# Freeway JavaScript Reference

**Edition 1**

# Freeway JavaScript Reference

## Using this document

If you are familiar with writing Actions for Freeway 3 then you may find these software engineer's notes useful. If you are new to writing Actions for Freeway and would like to attempt to write your own Actions, then work through the 'Writing Freeway Actions' PDF (Writing FW Actions.pdf) supplied on the Freeway CD (and copied to your Freeway folder on Freeway installation) first. As and when this document is updated, downloadable copies will be made available via the SoftPress Web site.

## Specific Properties and User Methods

### Global Properties and Methods

These properties and user methods are global and are not attached to a particular object.

**Properties (3)**

**`fwFreewayVersion (read only)`**
This returns a string with the version of Freeway that is running.

**`fwRegion (read only)`**
This returns the region code to which Freeway is currently localized. In a dual-language build such as Freeway 3J, which has an interface that can be switched between English and Japanese, it will return the region of the current interface. For a full list of regions see the support document "Localising Freeway Actions" on the Actions web site.

Examples:

```
if (fwRegion==0)
      alert("Freeway is localised for the United States.");
else
      alert("Freeway is NOT localised for the United States.");
or
if (fwRegion==14)
      alert("Freeway is localised for Japan.");
else
      alert("Freeway is NOT localised for Japan.");
```

**fwTimeout** (read/write)
This is the interval in 60ths of a second that can elapse before the JavaScript interpreter automatically terminates a JavaScript interface method in your Action. You should avoid writing code which takes so long to execute that it will time out. So this value is unlikely to need changing, the default is 5 seconds (5 × 60). Effectively, you should not need to alter this default – fix your code instead!

*Example:*

```
// set the timeout to 10 seconds
fwTimeout=10*60;
```

# Methods

**alert(string[,string])**
This will cause Freeway to put a dialog box on your screen displaying the text passed in. The dialog *One or more strings* has an **OK** and an **Abort** button. Clicking on the **Abort** button will cause execution of the JavaScript to abort.

The internal timer, which times out interface methods when they exceed the limit set in the global property `fwTimeout`, is paused during the execution of this method.

> Note: This method differs from the alert method that is present within the JavaScript of most browsers in that it accepts multiple arguments separated by commas.

*Examples*

**`alert("Hello")`**
causes an **alert** dialog box with the text “Hello" to appear.


**`alert(1+2);`**
`(non-string or numerical parameter)` causes an **alert** dialog box with the text "3" to appear.

**`alert("My name is ","Gustav.");`**
causes an **`alert`** dialog with the text `"My name is Gustav."` to appear (concatenating the strings).

**`confirm(string[,string])`**
This will cause Freeway to put a dialog box up on your screen displaying the text that is passed one or more strings in the parameters. The dialog has an **OK** button, a **Cancel** button and an **Abort** button.

The method returns true if the user clicks on the **OK** button but if the user clicks **Cancel** it returns false. Clicking on the **Abort** button will cause execution of the JavaScript to abort. If you are building a site then the build process will stop in the same way as if you had clicked on the Stop button in the **Progress** dialog when publishing.

The internal timer, which times out interface methods that exceed the limit set in the global property `fwTimeout` is paused during the execution of this method.

> Note: This method differs from the `confirm` method that is present within the JavaScript of most browsers in that it accepts multiple arguments separated by commas.

*Examples:*

**confirm("It is a nice day?");**
Causes a **confirm** dialog with the text "It is a nice day?" to appear:


**confirm("Your name is ","Gustav?");**
Causes a **confirm** dialog with the text "Your name is Gustav?" to appear.

**fwAbort(string[,string])**
This method will cause the publishing process to be aborted. It should only be called while you *One or more strings* are publishing and do not want to continue uploading. The strings passed will be concatenated together and non-string parameters converted to strings, and posted as an **alert**. The effect of this is the same as someone manually aborting publishing. It is not possible to abort publishing without the dialog appearing. You might use this in the case of an error occurring.

*Example:*

The following Action will abort publishing, with the message as shown in the dialog box

```
<item-action name="test">
<action-javascript>

   function fwAfterEndBody()
   {
      fwAbort("Your coat is to dark for publishing to continue.");
   }

</action-javascript>
</item-action>
```

**fwEncode(string[,page][,delimiters])**
This method will convert the string into the encoding specified for the page. The string that is returned is in an encoded form, suitable for use in the HTML on the page specified. For example, if the page is set to “ISO8859-1” encoding, the text will be converted to that specific encoding for this page, allowing coding to be consistent with the current page. This means that the text will be correctly encoded if the page is, for example, Japanese. It will also ensure that special characters are correctly translated into the required HTML. If no page is passed it will merely ensure that special characters are correctly translated.

*Arguments:*

`string:` The string to be encoded

**page (optional)**
The encoding for this page will be used to encode the string`delimiters (optional)`the returned string will be delimited by these characters

*Example:*

The following code adds the markup for a text area. It uses fwEncode to produce an encoded form of the value parameter in the output.

```
function addTextArea(name, value)
{
   // this adds the code and encodes the value
   // appropriately
   // <textarea ‘name= name value> </textarea>

   fwDocument.fwWrite("<textarea name = ‘", name, "’>",
      fwEncode(value, fwPage),
      "</textarea>");
}
```

**fwQuote(string[, newQuote[, oldQuote]])**
This method allows you to return a string in its quoted form. Now if you pass in the original quotes it will remove them (if present) and replace them with the new ones you have defined. This function is particularly useful if you are retrieving the values of properties and need them quoted in a different form. For example if you need to change the quotes from double to single.

| | | |
|---|---|---|
| `fwQuote("bat")` | will yield | `"bat"` |
| `fwQuote("bat","'")` | will yield | `'bat'` |
| `fwQuote("'bat'",'"',"'")` | will yield | `"bat"` |

# Files and Freeway Actions

It is possible for Freeway Actions to read and write files and to cause those files to be uploaded when the site is published. So for example you could make a suite of Freeway Actions that generate JavaScript files, text files or even HTML files that are uploaded automatically. Access to the files is largely through the `FWFile` class.

However there are a small number of global methods that augment the `FWFile` class. These methods all apply to the "working folder" of the Freeway document. The working folder is next to your Freeway Document and will contain files that are generated by Actions

## Methods

**fwDeleteFile(name)**
This attempts to delete a file with the specified name from the working folder. If the file of that name was not deleted it returns false otherwise it returns true. Boolean.

**fwFindAllFiles([type[, creator]])**
This returns an array with the names of all the files in the working folder. If a [Macintosh] file type is passed then Freeway will return just those files of that type. If a Macintosh creator type is also passed then Freeway will return just the files of that creator and type. The type and creator are expected to be strings of 4 characters. If there are no files that match that type or indeed there is no working folder then an array with no elements will be returned.

*Arguments:*

`type`*4 charstring* `creator`*String*

**fwFindFile(name)**
This is returned if a file of a given name exists.

*Argument:*

`name:` *string name of file to look for.*

# Classes

## Introduction

Two types of classes an interface with a data property an instance of an object.

*Example:*

**fileread**
The classes are listed in the following pages in alphabetical order for easy reference and their relationship is shown in the layout, organisation of classes and subclasses.

# Organisation of Classes

Class FWAction
Class FWActionList
Class FWColor
Class FWColorList
Class FWDocument
Class FWElement
Class FWLayoutElement
Class FWItem
Class FWPage
Class FWFile
Class FWFolder
Class FWItemList
Class FWLink
Class FWListData
Class FWOSAInterpreter
Class FWOutput
Class FWAttribute
Class FWTag
Class FWParameter
Class FWLinkParameter
Class FWColorParameter
Class FWFileParameterFWImageParameter (* hidden)
Class FWParameterList

**Class FWAction**
This class defines an instance of an action on a page. Every instance (occurrence) of an action on a page has a corresponding FWAction. All the user defined methods are called within the context of a FWAction. i.e. 'this' is the current FWAction.

### Properties (19)

**fwActions**
The list of Actions in which this Action appears, that is the list of Actions applied to the item or page where *FWActionList* this Action is applied.

**fwAltText**
The alt text that is applied to the Action's item. Note you should call fwEncode before generating HTML *String* with it.

**FwClearGif**
Gives the relative path to the clear-gif from the current page (for example, "Resources/_clear.gif"). *String*

**fwDocument**
The Action document (the Freeway document in which the Action is used). *FWDocument*

**fwFileName**
Returns the relative path of the file to which the Action is being applied. The file will contain all or part of the *String* image (if the image is sliced). This is only available after or during `fwAtContent`. This can vary across different calls to `fwAtContent` if an object is sliced.

**fwFolder**
Immediate folder containing the page on which the Action appears.*FWFolder*

**fwGeneratesForm**
Determines whether this Action generates a form (and therefore does not need Freeway to generate a form). *Boolean*

**fwGeneratesLink**
Determines whether this Action generates a link (and therefore Freeway does not need to generate a link).*Boolean*

**fwHeight**
The height of the item, or part (if the image is sliced), on to which the Action is applied. This can vary across *Integer* different calls to `fwAtContent` if an object is sliced.

**fwIndexOf**
The zero-based index of this Action in the action list `fwActions`.*Integer*

**fwItem**
Item to which the Action is applied. *FWElement*

**fwLayerName**
The name of the layer on which the Action is applied. If there is no layer, an empty string is returned. *String*

**fwMapName**
The name of the image map associated with this item. This can vary across different calls to `fwAtContent` if an object is sliced. *String*

**fwMarkups**
The list of `<action-markup>` tags defined for this Action.

*Examples:*

```
<action-markup>
<action-html before-end-head>
<action-markup custom name=fred>
```

These can be named and then easily referenced by name. They are particularly useful to store pieces of raw text to be output by the Action. `<action-html>` is a synonym for `<action-markup>`.

**fwPage**
Page on which the Action appears. *FWPage*

**fwParameters**
The list of parameters which this Action has. *FWParameterList*

**fwRequiresForm**
Determines whether this Action requires a form in order to be generated (for example, if it outputs a text field or *Boolean* button which needs to be contained in a form).

**fwTitle**
Name of the Action. *String*

**fwWidth**
The width of the item or part (if the image is sliced), on to which the Action is applied. This can vary across *Integer* different calls to `fwAtContent` if an object is sliced.

## Methods (1)

**fwSubstitute(string)**
This method will substitute variables of the form `&param;` with their values from the Action parameter list (in the same way that it occurs in the Simple Actions language). Returns a string with the parameters substituted.

*Example 1:*

This example has a string into which the parameters `param1` and `param2` are substituted.

```
<page-action name="Substitute Test 1">
<action-text name="param1" default=a>
<action-text name="param2" default=b>
<action-javascript>

function fwAfterEndHTML()
{
   var s="param1= &param1; param2= &param2;";
   alert(s);
   alert(fwSubstitute(s));
}

</action-javascript>
</page-action>
```

*Example 2:*

You can also use `<action-markup custom>` to dump in sections of text without having to quote it. This is very useful if you want to put pieces of JavaScript in the output. You can access it through JavaScript from the `fwMarkups` property of the action.

```
<page-action name="Substitute Test 2">
<action-text name="param1" default=c>
<action-text name="param2" default=d>
<action-markup custom name="mytext">
param1= &param1; param2= &param2;
</action-raw>
<action-javascript>

function fwAfterEndHTML()
{
   var s=fwMarkups.mytext;
   alert(s);
   alert(fwSubstitute(s));
}
</action-javascript>
</page-action>
```

## User Methods (4 methods)

These methods if present in your Freeway Action will be called in the places that their name suggests. This gives your Action an opportunity to add to the HTML that is generated by Freeway at a specific point in the output.

