plan to create an external C or C++ library that your **Extend Script** extension can call, you'll need Microsoft Visual C++ 6.0 to create a dynamically-linked library (DLL) for use on Windows® platforms, and Metrowerks' CodeWarrior 6 Pro to create shared libraries for use on Mac OS platforms.

Case Sensitivity

The HTML specification defines attribute and element names as case-insensitive; that is, the following HTML statements are equivalent:

```
<IMG ALT="graphics.tif">
<img alt="graphics.tif">
<Img Alt="graphics.tif">
```

In contrast, access to JavaScript property names is case-sensitive; that is, the following statements are not equivalent.

```
img.alt = "graphics.tif";
img.ALT = "graphics.tif";
```

GoLive 5 worked around this problem by doing a case-insensitive comparison of such identifiers, effectively making such statements equivalent.

In accordance with the JavaScript standard, GoLive 6 always evaluates JavaScript property names case-sensitively. For example, GoLive 6 does not consider `img.alt` and `img.ALT` to be equivalent.

JavaScript Scope of Name Attribute Values

A **module** is the object that a `main.html` file defines. In general, `name` attributes are defined within the scope of the module that defines them, except for the special cases this section describes.

FIGURE E.1 JavaScript NameSpaces in GoLive environment

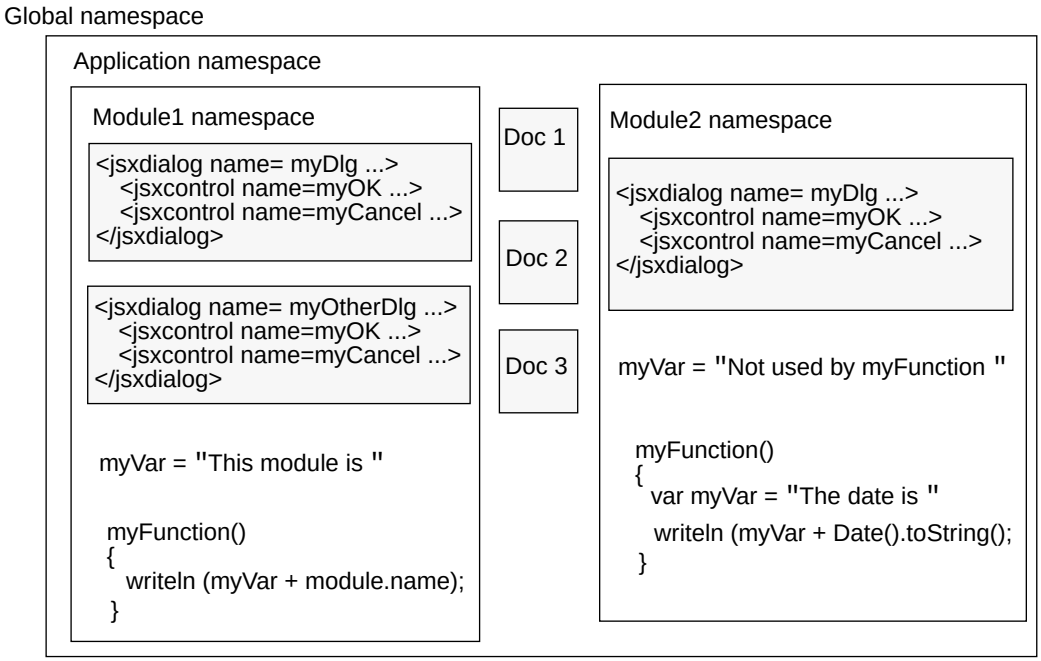


Figure E.1 on page 396 attempts to depict graphically how the rules of scope work to limit name clashes between extensions:

- In general, any object or function an extension module creates is visible only to that extension module, so its name attribute need only be unique within its module's namespace. Thus, both modules can define a myFunction function and a myVar variable, and a couple of the exact same <jsxdialog> elements without concern about accidentally calling each other's functions, using each others' variables, and so on.
- When the object is one that the SDK adds to one of its global collections, such as the menus [] array, a name clash can interfere with name-based retrieval of the object from the global collection. However, it causes no other difficulties.
- Control names can be reused by different dialogs, palettes, or inspectors within the same extension. In Figure E.1, both <jsxdialog> elements in module 1 can use the same <jsxcontrol> element names. Note, however, that controls having non-unique names cannot be retrieved by name reliably from collections such as the `controls` global collection.
- When the name attribute of a <jsxmenu> or <jsxpalettegroup> tag is non-unique, the SDK appends the non-unique entries to the current instance of the element, if it exists; for example, when a <jsxmenu> element's name attribute matches one the SDK has already loaded, the SDK appends the new menu's items to the menu that already uses the same name attribute.

Global scope

Items that are global in scope are available to all modules via properties in the global JavaScript namespace. Such items include

- Data in the Common object
- Data in the Prefs object
- Objects in SDK-provided global collections such as `menus[]`, `dialogs[]`, `controls[]`, and so on.

Application scope

Items of application-wide scope are available to the application object; for example, extension modules themselves are application-wide in scope. In many cases, the GoLive application makes such items available to all modules, so the only practical difference between application scope and global scope is in the way they are accessed. You must use properties or methods of

the global app object to obtain application-scope items; in contrast, a global item's JavaScript name appears directly in the global JavaScript namespace.