*Example:*

The following code would insert the word "wibble" just before the end of the header (ie before </Head>) of the published HTML file.

```
fwBeforeEndHead()
fwDocument.fwWrite ("wibble'); is equivalent to
      <action-markup before-end-head> wibble </action-markup >
```

Positions for insertion point which mirror specific points in the output:

```
fwAfterEndBody()                fwBeforeEndBody()
fwAfterEndForm()                fwBeforeEndForm()
fwAfterEndHead()                fwBeforeEndHead()
fwAfterEndHTML()                fwBeforeEndHTML()
fwAfterEndNoFrames()            fwBeforeEndNoFrames()
fwAfterEndTable()               fwBeforeEndTable()
fwAfterStartBody()              fwBeforeStartBody()
fwAfterStartForm()              fwBeforeStartForm()
fwAfterStartHead()              fwBeforeStartHead()
fwAfterStartHTML()              fwBeforeStartHTML()
```

| | |
|---|---|
| fwAfterStartNoFrames() | fwBeforeStartNoFrames() |
| fwAfterStartTable() | fwBeforeStartTable() |

**fwAtContent()**
This method, if present in your Freeway Action, is called when the content of an image would be written. If this method is present then Freeway will not write the <img> tag. Instead it will rely on your Freeway Action to generate the entire content for the item (for example, the <img> tag for an image) in your fwAtContent method.

**fwIsDirty()**
This method, if present in your Freeway Action, is called before your page is published. It gives your Action a chance to force Freeway to republish a page when Freeway does not think that the page is needs to be republished. The fwIsDirty method will only be called if Freeway believes your page does not need to be republished. This is because if your page is dirty then Freeway has no need to call your fwIsDirty method, as it will republish your page anyway. *Returns:* Returns true if the page is "dirty" and should be republished.

*Example 1:*

This example will always force the current page to be republished.

```
function fwIsDirty()
{
   return true;
}
```

*Example 2:*

This example method looks at the "Last Page" parameter and compares it against the next page in the containing folder. If it is different then it marks the page as dirty.

```
function fwIsDirty()
{
   // if the last page generated is not the same as 'current' page
   // then we are dirty

   var nextPage = fwFolder.fwNextPage(fwPage);
   var currPage = fwParameters["Last Page"].fwValue.fwInternal;

   return nextPage != currPage;
}
```

**fwInterface**

This method, if present, is called when the Actions palette is being set up. It allows you to perform any functions required and set necessary values to display in the interface

## Class FWActionList

This defines the list of actions that are applied to an item or page. This list is zero-based i.e. the first item is

**fwActions[0].**

### Property (1)

**fwLength**
Number of items in the list.  An Action item has an empty actions list.
*Integer*

## Class FWAttribute

This class is a subclass of `Class`

**FWOutput**.
It represents an attribute of a `Case`

**FWTag.**
Consider the following snippit of HTML.:

```
<img src="Resources/_clear.gif" border=0 width=50 height=1 alt="">
```

In the tag tree this would be represented as one `FWTag`. The attributes of the tag:

```
src="Resources/_clear.gif"
      border=0
      width=50
      height=1
      alt=""
```

are each represented as an `FWAttribute`.

When you add an attribute to a tag (in JavaScript you do this by just adding a property), such as:

```
myTag.onclick='"alert(1);”';
```

a new `FWAttribute` will be created.

### Property (1)

**`fwValue`**
This is the text value of the attribute.

## Class FWColor

This class represents the colors that appear in Freeway in the **Color** palette.

### Properties (8)

**`fwHasValue`**
Is the color a true color which has a value. For example "knockout" is not a proper color and has no *Boolean* value.

**`fwHex`**
The hex value of the color. *String*

**`fwIsForcedNetSafe`**
Is the color not "naturally" a net-safe color but the user has forced it to be Netsafe. *Boolean*

**`fwIsNetSafe`**
Is the color a net-safe color. *Boolean*

**`fwIsPermanent`**
Is the color a permanent color. *Boolean*

**fwIsTemporary**
Is the color a temporary color. *Boolean*

**fwTitle**
Name of the color. *String*

**fwValue**
This returns a JavaScript object with values for red, green and blue.

## Class FWColorList

This defines the list of colors that are in a document. The items are FwColor

### Property (1)

**fwLength**
Number of items in the list. *String*

## Class FWColorParameter

This class represents the `Color` Parameter. It is a subclass of FWParameter.

The `fwValue` property will return a value of type `Color.`

## Class FWDocument

This class represents a Freeway document. From this class you can access many document-wide properties like colors, pages, master pages etc.

### Properties (7)

**fwColors**
List of colors that are defined for this document.
FWColorList

**fwCurrentPage**
This returns the current page of the front-most window that contains this document.

**FWPage**
**Example:** This posts an alert with the current page in it.

```
<item-action name="Get Current Page">
   <action-button name="Current Page"
      onclick"alert(fwDocument.fwCurrentPage)">
</item-action>
```

**fwMasterPages**
List of master pages that appear in this document. *FWItemList*

**fwPages**
List of pages that appear in this document. *FWItemList*

**fwTags**
This is the tags that have been created during the generation of the HTML of the current page. If you *FWTag*access this property outside of the generation of the page then it will be null. If you access this propertybefore fwAfterEndHTML then it will be incomplete.

**fwTargets**
An array of targets that have been defined for this document. *Array of strings*

**fwTitle**
Name of the document. *String*

# Methods (4)

**fwSelection()**
This returns the current selection in an array. This is the selection of the front-most window of this*Array of FWLayoutElemen t*document. If there is no selection or if there is a text selection or a frameset is highlighted it will return an array with no elements.

*Example:*

The following example lists all the selected elements in an alert.

```
<item-action name="Get Selection">
<action-button name="Get Properties" onclick="GetSelection()">
<action-javascript>

function GetSelection()
{
   var selection = fwDocument.fwSelection();
   var temp = "";

   for (var i in selection)
      temp += selection[i] + "\r";
   alert(temp);
}

</action-javascript>
</item-action>
```

**fwWrite(string[,string...])**
This will write the *Arguments:*

as a concatenated string into the output at the current*One or more strings*position in the HTML file. So if you call this from your `fwAfterEndHead` method, the *Arguments:*

will be spooled into the HTML file at this point.

**fwWriteln([string[,string...]])**
This will write the *Arguments:*

as a concatenated string into the output at the *Zero or more strings*current position in the HTML file. Followed by a `<cr>` of the format that is specified in the Freeway Preferences.

**fwWritelnopt([string[,string...]])**
As `fwWriteln` with the exception that a `<cr>` is only written if the*Zero or more strings* HTML output has been set to be more readable in the preferences.

## Class FWElement

This is the base class of Freeway items. It includes all items that can appear on a page, the page itself, folders and Master pages. It is also used to represent the internal structures that manage and organize styling information within Freeway. There are no objects derived from this only

subclasses. The primary subclass is `Class FWLayoutElement` with two further subclasses in it. These are `Class FWItem` and `Class FWPage`. (You will find these described in alphabetical sequence but the structure is defined in the Classes map.)

[There is a subclass of this class: `Class FWLayoutElement` with two further subclasses `Class FWItem` and `Class FWPage`.]

## Properties (17)

**`fwHasTitle`**
Can the item have a title (name) For example table cells cannot have a name. *Boolean*

**`fwIsFolder`**
Is the element a folder. *Boolean*

**`fwIsFrameSet`**
Is the item a FrameSet page. *Boolean*

**`fwIsInlined`**
Is the element an in-lined item, that is the item inserted in the text inside another item. *Boolean*

**`fwIsItem`**
Is the element an item that appears on a page. *Boolean*

**`fwIsMapArea`**
Is the element a map area. *Boolean*

**`fwIsOval`**
Is the element an oval item. *Boolean*

**`fwIsPage`**
Is the element a page. *Boolean*

**`fwIsPath`**
Is the element a path. *Boolean*

**`fwIsRectangle`**
Is the element a rectangular item. *Boolean*

**fwIsRRectangle**
Is the element a round cornered rectangle. *Boolean*

**fwIsTable**
Is the element a table. *Boolean*

**fwIsTableCell**
Is the element a table cell. *Boolean*

**fwItems**
List of elements that are contained within the element. e.g. for a page this would be the list of items that *FWItemList* appear on a page.

**fwMaster**
The master element on which this element is based. If the element is derived from a master page then this *FWElement* will be the element on the master page. If the element has no master then this will be nil.

**fwParent**
The element that encloses or owns this item. For an item sketched on the page this would be a page. For *FWElement* a page this would be the folder in which it is stored.

**fwTitle**
Name of the element.*String*

# Method (1)

**fwHttpPath([page])**
This returns the relative path to the page on which an item appears. If you pass a page reference into the *Boolean*method then it will return the path relative to that page. *Argument:* Page - the page for which a relative path is returned.

## Class FWFile

Using JavaScript it is possible to read from and write to external files. There are some restrictions on the files that an Action can create - this is to prevent the possibility of malicious actions overwriting other actions, destroying data etc. If you reference a file by name it will reference a file in the document's working folder. The working folder is a folder next to your

Freeway Document and will contain files that are generated by Actions.

This class represents a file. You can create `FWFile` objects within JavaScript and use them to open, read and write files. The general format of the methods is that they return true if the operation succeeds or false if it fails. When you need to discover if an error prevented the operation from succeeding you can interrogate the `fwError` property of the object.

**Example:** This code fragment prompts the user to locate a file, then opens the file and posts the contents of it into an alert.

```
myFile = new FWFile;

// choose a file of type text with the prompt 'Locate File'
// and open it

if (myFile.fwLocateRead('Locate File','TEXT'))
{
   alert(myFile.fwReadString(myFile.fwAvailable));
}
```

When you write files in Freeway you can either get the user to choose the location where the file is to be written or you can specify the file programatically within Freeway. When you specify a file programatically you pass a filename and Freeway will create the file in a folder that is next to your Freeway document. If your Freeway document is called "home site" then Freeway will create a working folder called "home site." and place the document in there.

## Properties (3)

**`fwAvailable`**
Is the number of bytes left to be read. *Integer*

*Example:*
The following example shows how you can use a filename in a parameter to specify the file to be read. This sample action includes mark-up from external files, opens them and dumps the content using the `fwDocument.fwWrite()` method.

```
<page-action name="External Markup">
<action-file name="before-start-html">
<action-file name="after-start-html">
<action-file name="before-start-head">
<action-file name="after-start-head">
<action-file name="before-end-head">
<action-file name="after-end-head">
<action-file name="before-start-body">
<action-file name="after-start-body">
<action-file name="before-end-body">
<action-file name="after-end-body">
<action-file name="before-end-html">
<action-file name="after-end-html">
<value type="TEXT">
</action-file>
<action-javascript>

   function writeFile(fileParam)
   {
      if (fileParam.fwHasFile)
      {
         var file = new FWFile();
         file.fwOpenRead(fileParam);
         var s = file.fwReadString(file.fwAvailable);
         fwDocument.fwWrite(s);
      }
   }

   function fwBeforeStartHTML()
   {
      writeFile(fwParameters["before-start-html"])
   }

   function fwAfterStartHTML()
   {
      writeFile(fwParameters["after-start-html"])
   }

   function fwBeforeStartHead()
   {
      writeFile(fwParameters["before-start-head"])
   }
```

```
    function fwAfterStartHead()
    {
       writeFile(fwParameters["after-start-head"])
    }

    function fwBeforeEndHead()
    {
       writeFile(fwParameters["before-end-head"])
    }

    function fwBeforeEndHead()
    {
       writeFile(fwParameters["before-end-head"])
    }

    function fwBeforeStartBody()
    {
       writeFile(fwParameters["before-start-body"])
    }

    function fwBeforeEndBody()
    {
       writeFile(fwParameters["before-end-body"])
    }

    function fwBeforeEndBody()
    {
       writeFile(fwParameters["before-end-body"])
    }

    function fwBeforeEndHTML()
    {
       writeFile(fwParameters["before-end-html"])
    }

    function fwAfterEndHTML()
    {
       writeFile(fwParameters["after-end-html"])
    }

</action-javascript>
</page-action>
```

**fwCanRead**

Is the file open for reading. *Boolean*

**fwCanWrite**
Is the file open for writing. *Boolean*

## Methods (10)

**fwClose()**
This closes an open file. You should close a file when you have finished writing to it so the write buffers will be flushed and the file can be opened by another application or can be uploaded. You close files opened for reading when you have finished with them.

**fwDelete()**
This deletes the current file. It will only delete the file if it is one that has been specified by the user using `fwLocateWrite` or if it is in the working folder.

**fwLocateRead([prompt string][,FileType1, ... FileType2])**
This will allow the user to *String* choose a file. You can then open this file and read it. `FileType1, 2` etc- list of file types. Each file type is a 4 character string. If you pass less than 4 characters it will be padded with spaces. If you pass more than it will be truncated at 4. If an array is passed then it will be interpreted as a list of 4 character strings.

*Arguments:*

`prompt string` - string that will appear in the open dialog box to prompt the user.

*Example:1*

Choose a file of type 'TEXT'.

```
var wasChosen = myFile.fwLocateRead('Locate a text File','TEXT');
```

*Example 2:*

Choose a file of type "GIFf" or "JPEG".

```
var wasChosen = myFile.fwLocateRead('Locate a graphic
File','GIFf','JPEG');
```

*Example 3:*

Choose a file of type "GIFf" or "JPEG".

```
var fileTypes = Array('GIFf','JPEG');

var wasChosen = myFile.fwLocateRead('Locate a graphic
File',fileTypes );
```

*Example 4:*

Choose any file.

```
var wasChosen = myFile.fwLocateRead('Locate a File');
```

**fwLocateWrite([prompt string[,defaultfilename[, type[,creator]]])**
This is used to see if *Boolean* Freeway created the file successfully. Returns true if Freeway created the file. i.e. if the user did not cancel and if the file was created successfully.

*Arguments:*

`prompt string` the string that will appear in the dialog as a prompt

**defaultfilename**
the name that will appear in the dialog box as the default name of the file

**type**
the type of the file to be created e.g. 'Text' If omitted it will be given the file type of "????". Each type is a string 4 characters long.

**creator**
The creator of the file, a character string of 4 characters. If not present it will be given the creator type of Freeway.

*Example:*

This allows the user to create a file, selecting the name that they want and putting the text "test" into it.

```
<item-action name="Test">
<action-button name="locate" onclick = "DoLocate()"/>
<action-javascript>

   function DoLocate()
   {
      // create a new file object
      myFile = new FWFile;

      // get the user to create the file
      var wasChosen = myFile.fwLocateWrite('Locate a NEW File',"My
File",'TEXT');
      if (wasChosen)
      {
         myFile.fwWrite('test');
         myFile.fwClose();
      }
   }

</action-javascript>
</item-action>
```

**fwOpenRead(fileobject/fileparameter/tempfilename)**
This will open a file that is specified for reading. If the file cannot be opened it returns false. The file to be read is either the name of a file in the working folder, a fileparameter or a FWFile object (fileobject/fileparameter/tempfilename). *Boolean*

**fwOpenWrite(filename[,deleteexisting][type[,creator]]] )**
This will create a file of the given name or open an existing file of this name for writing.

*Arguments:*

`filename:` name of the file to be created

`deleteexisting:` (boolean) delete the existing file (if any).

`type4` *chars* the file type

`creator4` *chars* the creator string

*Example:*

This creates a file "Test File", writes "test" into it and then closes it.

```
<item-action name="Test">
<action-button name="Make File" onclick = "doit()"/>
<action-javascript>

   function doit()
   {
      myFile = new FwFile;

      myFile.fwOpenWrite('Test File',false,'TEXT');
      myFile.fwWrite('test');
      myFile.fwClose();
   }

</action-javascript>
</item-action>
```

**fwReadln()**

This will read from the current read position to the end of the line, unless the end of the file is reached in *String* which case it will read to the end of the file. All the characters with the exception of the line end character /r or /n will be returned. If a line is empty (i.e. only contains a line-end character) then an empty string will be returned.

*Example:* This locates a text file and will post the contents one line at a time in an alert.

```
<item-action name="Test fwReadln">
<action-button name="read file" onclick = "DoLocate()"/>
<action-javascript>

   function DoLocate()
   {
      // create a new file object

      myFile = new FWFile;

      // choose a file of type text with the prompt 'Locate File'
      // and open it

      if (myFile.fwLocateRead('Locate File','TEXT'))
      {
         do
         {
           var s = myFile.fwReadln();
           if (s)
             alert(s);
           else
             break;
         } while (true);
         myFile.fwClose();
      }
   }

</action-javascript>
</item-action>
```

**fwReadString([numberofbytes])**
This reads the number of bytes specified and returns them as a string. If you omit the*String* number of bytes then it will attempt to read the whole file.

**fwWrite(string[,string...])**
This will write the Arguments as a concatenated string into the file. *One or more Strings*

**fwWriteln(string[,string...])**
This will write the arguments as a concatenated string into the file, followed by a *One or more strings* carriage return.

## Class FWFileParameter

This class provides an interface to the file parameter. It is a subclass of `FWParameter`. The File parameter provides a way that the user can specify a reference to a file. If a file parameter is referenced from within Freeway in such a way that it's file name is resolved (i.e. if you ask for the location of a file within the web site) then the file will be uploaded when the site is published.

The location of a file is resolved as a "normal" part of writing an action. In a non-JavaScript action you would reference the location of the file (relative to the page) by referencing the parameter name so that it is output during publishing. In a JavaScript action this will happen if you call the methods `fwFullFileName()`, `fwFileName()`, `toString()` - or cause the `toString()` method to be called implicitly.

### Properties (3)

**`fwFileName`**
The short name of the file (without the path) as published. *String*

**`fwFullFileName`**
The name of the file (with path) as published. *String*

**`fwHasFile`**
Does the parameter have a file set. *Boolean*

# Method (1)

**`fwSpecify(file)`**
This allows you to specify the file that will be used where file is a `FWFile`.

## Class FWFolder

This class represents the folders that you can create for pages within Freeway that appear in the Site palette.

### Properties (3)

**`fwFileName`**
This is the file name of the folder. *String*

**fwFirstPage**
This is the first page in the folder. It is null if the folder has no pages in it.
*FWPage*

**fwLastPage**
This is the last page in the folder. It is null if the folder has no pages in it.
*FWPage*

## Methods (2)

**fwNextPage(page)**
Returns the next page in the folder that is after the 'page' passed in as an argument.*FWPage*

**Example:** The page after the current page is returned but if the current page is the final page then this method returns null.

```
var nextPage = fwFolder.fwNextPage(fwPage);
```

**fwPreviousPage(page)**
Returns the page in the folder that is before the page passed in as an argument.*FWPage*

**Example:** This will return the page before the current page. If there is no page before the page passed in then this method returns null.

```
var prevPage = fwFolder.fwPreviousPage(fwPage);
```

## Class FWImageParameter

This is a subclass of `FWFileParameter`. Parameters of this type are generated by `<action-image>` in the Freeway Action. These have the same properties and methods as `FWFileParameter`.

## Class FWItem

This class represents items that appear on the Freeway page. It is a subclass of `FWLayoutElement`.

### Properties (17)

**fwAltText**
This returns the alt text for the item. If "Auto" is set then it will return the

automatic alt text (the alt text *String* that is calculated by Freeway) if this item is the bottom-most item in an image group. If there is alt text specifically set for this item then it will return that alt text. If there is no alt text set then it will return an empty string.

*Example:*

This posts an alert using the alt text of the item onto which the Action is applied.

```
<item-action name="Alt Text">
<action-javascript>

   function fwAfterEndBody()
   {
      alert("Alt Text = '",fwItem.fwAltText,"'");
   }

</action-javascript>
</item-action>
```

**fwBottom**
This is the bottom (y) co-ordinate of the item.*Integer*

**fwExportType**
This is the export-type of the item. It has the following values.

*Integer*

0. Other File Plugins, applets, pass-through graphics
1. HTML Item
2. Export as GIF file(s)
3. Export as JPEG file(s)
4. Export as URL item
5. Export as PNG file(s)
6. Map Area*the item is an image map*

**fwHasAutoAltText**
Returns true if “Auto Alt Text” has been selected for this item. *Boolean*

**fwHeight**
This is the height of the item. *Integer*

**fwIsInLayer**
Is the item in a layer – for inline items or table cells this will be true as their parent is in a layer. *Boolean*

**fwIsLayer**
Is the item a layer - for inline items or table cells this will be false as they are in a layer but are not layers *Boolean* themselves.

**fwIsOnPasteboard**
Is this item entirely on the pasteboard. *Boolean*

**fwIsOverflowed**
Does this item have overflowed content.*Boolean*

*Example:*

An alert is posted when the button is pressed, if the item to which the action is applied has overflowed content.

```
<item-action name="Is Overflowed">
    <action-button name="Is Overflowed"
    onclick="alert(fwItem.fwIsOverflowed)">
</item-action>
```

**fwIsPublished**
Is the publish attribute of the item set? This will only be false if the user sets the publish checkbox of the item so that it will not be published. *Boolean*

**fwLeft**
This is the left (x) co-ordinate of the item. *Integer*

**fwLink**
This is the link object that is applied to an item. If there is no link then this will return null otherwise it will return an `FWLinkObject.`

**fwMainItem**
This returns the main item in a combined image. This item is the item to which the active Action is `FWItem` applied. In the case of a combined image it returns the bottom-most item in that combined image.

*Example:* This posts an alert with the name of the main item.

```
<item-action name="Get Main Item">
<action-button name="Main Item" onclick"alert(fwItem.fwMainItem)">
</item-action>
```

**fwRight**
This is the right (x) co-ordinate of the item. *Integer*

**fwSubmit**
This is the name that has been entered as the submit name if the item is an image submit button. If it is *String* not then this property will be null.

**fwTop**
This is the top (y) co-ordinate of the item. *Integer*

**fwWidth**
This is the width of the item. *Integer*

## Methods (9)

**fwCountLines([includeoverflow])**
This returns number of lines in the item, the overflow is included if you pass a *Integer*Boolean value of true to the overflow.

**Example:** This returns the number of lines in the item. A checkbox allows you to consider the lines in the overflow.

```
<item-action name="Count Lines">
<action-checkbox name=overflow>

    <action-button name="Count"
     onclick="alert(fwItem.fwCountLines(fwParameters.overflow ==
    'yes'))">

</item-action>
```

**fwFindLines([firstline][lastline][includeoverflow])**
This returns an array of strings that *An array of zero* or satisfy the criteria. The `firstline` & `lastline` are optional and work in the same way as more text strings to the string slice operator. So index elements are positive numbers from the start of the content, negative numbers indicate offsets from the end of the offset so that `fwFindLines(1,-1)` will return an array of all the text except for the first and last lines. The default behaviour is to

exclude the overflow. If you wish to include the overflow pass in a Boolean value of true.