Items of application-wide scope include the following:

- Menu names
  `<jsxmenu name="xxxx">`

  When a menu's name attribute is not unique among the name attributes defined by previously-loaded menus, the non-unique menu's items are appended to the previously-loaded menu having the same JavaScript name.
- Object palette tab names
  `<jsxpalettegroup name="xxxx">`

  A non-unique name attribute causes this palette's items to be added to the already-defined palette. Although palettegroup names are shared application-wide, data belonging to an individual palette entry is local to that palette entry.
- Tag names
  `<jsxelement tagname="xxxx">`

  The value of your custom element's tagname attribute must be unique among all `<jsxelement>` elements that all currently-running extensions define.
- `classid` attributes
  `<jsxelement ... classid="`*uniqueIdentifier*`">`
  `<jsxinspector ... classid="`*uniqueIdentifier*`">`
  `<jsxpaletteentry ... classid="`*uniqueIdentifier*`">`

  An individual `classid` value must be unique among all currently-running extensions, and must be used by one extension only.
- Data in the GlobalPrefs object
- Documents: HTML (Web pages), Non-HTML (site documents, text documents)

Module scope

An item of module-wide scope is visible only to the module that defines it; as a result, different modules can reuse the same name attribute values for such items. This feature simplifies the creation of dialogs, palettes, and inspectors because their JavaScript names need only be unique within the extension that defines them; for example, no JavaScript naming conflict results when two different modules can define a `<jsxdialog name=myDlg ...>` element.

Items of module-wide scope include the following:

- Module names
  `<jsxmodule name="xxxx">`

  The value of your module's name attribute must be unique among all name attributes defined by all currently-running extensions.

- Dialogs names
  `<jsxdialog name="xxxx">`

  The value of your dialog's `name` attribute must be unique among all `name` attributes defined by the extension that defines this dialog. Other extensions can reuse this `name` attribute value without conflict.

- Inspector names
  `<jsxinspector name="xxxx">`

  The value of your inspector's `name` attribute must be unique among all `name` attributes defined by the extension that defines this palette. Other extensions can reuse this `name` attribute value without conflict.

- Palette names
  `<jsxpalette name="xxxx">`

  The value of your palette's `name` attribute must be unique among all `name` attributes defined by the extension that defines this palette. Other extensions can reuse this `name` attribute value without conflict.

- All other JavaScript `name` attribute values except those which name controls

  Controls are local to their containing dialog, palette or inspector. Although different dialogs, palettes, and inspectors in the same extension can reuse control names, the usefulness of such architecture is somewhat limited by the fact that the `controlSignal` method typically uses the passed control's `name` property to identify it.

- Boxes

  GoLive creates box objects on demand and names them dynamically. These names are local to the module that creates them, but you never set a box object's name yourself or retrieve box objects by name.

- Pictures

  The name attribute of an `<img>` element or that of a picture created by the `createPicture` method is local to the module that creates the picture.

- Global variables and functions defined by this extension module.

  When an extension declares a variable outside the body of a function, the variable appears as global to the module that defines it; however, it is not visible outside this module. Functions follow similar rules of scope: a function appears as global only within the module that defines it.

  An extension cannot define a true global variable—one that is defined in all extensions. The only truly global variables are those which the SDK provides, such as the `app` global. An extension module makes data available globally by storing it in the Common object or by using the broadcast mechanism to pass string data to other extensions.

Container-wide scope

The JavaScript names of controls and palette entries are local to their containing entity; in the case of a control, the containing entity is the dialog, palette, or inspector window that contains

it. In the case of a palette entry, the containing entity is the palette group (if provided) or palette in which the entry is installed.

- Controls

  Control names must be unique within their containing dialog, palette, or inspector; that is, different dialogs can reuse control names without causing naming conflicts.

- Palette Entries

  A paletteentry element’s name must be unique within the palette or palettegroup that contains it. A paletteentry element’s content is always local to the paletteentry element that provides it.

Local scope

Variables declared as `var` in the body of a function are available only within that function. All other variables and functions the extension defines are available in the module's namespace.

F Additional Resources

This book presents reference descriptions of the Extend Script SDK API only. It doesn't describe the JavaScript language, how to use the SDK, or how to use the GoLive application. You can find this information in any number of well-written, widely-available books, including the following:

- For descriptions of how to use the tags, objects, and functions that the Extend Script SDK provides, see the *Extend Script SDK Programmer's Guide*.
- For documentation of the JavaScript language or descriptions of how to use it, see any of numerous works on this subject, including the following:

  JavaScript: The Definitive Guide; Flanagan, D.; O'Reilly 2001; ISBN 0-59600-048-0

  JavaScript Programmer's Reference; Wootton, C.; Wrox 2001; ISBN 1-861004-59-1

  Pure JavaScript; Wyke, Gilliam, Ting, Michaels; Sams 2001; ISBN 0-672-32141-6

  JavaScript Bible; Goodman, D.; IDG 1998; ISBN 0-7645-3188-3
- For information on using the GoLive application to design web pages and web sites:
  - See the *Adobe GoLive User Guide* for complete documentation of product features.
  - See the following third-party books, which provide detailed discussions of advanced GoLive features, such as WebDAV and Dynamic Link.

    Real World GoLive; Carlson, Fleishman; Peachpit 2001; ISBN 0-201-70408-4

    GoLive 5 Bible; Shadovitz, D.; IDG 2001; ISBN 0-7645-3347-9

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y