*Arguments:*

`firstline:` first line to get the text from `lastline`last line to get the text from `includeoverflow`include overflowed text

*Example:*

This Action will list the lines of text as specified by the start & end parameters.

```
<item-action name="Find All Lines">
<action-number name=start default=0>
<action-number name=end default=0>
<action-checkbox name=overflow>
<action-button name="Find All Lines" onclick="FindAllText()">
<action-javascript>

function FindAllText()
{
   var overflow = fwParameters.overflow == "yes";
   var start = parseInt(fwParameters.start);
   var end = parseInt(fwParameters.end);
   var text = fwItem.fwFindAllLines(start, end, overflow);
   var temp = "";

   for (var i in text)
      temp += text[i];
   alert(temp);
}

</action-javascript>
</item-action>
```

**`fwFlash`**

This returns an object with the properties that have been set for Flash export. The object has the properties: `fwPlay` and `fwLoop.`

**`fwImageGroup`**

This returns an array of all the items that are in the same image group as this item. Remember – graphic *An array of zero* or items that overlap each other and do not have their combine flag turned off will be grouped with other*more text strings* items to form a single image. This function will return all these items in an array. *r more items*

*Example:*

This posts an alert with the names of all the items in the current image group.

```
<item-action name="Get Group">
<action-button name="Get Group" onclick="GetGroup()">
<action-javascript>

function GetGroup()
{
    var imageGroup = fwItem.fwImageGroup();
    var temp = "";

    for (var i in imageGroup)
        temp += imageGroup[i] + "\r";
    alert(temp);
}

</action-javascript>
</item-action>
```

**fwPath()**

This returns the set of points that describe the item as a closed path. The points are appear in the array as x,y co-ordinates. So a rectangle with a top-left of 5,10 and a height of 20 and a width of 30 will appear as the points: 5,10,35,10,35,30,5,30

*Example:*

This example gets the path of the item that the action is applied to and puts it in an alert. The alert method will automatically convert the path to a comma separated string.

```
<item-action name=showpath>
<action-javascript>
function fwAfterEndBody()
{
   alert(fwItem.fwPath());
}
</action-javascript>
</item-action>
```

**fwQuickTime**
This returns an object with the properties that have been set for QuickTime export. The object has the following properties: `fwAutoPlay`, `fwLoop`, `fwController`

**fwSelect**
This returns an object with the properties `fwSize` and `fwWidth` that defines the size and width those dimensions would have as a selection area (popup or list form item).

**fwTextarea**
This returns the dimensions of an object's text area [of what type with what parameters]with the properties `fwRows` and `fwCols` that define the rows and an item

**fwTextfield**
This returns an object with the property `fwSize` that defines the size of the item with those dimensions as a text field.

## Class FWItemList

This defines a list of `FWElements`. It appears as the `fwItems` property of the `FWElement.` It is a zero indexed list that you can also "associatively" access. For a page it will be the list of items on the page. For a folder, it will be the list of pages. For an item it will be the list of inline items within it.

*Example: 1:*
Set myItem to the item named "anitem"

```
var myItem = fwPage.fwItems["anitem"];
```

or

```
var myItem = fwPage.fwItems.anitem;
```

*Example: 2:*
Set myItem to the first item on the page

```
var myItem = fwPage.fwItems[0];
```

*Example: 3:*
Set myItem to the last item on the page

```
var myItem = fwPage.fwItems[fwPage.fwItems.fwLength - 1];
```

**Property (1)**

**fwLength**
Number of items in the list. *Integer*

## Class FWLayoutElement

This class is the super class of all items, pages and folders. There are no actual instances of this class.

**Properties (4)**

**fwActions**
The list of actions applied to this item. *FWActionList*.

**fwContentFile**
This returns a string that is the full filename of the source content (if the element has content). Your *String* would use this to find the file that is responsible for the content e.g. the image file if the item has an image imported from an external file. If the item has no content it returns the empty string. For a page it will contain the background image.

**fwFolder**
The folder that this item appears in within the Site palette. *FWFolder*

**fwPage**
The page (if any) that the element is or appears on. *FWPage*

## Class FWLink

This class represents a link in Freeway. It can represent internal links (linked to other parts of Freeway) and external links.

### Properties (7)

**`fwExtended`**
These are the extended attributes that have been set on the link.*FWListData*

**`fwExternal`**
This is the external link that has been set. e.g. `'http://www.softpress.com'.` If there is no link or the *String* link is an internal link then this will return null.

**`fwHasLink`**
If a link has been set this is true. *Boolean*

**`fwInternal`**
This is the item such as an anchor or page that is linked to. If it is not an internal link then this property *FWElement* is null.

**`fwIsExternal`**
If there is an external link this is true. *Boolean*

**`fwIsInternal`**
Is true if there is an internal link. *Boolean*

**`fwTarget`**
This is the target that has been set on the link. *String*

### Methods (2)

**`fwClear()`**
Set the link object so that it is not linked to anything i.e it is cleared.

**`fwUrl([page])`**
This returns the URL as a string relative to a particular page – if the page is passed in.

## Class FWLinkParameter

This class represents the Link Parameter. It is a subclass of `FWParameter`.

The `fwValue` property will return a value of type `FWLink`.

## Class FWListData

This class is used to represent lists of items such as extended attributes.
*List of strings*

### Property

**`fwLength`**
Number of items in the list. The list will have name/value pairs e.g. meta tags. *Integer*

```
meta = fwPage.fwMetaTags
alert(meta["GENERATOR"]);
```

shows "freeway 3.0"

## Class FWOSAInterpreter

This class can be constructed within JavaScript. It provides an interface by which OSA scripts can be compiled, run and executed. This is most useful for allowing AppleScript to be executed from within Freeway Actions.

Note when executing the FWOSAInterpreter methods, the time-out is suspended.

The object works by having a script set in it. You build the script by a series calls to `fwWrite` and `fwWriteln`. Then you ask the object to compile the script, after which a call to `fwRun` will execute it.

*Example 1:*

The following script will allow the user to locate a file and that file will then be executed as an AppleScript.

```
// locate the file with the AppleScript in it
var theFile = new FWFile;
if (      theFile.fwLocateOld('Locate AppleScript File','TEXT')
       &&   theFile.fwOpen('read'))
{
   // create the object to interpret the script
   var osa = new FWOSAInterpreter;

   // read the text
   var text = theFile.fwRead(theFile.fwAvailable);

   // set the text in the OSA interpreter
   osa.fwWrite(text);

   // compile it
   osa.fwCompile();

   // run it
   var result = osa.fwRun();

   // if there are some results put them in an alert
   if (result && result != '')
      alert(result)
}
```

*Example 2:*

This example shows an action with a small amount of AppleScript written as markup. This markup is then executed in the OSA Interpreter. It provides the easiest method of adding AppleScript, which is generally a large amount of text, to an Action.

```
<action-markup custom name="osa">
tell application "FileMaker" display dialog (&param;)
end tell
</action-markup>

<action-javascript>
function fwATContent()
{
    var osa=new FOSAInterpreter;
    var text= fwMarkups["osa"];
    fwSubstitute(text);
    osa.fwWrite(text);
    osa.fwCompile();
    alert(osa.fwRun());
}
</action-javascript>
```

## Properties (2)

**`fwIsCompile()`**
Has the script been compiled? *Boolean*

**`fwScriptType`**
This is the type of script denoted by a 4 char code, by default it is AppleScript. *String (4 char)*

## Methods (5)

**`fwClear()`**
Clear the current script.

**`fwCompile`**
Compile whatever script is in the object. After compiling, the next time you call `fwWrite` it will first clear the current script.

**`fwRun()`**
Run the current script. If the script has not been compiled then it will be compiled at this point.

**`fwWrite(string[,string...])`**
This will add the arguments as a concatenated string into the script object. *One or more strings*

**fwWriteln([string[,string...]])**
This will add the arguments as a concatenated string into the script object, *Zero or more strings* followed by a carriage return.

## Class FWOutput

This is the super class of `FWTag` and `FWAttribute` used for creating output from Freeway. Raw output is stored as an `FWAttribute`.

### Properties (4)

**fwEnclosing**
The tag or raw data that encloses this attribute. *FWAttribute*

**fwFirst**
The first item that this tag contains. *String*

**fwLast**
The last item that this tag contains. *String*

**fwRoot**
The root or the tag that ultimately encloses everything, the same as `fwDocument.fwTags`

### Methods (17)

**fwAdd(name/htmltag[,after][,hasClose])**
This method will add a new tag if you pass in a string with the name of the new tag that you want to add.

*Arguments:*
*:*
`name/htmltag`: the name of a new tag or an existing tag after the tag after which it should be added `has Close` sets this to true if you want the tag to be closed

*Example: 1:*
This example will add a `<p></p>` after the first image tag and give it contents of the image "`srce`".

```
<page-action name="Label First Image">
<action-javascript>

function fwAfterEndBody()
{
    // Find the first img tag
    var firstImg = fwDocument.fwTags.fwFind("img");
    var enclosing = firstImg.fwEnclosing;

    // add a <p> tag after the <img> tag
    var para = enclosing.fwAdd("p", firstImg, true);

    // and give the paragraph tag raw data of the image path
    para.fwAddRaw(firstImg.src.toString());
}

</action-javascript>
</page-action>
```

**fwAddEnclosing(tagType/content[,hasClose])->FWTag**
This method adds an enclosing tag around an existing tag content either a tag or raw code. *FWTag*

*Arguments:*

`tagType/content` Either the name of a new tag to be created or an existing tag. *FWTag or raw code* `has Close` This specifies if the tag should have it's close attribute set. *Boolean*

*Example:*

This code finds the first `<img>` tag on the page and adds an enclosing link tag to it. It passes true as it wants the tag to be closed i.e. a `</a>` to be written as well.

```
<page-action name="Add Dummy Link">
<action-javascript>

   function fwAfterEndBody()
   {
      // Find the first <img> tag
      var firstImg = fwDocument.fwTags.fwFind("img");

      // add a link to it
      var linkTag = firstImg.fwAddEnclosing("a", true);

      // set the href
      linkTag.href='"#"';
   }

</action-javascript>
</page-action>
```

In this example the first `<img>` will be enclosed in a <a> tag so that it will change from `<img src= ... >` to `<a href="#"><img src= ... ></a>`

**`fwAddJavaScript([,after tag][string])`**
This is used to add JavaScript to the HTML output. You optionally *FWTag* specify the script name and version and Freeway will return a `<script>` tag. Freeway will look for a tag of the sort you request. If it finds one it will return that. If it does not it will create one. This method is very useful for adding JavaScript to a block of code. In addition to creating the tag Freeway will correctly enclose it in the correct hiding mechanism to prevent early browsers that do not understand JavaScript from having problems. It returns a `<script>` tag.

*Arguments:*

`after tag:` the tag to add this after. *FwTag*
`string`: the name of the script tag to add. *String*

*Example:*

The function below will add a piece of JavaScript stored in "action-markup" if not there:

```
// This appends a piece of JavaScript stored in /action-markup/ to
a specific tag
   // This appends a piece of JavaScript stored in /action-markup/
to a specific tag
   function AppendJavaScript(tag, markup)
   {
      // Append a piece of markup if it is not already defined
      if (!fwPage[markup])
      {
         var headTag = fwDocument.fwTags.fwFind(tag);
         if (headTag)
         {
            var javascript = headTag.fwAddJavaScript();
            javascript.fwAddRawOpt(fwMarkups[markup]);
            fwPage[markup] = true;
         }
      }
   }
```

**fwAddRaw([text1][,after][,text2])**
This returns the FWAttribute that has been created to hold the raw data.

*Arguments:*

`text1` If there is no `text2` then this is the raw value. If there is a `text2` then this denotes the name of the raw text added.
`after:` This is the content object that it should be placed after. If there is no content object then it will be appended.
`text2:` This is the value of the object.

*Example: 1:*
This will append the text "Test" as raw data after the last tag within the body.

```
function fwAfterEndBody()
{
   var body = fwDocument.fwTags.fwFind("body");
   body.fwAddRaw("Test");
}
```

*Example: 2:*
This action will append the text "Test" as raw data after the head tag within the HTML.

```
function fwAfterEndBody()
{
   var html = fwDocument.fwTags.fwFind("html");
   var head = fwDocument.fwTags.fwFind("head");
   html.fwAddRaw("Test", head);
}
```

**fwAddRawln[text1][,after][,text2])**
As fwAddRaw but adds a carriage return.

**fwAddRawOpt([text1][,after][,text2])**
As fwAddRaw but adds a carriage return at the end if more readable code is set in the preferences.

**fwDelete**
This will delete a tag so that it will not be included in the output.

*Example:*
This will delete the first image on the page.

```
<page-action name="Delete First Image">
<action-javascript>

   function fwAfterEndBody()
   {
      // Find the first img tag
      var firstImg = fwDocument.fwTags.fwFind("img");

      // Delete it
      firstImg.fwDelete();
   }

</action-javascript>
</page-action>
```

Note: Deleting this image will simply remove it from the page. The table structure, or layer that this image would have appeared in will be preserved.

**fwFind([owner][after tag][type [,name [,value]]])**
This method allows you to find the first occurrence of a tag that satisfies a particular criteria. It returns the tag found or null.

*Arguments:*

- `ownerfind:` a tag that was owned (created) by this element *FWElement*
- `aftertag:` find a tag after this one. *FWTag*
- `name` find a tag with an attribute of this name`value` find a tag that has an attribute of this value`type`name of tag (eg.”`img`”)

*Example 1:*
This finds the first `<img>` tag in the document

```
function fwAfterEndBody()
{
   // find first tag
   var firsttag = fwDocument.fwTags.fwFind("img");

   alert(firsttag);
}
```

*Example 2:*

This action finds the first `<img>` tag that belongs to the item onto which the Action is applied.

```
<item-action name="Find First Img">
<action-javascript>

   function fwAfterEndBody()
   {
      // find first tag
      var firsttag = fwDocument.fwTags.fwFind("img", fwItem);

      alert(firsttag);
   }

</action-javascript>
</item-action>
```

*Example: 3:*
This method will find the first `<img>` with a name attribute

```
function fwAfterEndBody()
   {
      // find first tag with a name parameter
      var firsttag = fwDocument.fwTags.fwFind("img","name");

      alert(firsttag.name);
   }
```

*Example: 4:*
This method will find the first <img> with a name attribute whose value is "foobar".

```
function fwAfterEndBody()
   {
      // find first tag with a name parameter
      var firsttag = fwDocument.fwTags.fwFind("img","name");

      alert(firsttag.name);
   }
```

*Example 5:*
This method will find the first <img> with a name attribute.

```
function fwAfterEndBody()
   {
      // find first tag with a name parameter
      var firsttag = fwDocument.fwTags.fwFind("img","name");

      alert(firsttag.name);
   }
```

*Example 6:*
This method will count up all the <img> tags that appear in your document. After finding the first tag it passes the current tag to the fwFind method as the point to start the find from. This finds the first <img> tag in the document.

> Note: It is more efficient to use `fwFindAll` for this sort of situation.

```
<page-action name="Find All Images">
<action-javascript>

   function fwAfterEndBody()
   {
      // clear the counter
      var totalImages = 0;

      // find first <img> tag
      var currentTag = fwDocument.fwTags.fwFind("img");

      while (currentTag)
      {
         totalImages++;

         // find the next tag after the current tag
         currentTag = fwDocument.fwTags.fwFind("img", currentTag);
      }

      alert("total <img> = ",totalImages);
   }

</action-javascript>
</page-action>
```

**`fwFindAll([owner][after tag][type [,name [,value]]]`**
This method allows you to find all the tags that satisfy a particular criteria. If no tags are found then it will return an array with no elements. The array will contain all the tags found in the order in which they where found.

*Arguments:*

- `owner:` Finds a tag that was owned (created) by this element. *FWElement*
- `after tag`: Find a tag after this one. *FWTag*
- `name` Find a tag with an attribute of this name. *FWTag*
- `value` Find a tag that has an attribute with this value. *FWTag*

*Example: 1:*

This action will count up all the `<img>` tags that appear in your document by finding all the `<img>` tags

```
<page-action name="Find All Images">
<action-javascript>

   function fwAfterEndBody()
   {
      // find all the <img> tags
      var images = fwDocument.fwTags.fwFindAll("img");
      alert(images.length);
   }

</action-javascript>
</page-action>
```

*Example: 2:*

This action will count all the <img> tags that belong to the item onto which the action is applied.

```
<item-action name="Find All Images">
      <action-javascript>
            function fwAfterEndBody()
                  {// find all the <img> tags that belong to this
                  item
                  var images = fwDocument.fwTags.fwFindAll("img",
                  fwItem);
                  alert(images.length);}
      </action-javascript>
</item-action>
```

**fwFindAllContent([owner][after tag][type [,name [,value]]])**

This method allows you to find all the tags that are contained within an `FWOutput`. It refers to an array of tags.

*Example:*

The following example will remove all the tags except for those that are within the `<body></body>`. This sort of technique can be useful for generating server-side includes and page fragments.

```
<page-action name="Body Contents">
<action-javascript>

    function fwBeforeEndHTML()
    {
        body = fwDocument.fwTags.fwFind("body");
        bodyContent = body.fwFindAllContent();
        html = fwDocument.fwTags.fwFind("html");
        html.fwDelete();
        fwDocument.fwTags.fwMove(bodyContent);
    }

</action-javascript>
</page-action>
```

**`fwFindContent([owner][after tag][type [,name [,value]]])`**
This method allows you to Find the first occurrence of a tag that satisfies a particular criteria. It will only search within the tag's contents and will not search hierarchically through the tag tree. It returns the tag found or null.

*Arguments:*

- `owner:` Finds a tag that was owned (created) by this element. *FWElement*
- `after tag:` Find a tag after this one. *FWTag*
- `name:` Find a tag with an attribute if name is FWTag. *String*
- `value:` Find a tag that has an attribute with this value. *String*

**`fwFindEnclosing(tag[,name[,value]])`**
This method is designed to find a tag that encloses a given tag. It returns *FWTag* – the tag that has been found or null if nothing has been found.

*Arguments:*

- `tag:` the tag to be found *String*
- `name:` look for a tag with this argument. *string*
- `value:` the argument has to have this value. *String*

*Example:*

The following action will look for an enclosing link <a> tag. If it does not find one it will add one. This sort of technique is useful when writing rollovers.

```
<page-action name="Find or Add Link">
<action-javascript>

   function fwAfterEndBody()
   {
      // find first <img>
      var firstImg = fwDocument.fwTags.fwFind("img");

      // find the link
      var link = firstImg.fwFindEnclosing("a");

      // if there is no link then add one
      if (!link)
      {
         link = firstImg.fwAddEnclosing("a");
         link.href='"#"';
      }
   }

</action-javascript>
</page-action>
```

**fwFindRaw([string][,after tag][,owner])**
This will find raw text in the tags. All the arguments are optional.
*FWAttribute*

If you do not specify a string then Freeway will return the first raw data that matches the other search criteria. If you specify an `after tag` Freeway will start looking after this tag. If you specify an owner then Freeway will only return a tag created by that particular owner. If nothing matching this criteria was found or no raw data was found it returns null.

*Arguments:*

- `string:` the text to be found.
- `after tag:` start looking after this tag *FWTag*
- `owner:` only find raw data with this owner *FWElement*

*Example:*

The following code appends the raw data "foobar" to the body tag. It then uses `fwFindRow` to search for the text “foobar” and displays what it finds in an alert.

```
<page-action name="foobar">
<action-javascript>

    function fwAfterEndBody()
    {
        // Find the first img tag
        var body = fwDocument.fwTags.fwFind("body");

        // add the raw data "foobar"
        body.fwAddRaw("foobar");

        // find it
        var newRaw = body.fwFindRaw("foobar");

        // display an alert with it's value
        alert(newRaw);
    }

</action-javascript>
</page-action>
```

**fwIndent()**
This will increase the indent level if more readable code is specified in the preferences.

**fwMove([htmlTag/tagArray][after tag])**
This action is designed to remove the tags from a page that make it a frame-set so that you can more-easily view the NoFrames content (you do not have to set your browser preferences to check it). As the Action is doing something radical to your page it posts an alert warning you that something you probably don't normally want to happen is going on. Generally you would never post an alert during the publishing process. The action also has a `fwIsDirty` method that will force the page to be republished, so that you will always get the warning.

To remove the `FrameSet` tags the Action has to first remove the `<frameset>` tag. Then it has to remove the `<noframes>` tag. However before it removes the `<noframes>` tag it has to move `<body>` tag that is within it. It moves the body tag by calling `fwMove`, and passing in an existing tag (the body tag).

*Example:*

```
<page-action name="Preview NoFrames Page">
<action-javascript>

   function fwIsDirty()
   {
      return true;
   }

   function fwAfterEndBody()
   {
      alert('Your FrameSet has been deleted on page "',fwPage,
         '" to allow you to preview the "','NoFrames', '"
content.');

      // find and delete the <frameset> tag
      var getFrm = fwDocument.fwTags.fwFind("frameset");
      getFrm.fwDelete();

      // find the <body> tag
      var getBdy = fwDocument.fwTags.fwFind("body");

      // find the <noframes> tag
      var getNfm = fwDocument.fwTags.fwFind("noframes");

      // move the <body> so it is after the <noframes> tag
      // and no longer enclosed by it
      getNfm.fwEnclosing.fwMove(getBdy, getNfm);

      // delete the noframes tag
      getNfm.fwDelete();
   }

</action-javascript>
</page-action>
```

**fwOutdent()**
This will decrease the indent level if more readable code is specified in the preferences.

**fwToHTML([efficientCode ][upperCase][newLineType])**
This will convert a tag and it's contents into a text string. This text string is the text that would be output when the tags are turned into HTML. This is particularly useful if you want to make some parts of the page generated by

JavaScript within the page i.e. in the generated HTML page's document.write.

*Arguments:*

- efficientCode 0 = use document setting, readable = 1 , efficient = 2
- upperCase 0 = use document setting, lowercase = 1, uppercase = 2
- newLineType 0= use document setting, Macintosh = 1, UnixNewlines = 2, DOSNewlines = 3

> Note: If you do not pass any arguments the document defaults will be used.

*Example:*

The following example will display the HTML of the item to which it is applied in an alert.

```
<item-action name="fwToHtml">
<action-javascript>

    function fwBeforeEndHTML()
    {
        ourTag = fwDocument.fwTags.fwFind(fwItem);
        alert(ourTag.fwToHTML());
    }

</action-javascript>
</item-action>
```

## Class FWPage

This class represents a page within the Freeway document.

### Properties (13)

**fwAlignment**
This is the alignment of the page. left = 1, center =2, right = 3 *Integer*

**fwAlinkColor**
This returns the active link color that has been set on the page. *FWColor*

**fwBgColor**
This returns the background color that has been set on the page. *FWColor*

**fwEncoding**
This returns the Internet name encoding that has been set on a page (eg. “ISO-8859-1”). *String*

**fwFileName**
This is the file name of the page. *String*

**fwHeight**
This is the height of the page within Freeway. *Integer*

**fwHTMLLevel**
This is the HTML level that will be generated from this page (HTML 3.2 = 1, |HTML3.2+CSS = 2, HTML4 = 3).

> Note: the return value of this is incorrect in Freeway 3 but is corrected in Freeway 3.1

**fwLinkColor**
This returns the link color that has been set on the page. *FWColor*

**fwScript**
This returns the script code of a page. (See Tech Note: “Localizing Freeway Actions”)

*Example:*

The following code fragment behaves differently if the page has a script-code of "1" i.e. is Japanese. It returns the encoding that has been set on a page.

```
// if we are on a Japanese page then add Japan in Japanese
if (fwPage.fwScript==1)
   addOption(tag, "日本");
else
   addOption(tag, "Japan");
```

**fwSystemMetaTags**
This returns a list of system meta tags on the page.*FWListData*

**fwUserMetaTags**
This returns a list of user meta tags on the page.*FWListData*

**fwVlinkColor**
This returns the visited link color (`FWColor`) that has been set on the page. *FWColor*

**fwWidth**
This is the width of the page. *Integer*

## Class FWParameter

This class is the basic parameter that is stored for parameters that are set in the Freeway Actions by the user such as <action-text>, <action-number>, <action-checkbox>, and <action-popup> are all of this type and fwValue is a string for these. There are three subclasses of FWParameter which have other parameter types. These are Class FWLinkParameter, Class FWColorParameter and Class FWFileParameter with its own subclass Class FWImageParameter.

### Properties (8)

**fwAction**
The action to which this parameter belongs. *FWAction*

**fwBoolValue**
The value of this parameter as a Boolean. *Boolean*

**fwContentFile**
This returns a string that is the full filename of parameters source if it has a source file. This is only possible for <action-file> and <action-image>. If the parameter has no file it returns the empty string. *String*

**fwHasImages**
Returns whether there are any images associated with this parameter. It returns true only if the parameter. Has images that are generated, not if it has a file associated with it. *Boolean*

**fwName**
Name of the parameter. *String*

**fwParameters**
The parameter list of which this parameter is a member.

**fwType**
The type of the parameter. It has the following values:

Code

**`fwValue`**
The value that has been assigned to this parameter or set by the user. The type depends on the type of parameter (see individual parameter types eg *FWlinkParameter, FWColorParameter, FWFileParameter, FWImageParameter*).

## Methods (3)

**`fwClear`**
This clears the current parameter.

**`fwFindAllImages()`**
This will return all the images associated with a particular parameter in an array. *String*

**`fwFindImage(imageNumber)`**
This returns a specific image associated with the image number parameter. If the image is sliced then there can be more than one image. The index of the image is zero-based. If there is no image at a particular index then it will return null. If the parameter is not an image parameter it will return null. *String*

## User Methods/Properties ( 4)

**`fwEnable`**
This property allows you to control if a parameter is enabled in the interface (in the Actions palette). If *Boolean* you define this property as a method then Freeway will evaluate the method to return a Boolean value and use that to determine if the parameter is enabled. If you define this property as a Boolean i.e. set it to true or false then Freeway will use this value to determine if the parameter is enabled.

*Example:*

This example sets the `fwEnable` property of the parameter to be a Boolean from the value of a checkbox. Using this technique the checkbox will control if the parameter "`param1`" is enabled in the interface.

```
<item-action name="Enable Test">
<action-checkbox name ="enable"/>
<action-text name ="param1"/>
<action-javascript>

    function fwInterface()
    {
        var enabled = fwParameters["enable"].fwBoolValue;
        fwParameters["param1"].fwEnable = enabled;
    }

</action-javascript>
</item-action>
```

*Example:*

This example sets the `fwEnable` property of the parameter to be the method `CalcVisibility` which is then evaluated to determine if the parameter is enabled or not. Using this technique the checkbox will control if the parameter "`param1`" is enabled in the interface.

```
<item-action name="Enable Test">
<action-checkbox name ="enable"/>
<action-text name ="param1"/>
<action-javascript>

    function CalcEnable()
    {
        var enabled = fwParameters["enable"].fwBoolValue;
        return enabled;
    }

    function fwInterface()
    {
        fwParameters["param1"].fwEnable = CalcEnable;
    }

</action-javascript>
</item-action>
```

**`fwMenuItems`**

This property allows you to generate the items that appear in a popup menu programatically. This has the advantage that you can make menus that reflect names of items on a page or are derived from the results of

executing OSA scripts. This property should either be an array or a function that returns an array.

*Example: 1:*

This code will create a menu with the item labelled "`item [0]`", "`item[1]`" etc.

```
<item-action name="Test Popup">
<action-popup name="Test Popup"/>
<action-javascript>

   function fwInterface()
   {
      var array = new Array;
      for (i = 0 ; i < 20 ; i++)
         array[i]="test ["+i+"]";
      fwParameters["Test Popup"].fwMenuItems = array;
   }

</action-javascript>
</item-action>
```

*Example: 2:*

This code will create a popup menu that is based on values selected from another menu.

```
<page-action name="Smart Popups">
<action-popup name="Data Base">
    <value name="Customers.fp5"/>
    <value name="Products.fp5"/>
    <value name="Orders.fp5"/>
</action-popup>
<action-popup name="Fields"/>
<action-javascript>

    function fwInterface()
    {
        var array = new Array;
        var dataBase = fwParameters["Data Base"].toString();

        switch (dataBase)
        {
           default:
           case "Customers.fp5":
              array.push("name");
              array.push("address");
              array.push("city");
              array.push("state");
              array.push("zip");
              break;

           case "Products.fp5":
              array.push("quantity");
              array.push("amount");
              array.push("total");
              array.push("tax");
              break;

           case "Orders.fp5":
              array.push("product_id");
              array.push("price");
              array.push("weight");
              array.push("color");
              array.push("buy_url");
              break;
        }

        fwParameters["Fields"].fwMenuItems = array;
    }
```

*Example: 3:*

This code will create a menu based on the selection of a file. The file is expected to be of the format:

```
Customers.fp5
    name
    address
    city

Products.fp5
    product_id
    price
    weight

Orders.fp5
    quantity
```

The action then parses this file and creates two menus from it

```
</page-action>
<page-action name="Smart File Popups">
<action-file name="DB File">
   <value type="TEXT">
</action-file>
<action-popup name="Data Base">
</action-popup>
<action-popup name="Fields"/>
<action-javascript>

   function BuildDataBaseMenu(file)
   {
      file.fwPosition=0;
      var array = new Array;

   // read all the lines in the file
      while (file.fwAvailable)
      {
         var s=file.fwReadln();

         // ad it to the menu only if it contains the string
".fp5"
         if (s.indexOf(".fp5")>0)
         array.push(s);
      }

      return array;
   }

   function BuildFieldMenu(file, database)
   {
      file.fwPosition=0;
      var array = new Array;

   // skip to the 'database'
      while (true)
      {
         if (!file.fwAvailable)
            return array;

         var s=file.fwReadln();
         if (s==database)
         break;
```

```
    }

    // read it's fields
    while (true)
    {
       if (!file.fwAvailable)
          return array;

       var s=file.fwReadln();
       if (s.indexOf(".fp5")>0)
          return array;

       array.push(s);
    }

    return array;
}

function GetMenuSelection(menu, menuArray)
{
    var selection = menu.toString();
    if (!selection && menuArray.length)
       return menuArray[0];

    for (var i in menuArray)
    {
       if (menuArray[i]==selection)
          return selection;
    }

    if (menuArray.length)
       return menuArray[0];

    return "";
}

function fwInterface()
{
    var theDBfile = fwParameters["DB File"];
    var theDatabaseMenu = fwParameters["Data Base"];
    var theFieldMenu = fwParameters["Fields"];

    if (theDBfile.fwHasFile)
```

```
      {
         theFile = new FWFile;
         if (theFile.fwOpenRead(theDBfile))
         {
            var dataBaseMenuOptions = BuildDataBaseMenu(theFile);
            theDatabaseMenu.fwMenuItems = dataBaseMenuOptions;
            var dataBaseOption=GetMenuSelection(theDatabaseMenu,
dataBaseMenuOptions);

            var fieldMenuOptions = BuildFieldMenu(theFile,
dataBaseOption);
            theFieldMenu.fwMenuItems = fieldMenuOptions;
            fieldMenuOptions.toString();

            theFile.fwClose();

            theDatabaseMenu.fwEnable=true;
            theFieldMenu.fwEnable=true;
            return;
         }
      }

      theDatabaseMenu.fwEnable=false;
      theFieldMenu.fwEnable=false;
   }

</action-javascript>
</page-action>
```

**fwTitle**

This property allows you to control the title of a parameter as it appears in the actions palette. If you define this property as a method then Freeway will evaluate the method to return a string value which will be used to set the title that appears in the parameter. If you define this property as a string value then Freeway will use this value as the title. *String*

*Example: 1:*

The following example will change the title of one of "param1" between on and off by setting the `fwTitle` property to the string "on" or "off"

```
<item-action name="Title Test">
<action-checkbox name ="on/off"/>
<action-text name ="param1"/>
<action-javascript>

   function fwInterface()
   {
      var isOn = fwParameters["on/off"].fwBoolValue;
      if (isOn)
         fwParameters["param1"].fwTitle = "on";
      else
         fwParameters["param1"].fwTitle = "off";
   }

</action-javascript>
</item-action>
```

*Example: 2:*

The following example will change the title of one of "`param1`" between on and off by setting the `fwTitle` property to the method `CalcTitle`.

```
<item-action name="Title Test">
<action-checkbox name ="on/off"/>
<action-text name ="param1"/>
<action-javascript>

   function CalcTitle()
   {
      var isOn = fwParameters["on/off"].fwBoolValue;
      if (isOn)
         return "on";
      else
         return "off";
   }

   function fwInterface()
   {
      fwParameters["param1"].fwTitle = CalcTitle;
   }

</action-javascript>
</item-action>
```

**fwVisible**

This property allows you to control if a parameter appears in the interface (in the actions palette) or not. *Boolean* If you define this property as a method then Freeway will evaluate the method to return a Boolean value and use that to determine the parameter's visibility. If you define this property as a Boolean i.e. set it to true or false then Freeway will use this value to determine if the item is visible or not.

*Example:*

This example sets the `fwVisible` property of the parameter to be a Boolean from the value of a checkbox. Using this technique the checkbox will control if the parameter "`param1`" is visible in the interface.

```
<item-action name="Visible Test">
<action-checkbox name ="visible"/>
<action-text name ="param1"/>
<action-javascript>

   function fwInterface()
   {
      var visible = fwParameters["visible"].fwBoolValue;
      fwParameters["param1"].fwVisible = visible;
   }

</action-javascript>
</item-action>
```

*Example:*

This example sets the `fwVisible` property of the parameter to be the method `CalcVisibility` which is then evaluated to determine if the parameter is visible or not. Using this technique the checkbox will control if the parameter "`param1`" is visible in the interface.

```
<item-action name="Visible Test">
<action-checkbox name ="visible"/>
<action-text name ="param1"/>
<action-javascript>

   function CalcVisibility()
   {
      var visible = fwParameters["visible"].fwBoolValue;
      return visible;
   }

   function fwInterface()
   {
      fwParameters["param1"].fwVisible = CalcVisibility;
   }

</action-javascript>
</item-action>
```

## Class FWParameterList

This defines a list of `FWParameters`.

*Example:*

Get the Boolean value of the parameter "myParam".

```
var isOn = fwParameters["myParam"].fwBoolValue;
```

or

```
var isOn = fwParameters.myParam.fwBoolValue;
```

*Example:*

Get the Boolean value of the first parameter.

```
var isOn = fwParameters[0].fwBoolValue;
```

*Example:*

Get the Boolean value of the last parameter.

```
var isOn = fwParameters[fwParameters.fwLength - 1].fwBoolValue;
```

## Class FWTag

This class represents a tag. The attributes appear as normal properties, so you can add or change them by setting them as properties.

You can construct these by using `fwAdd` and `fwAddEnclosing`. This class is a subclass of Class `FWOutput`.

*Example: 1:*

The following Action will name all the images foobar0, foobar1, foobar2, etc.

```
<page-action name="Name all img">
<action-javascript>

   function fwAfterEndBody()
   {
      // find all the <img> tags that belong to this item
      var images = fwDocument.fwTags.fwFindAll("img");
      for (i in images)
      {
         var image = images[i];
         image.name = '"'+'foobar'+i+'"';
      }
   }

</action-javascript>
</page-action>
```

*Example: 2:*

The following Action will name all the images without names, so that the first will have name="`foobar0`", the second will have name=" `foobar1`", then name=" `foobar2`" etc

```
<page-action name="Name all unnamed img">
<action-javascript>

   function fwAfterEndBody()
   {
      // find all the <img> tags that belong to this item
      var j = 0;
      var images = fwDocument.fwTags.fwFindAll("img");
      for (i in images)
      {
         var image = images[i];
         if (!image.name)
         {
            image.name = '"'+'foobar'+j+'"';
            j++;
         }
      }
   }

</action-javascript>
</page-action>
```

## Properties (2)

**`fwClose`**
This determines if the closing form of the tag is written. e.g. the `</p>` of a <p> tag.

**`fwOwner`**
The primary element that created this tag.

## Method

**`fwAddJToTag(name,value)`**
This method is designed to make it easier to append JavaScript statements to tag values. Your value is appended to the end of any existing value but before the statement `return true` or `return false`.

*Arguments:*

`name` the names of the argument to add`value` the argument to add

*Example:*

This will add two alert calls to the onclick of the enclosing link so that it is terminated with return false.

```
<item-action name="Click Alert">
<action-javascript>

   function fwAfterEndBody()
   {
      // find all the <img> tags
      var image = fwDocument.fwTags.fwFind("img");
      var link = image.fwFindEnclosing("a");
      link.fwAddJToTag("onclick","return false");
      link.fwAddJToTag("onclick","alert('click')");
      link.fwAddJToTag("onclick","alert('click again')");
   }

</action-javascript>
</item-action>
```

# Global Properties and Methods

**prompt(string[,string,[script]])**

*Arguments:*

- String that appears as a prompt.
- Default Text.
- Script for the text to be input in (default is roman).

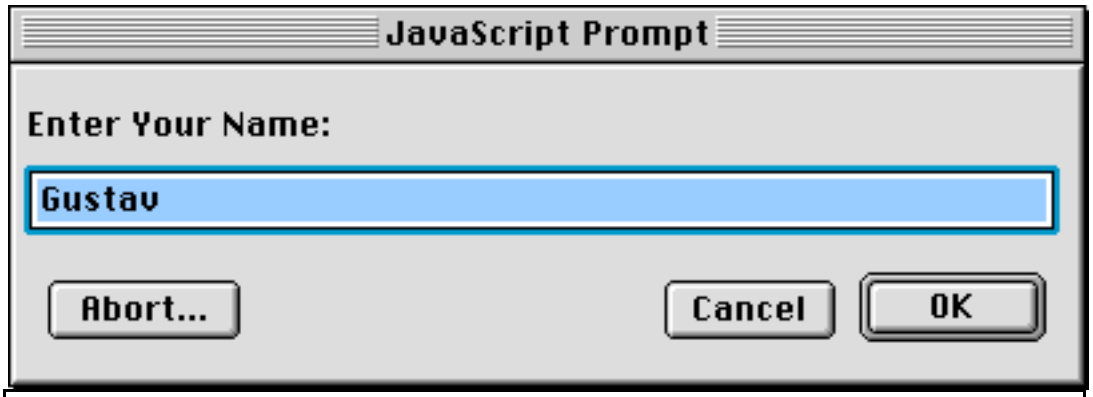
*Return Value:*

Null if the user clicked cancel otherwise the string that they entered. This will prompt the user for input.

*Example:*

```
var s = prompt("Enter Your Name:","Gustav");
```

Will give rise to this dialog, with "Gustav" as the default value.




Note: this dialog is intended for the user to enter small amounts of text. It is not possible to enter more than 255 characters. If you want the user to edit a lot of text you should use `fwBigPrompt`.

**`fwBigPrompt(string[,string])`**

*Arguments:*

- String that appears as a prompt
- Default Text

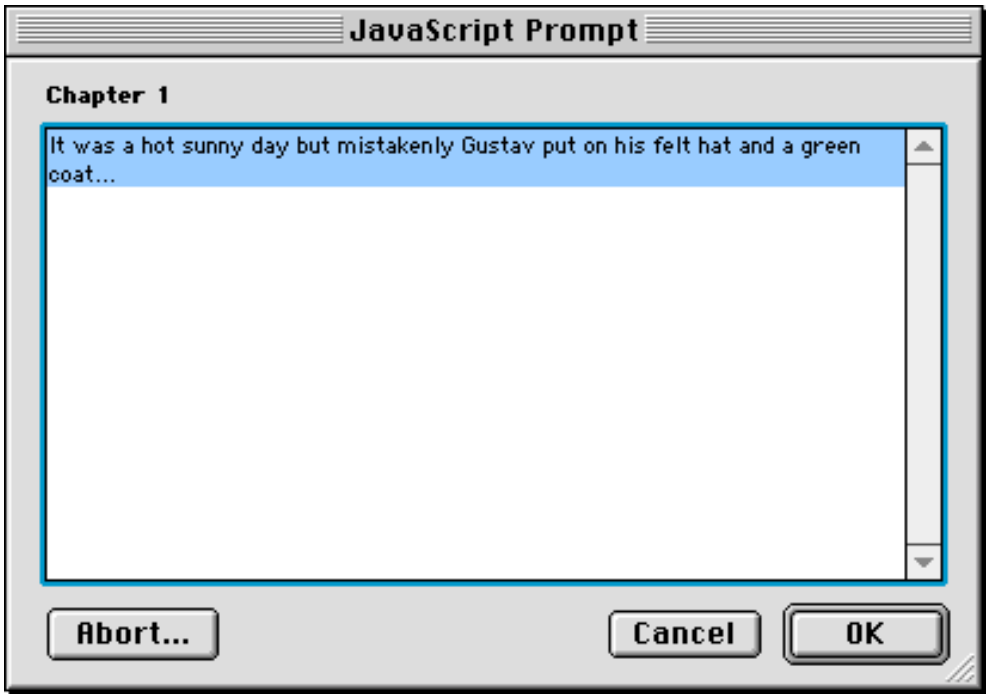
*Return Value:*

`Null` if the user clicked cancel otherwise the string that they entered. This will prompt the user for input.

*Example:*

```
var s = fwBigPrompt("Chapter 1","It was a hot sunny day but
mistakenly Gustav put on his felt hat and a green coat...");
```

Will give rise to this dialog:




## Class FWAction

**`fwParameterChanged(newParameter,oldParameter)`**

*Arguments:*

The new parameter that is intended to be set and the existing parameter.

*Return Value:*

If this function returns true then Freeway will set the new parameter. If the function returns false the new parameter will not be set. If the function simply returns nothing then the parameter will be set.

*Description:*

This method, if present in your Freeway Action, is called when Freeway after the user has changed a parameter in a Freeway action. It gives you, the action writer, an opportunity to validate what has been set, correct it or reject it. It also gives you an opportunity to set the values of other parameters if you need to.

*Example:*

This example action will validate the field "test" and reject any value that is greater than 4 characters.

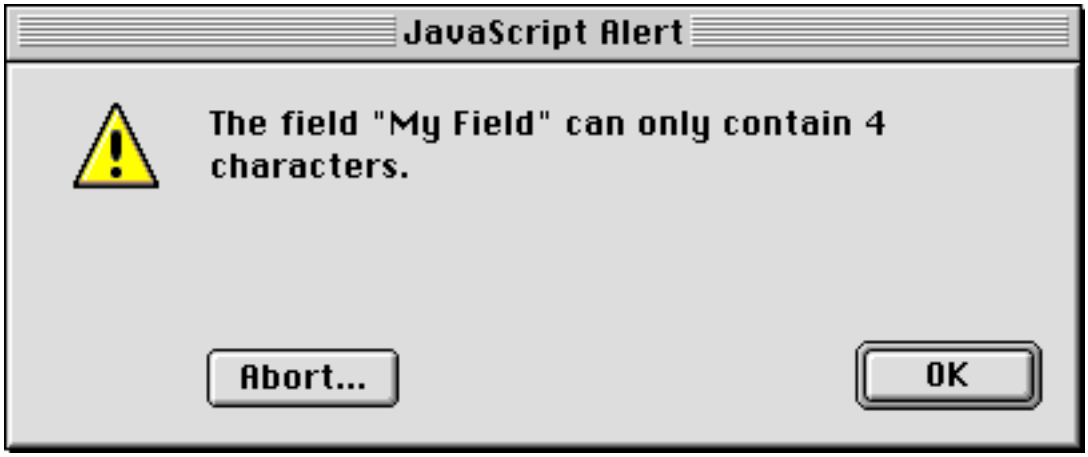
```
<page-action name="Test Parameter Changed">
<action-text name="text" title="My Field">
<action-javascript>

   function fwParameterChanged(newParam, oldParam)
   {
      switch (newParam.fwName)
      {
         // check the "text" parameter
         case "text":
           var str = newParam.toString();
           if (str.length>4)
           {
             alert('The field "', newParam.fwTitle,'" can only
contain 4 characters.');
             return false;
           }
           break;
      }

      return true;
   }

</action-javascript>
</page-action>
```

If you enter more than 4 characters the previous value will be restored and you will get this dialog.



*Example:*

This example will change the value entered by the user truncating it to 4 characters.

```
function fwParameterChanged(newParam, oldParam)
{
   switch (newParam.fwName)
   {
      // check the "text" parameter
      case "text":
        var str = newParam.toString();
        if (str.length>4)
        {
          alert('The field "', newParam.fwTitle,
            '" has been truncated to 4 characters.');
          newParam.fwValue=str.slice(0,4);
        }
        break;
   }

   return true;
}
```

*Example:*

This example has a popup menu and a text field. It the user selects a value from the popup the value of the text field will change as well.

```
<page-action name="Test Parameter Changed">
<action-popup name="Material">
 <value name="Wood"/><value name="Paper"><value name="Steel"><value
name="Plastic">
</action-popup>
<action-text name="text" title=" ">
<action-javascript>

   function fwParameterChanged(newParam, oldParam)
   {
      switch (newParam.fwName)
      {
         case "Material":
            var str = newParam.toString();
            fwParameters["text"].fwValue = str;
            break;
      }
   }

</action-javascript>
</page-action>
```

**fwAtContent([string])**

*Arguments:*

The full path of the file that is the original content file.

*Return Value:*

The full path of the content file that you want to be used in it's place. If you do not want to change the file return the file that was passed in or `null`.

*Description:*

This method, if present in your Freeway Action, is called when Freeway is going to make a given content file a part of your site and referenced by a given item. It makes it possible for a Freeway action to specify a different file in it's place. This is very useful if you have a Freeway Action that will, say, generate a custom flash-file as a temporary file and wants that file to be used rather than the origins.

*Example:*

The following example generates a new flash file based on the original one but sets the BG colour of the flash file to that of the item.

```
function fwContentFile(inFileStr)
{
    inFile = new FWFile;
    outFile = new FWFile;

    if (inFile.fwOpenRead(fwItem) &&
    // open the file associated with the content of fwItem
       outFile.fwOpenWrite(inFile.toString(),"SWFL","SWF2"))
    // open a file for writing with the same name as inFile
    {
       // read the file to get the info
       var info = inFile.fwReadFlashInfo();

       // set the BG colour of the move to the same as the item
       info.fwBgColor = fwItem.fwBgColor.fwValue;

       // reset the file position back to the begining of the
file
       inFile.fwPosition = 0;

       // replace the urls in the inFile writing them to the
outFile
       outFile.fwWriteFlash(info, inFile);

       // close both files
       outFile.fwClose();
       inFile.fwClose();

       // refurn the path of the new file
       return outFile.fwPathName;
    }

    outFile.fwClose();
    inFile.fwClose();

    return inFileStr;
}
```

*Property*

fwActionType(read)

*Description:*

This is the type that an action is. The following types are defined

```
Object Action = 0
Page Action= 1
Item Action = 2
Library Action = 3
```

*Property*

```
fwVersion(read)
```

*Description:*

This is the version as defined in `version` attribute of the `action-version` tag within an action. So for an action with the following version information:

```
<action-version version="1.0">
      “Non Repeating BG” Action
      Softpress Systems Limited 2000
</action-version>
```

The code fragment:

```
alert(fwVersion);
```

Will yield an alert displaying "1.0"

If the version has not been defined then the property will return `null`.

## Class FWElement

*Property*

```
fwPage
```

*Description:*

The page that an element appears on. For a `FWItem` this page will be the page that it is on. For a `FWPage` it will be it's self. For a `FWFolder` it will be null.

**`fwFindAction([type][name/namearray][searchchildren])`**

*Arguments:*

- type - the type of actions to search for:

```
All Actions = -1
Object Action = 0
Page Action = 1
Item Action = 2
Library Action = 3
```

- `name` *(string)* or array of strings - the name of the action(s) to search for:
- `searchchildren` *(boolean)* - should the children be searched for this action

*Returns:*

The first action that satisfies the search, or null if nothing has been found.

*Description:*

This method is used to find other actions within the Freeway document. You might typically use it of you have an action that works with one or more actions. In such a case you will need to find those actions that you wish to work with.

*Example:*

The following code fragment will find the first item action with the name of "Wibble" that is applied to the page.

```
var action = fwPage.fwFindAction(2,"Wibble", true);
```

**fwFindAllActions([type][name/namearray][searchchildren])**

*Arguments:*

- type - the type of actions to search for:
  ```
  All Actions = -1
  Object Action = 0
  Page Action = 1
  Item Action = 2
  Library Action = 3
  ```
- name (string) or array of strings - the name of the action(s) to search for:
- searchchildren (boolean) - should the children be searched for this action

*Returns:*

An array of all the actions that satisfy the search. If no actions are found then an empty array will be returned.

*Description:*
This method is used to find other actions within the Freeway document. You might typically use it of you have an action that works with one or more actions. In such a case you will need to find those actions that you wish to work with.

*Example:*
This code snippet will find all the actions that are applied to a page or items on that page.

```
var array = fwPage.fwFindAllActions(true);
alert(array.join("\r"));
```

## Class FWFile

**fwModificationDate()**

*Arguments:*

None:

*Returns:*

A JavaScript date object with the modification date of the file.

*Description:*

This will return the date that a file was last modified.

**fwWriteBytes([string/num/array/bool..string/num/array/bool])**

*Arguments:*

One or more string, number, boolean or array objects

*Description:*

This method allows you to write binary data to a file. You can pass in a number of arguments that will be written. Arguments are written as follows:

1. **Strings** - one byte is written for each character in the string.

2. **Numbers** - numbers are converted into integers. The lowest byte of the integer is written.

3. **Boolean** - 0x01 is written if the value is true otherwise 0x00 is written

4. **Null** - 0x00 is written

5. **void** - 0x00 is written

6. **Arrays** - the elements of the arrays are written recursively using `fwWriteBytes`.

7. **Other objects** - nothing is written

*Example:*

The following action will write a 1 pixel GIF of a colour set in by the colour parameter when you click on the button.

```
<page-action name="Make GIF">
<action-color name="color">
<action-button name="make gif" onclick = "makeGIF()">
<action-javascript>

   function GetColorPalette()
   {
      var result = new Array;
      var color = fwParameters["color"].fwValue;

      if (color.fwHasValue)
      {
         var selecetdColor = color.fwValue;

         result.push(selecetdColor.red);
         result.push(selecetdColor.green);
         result.push(selecetdColor.blue);
      }
      else
      {
         result.push(0xff);
         result.push(0xff);
         result.push(0xff);
      }
      return result;
   }

   function makeGIF()
   {
      outFile = new FWFile;

      var color = GetColorPalette();

      if
(outFile.fwLocateWrite("Graphic","peanut.gif","GIFf","MSIE"))
      {
         outFile.fwWriteBytes(
           'G', 'I', 'F', // Signature
           '8', '9', 'a', // Version
           1, 0,           // Width: 1 pixel
           1, 0,           // Height: 1 pixel
           0x80,           // Flag: 1-bit global Color table
           0,            // No background Color
```

```
            0,              // No aspect ratio

            color,            // Palette
            0, 0, 0,

            '!',              // GIF extension label
            0xf9,             // Graphic control label
            4,              // Extension block size
            false,            // Transparency flag
            0, 0,             // No delay
            0,              // Transparent Color
            0,              // Block terminator

            ',',              // Image descriptor label
            0, 0,             // X position
            0, 0,             // Y position
            1, 0,             // Width: 1 pixel
            1, 0,             // Height: 1 pixel
            0,              // Flag: not interlaced

            1,              // LZW codesize
            1,              // LZW data blocksize
            0x32,             // LZW code
            0,              // Block terminator

            ';'               // GIF terminator
         )
      }

      outFile.fwClose();
   }
```

</action-javascript>
</page-action>

**fwWriteShorts([num/array/bool..num/array/bool])**

*Arguments:*

one or number, Boolean or array objects

*Description:*

This method allows you to write binary data to a file. You can pass in a number of arguments that will be written. Arguments are written as follows:

1. **Numbers** - numbers are converted into integers. The two lowest bytes of the integer are written.
2. **Boolean** - 0x0001 is written if the value is true otherwise 0x0000 is written
3. **Null** - 0x0000 is written
4. **void** - 0x0000 is written
5. **Arrays** - the elements of the arrays are written recursively using `fwWriteShorts`
6. **Other objects** - nothing is written

> Note: If the property `fwReverseBytes` is set then the byte order of all the values written will be reversed.

**`fwWriteLongs([num/array/bool..num/array/bool])`**

*Arguments:*

one or number, Boolean or array objects

*Description:*

This method allows you to write binary data to a file. You can pass in a number of arguments that will be written. Arguments are written as follows:

1. **Numbers** - numbers are converted into integers. All four bytes of the integer are written.
2. **Boolean** - 0x00000001 is written if the value is true otherwise 0x00000000 is written
3. **Null** - 0x00000000 is written
4. **void** - 0x00000000 is written
5. **Arrays** - the elements of the arrays are written recursively using `fwWriteLongs`
6. **Other objects** - nothing is written

> Note: If the property `fwReverseBytes` is set then the byte order of all the values written will be reversed.

**`fwReadByte()`**

*Arguments:*

returns - an integer

*Description:*

This reads a single byte from the file.

**`fwReadShort()`**

*Arguments:*

returns - an integer

*Description:*

This reads a two bytes from a file.

> Note: If the property `fwReverseBytes` is set then the byte order of all the values read will be reversed.

**Method**

**`fwReadLong()`**

*Arguments:*

returns - an integer

*Description:*

This reads a four bytes from a file. The highest bit of the byte is discard - this is because integer values in this JavaScript implementation are not true four byte values. If you need this top bit you will have to use two calls to `fwReadShort` and some general trickery after that.

> Note: If the property `fwReverseBytes` is set then the byte order of all the values read will be reversed.

## Method

### `fwReadFlashInfo()`

*Arguments:*

None:

*Returns:*

An object with properties that are set from parsing a flash file.

*Description:*

This method will parse a Macromedia Flash file and return information about that file in an object. If it fails to parse the file it will return `null`. Parsing starts from the current read position. It is intended that the should the Actions writer want to make changes to a flash file then they should amend the properties of the object and call the `fwWriteFlash` to create a new flash file.

*Example:*

The following action will post the background colour of a flash file in an alert.

```
<item-action name="Flash Sample">
<action-button name="BG Colour" onclick="BGColour()">
<action-javascript>

   function BGColour()
   {
      inFile = new FWFile;

      if (inFile.fwLocateRead("Locate Flash File","SWFL"))
      {
         var info = inFile.fwReadFlashInfo();
         if (info)
           alert("Background Colour:\r",
                " red=",info.fwBgColor.red,"\r",
                " green=",info.fwBgColor.green,"\r",
                " blue=",info.fwBgColor.blue);
         else
           alert("Not a flash file");
      }
   }

</action-javascript>
</item-action>
```

## Method

### fwWriteFlash(flashInfoObject,FWFile)

*Arguments:*

- `flashInfoObject` - a Flash Info object that specifies the attributes of the flash file that should be replaced.
- `FWFile` - the source file that is read. The contents of this file will be written with the substutions specified in the flashInfoObject.

*Returns:*

`True/False` - was the file successfully written.

*Description:*

This method provides a means by which a new flash file can be generated from an existing one with certain changes made specified by the Flash Info

Object. This means that it is possible to generate a new copy of a flash file with some changes. In particular the Actions writer has control of:

a) The URLs embedded in a Flash Movie
b) The Background colour of the Flash Movie
c) If the movie is protected against editing (it is possible to set this but not to clear this attribute)
d) The portion of the Flash Movie that is visible (i.e. it's bounding box)

Details of these properties are given in the details about the Flash Info Object.

The method will start reading the flash file from the current position. This means if you have already read the file you will have to set the position back to '0' before you pass the file in.

*Example:*
The following Example will write a new copy of a flash file with a white background.

```
<item-action name="Flash Sample">
<action-button name="White BG Colour" onclick="WhiteBGColour()">
<action-javascript>

   function WhiteBGColour()
   {
      inFile = new FWFile;

      if (inFile.fwLocateRead("Locate Flash File","SWFL"))
      {
         var info = inFile.fwReadFlashInfo();
         if (info)
         {
            // set the BG to be white
            info.fwBgColor.red = 255;
            info.fwBgColor.green = 255;
            info.fwBgColor.blue = 255;

            var newFileName = "white "+inFile.toString();
            outFile = new FWFile;
            if (outFile.fwLocateWrite("New
File",newFileName,"SWFL","SWF2"))
            {
               // reset the file position
               inFile.fwPosition = 0;

               // replace the urls in the inFile writing them to
the outFile
               outFile.fwWriteFlash(info, inFile);

               // close both files
               outFile.fwClose();
               inFile.fwClose();
            }
         }
      }
   }

</action-javascript>
```

```
</item-action>
```

**Property**

**fwPathName**

*Description:*

This returns the full path of a file.

**Property**

**fwReverseBytes**

*Description:*

This allows you to set and clear the reversing of the byte order. You will need to do this if reading PC formated binary files as the PC byte order is different than on a Macintoish. This affects the methods:

`fwWriteShort`
`fwWriteLong`
`fwReadShort`
`fwReadLong`

## Flash Info Object

This Object is a normal JavaScript object in which a number of properties are specified. This object is returned by the `fwReadFlashInfo` of `FWFile`. And is passed in to the `fwWriteFlash` method of `FWFile`.

**Property**

**fwUrls**

*Description:*

This is an array of strings that specifies URLs and their Targets. It is formatted in pairs so that the first string is a URL, the second is it's target.

**Property**

**fwTop**

*Description:*

The top co-ordinate of the bounding box of the flash movie. These co-ordinates are 20 times the co-ordinates in Freeway.

**Property**

**`fwBottom`**

*Description:*

The bottom co-ordinate of the bounding box of the flash movie. These co-ordinates are 20 times the co-ordinates in Freeway.

**Property**

**`fwLeft`**

*Description:*

The left co-ordinate of the bounding box of the flash movie. These co-ordinates are 20 times the co-ordinates in Freeway.

**Property**

**`fwRight`**

*Description:*

The right of the bounding box of the flash movie. These co-ordinates are 20 times the co-ordinates in Freeway.

**Property**

**`fwBgColor`**

*Description:*

The background colour of the Flash Movie. This is an object with properties "red", "green" and "blue". These properties specify the red green and blue components of the colour with '0' being the minimum and '255' the maximum.

### Property

**`fwProtected`**

*Description:*

This defines of the Flash Movie is protected or not. Protected movies can not be opened in another authoring program.

## Class FWItem

### Method

**`fwTableCells()`**

*Arguments:*

none.

*Returns:*

An array of rows. containing an array of the items that appear in each row.

*Description:*

This function returns all the cells that appear in a table. The result is an array with an entry for each row in the table. Each of these rows is an array with an array of table cells, one for each column. The elements in the arrays are '0' based - so the first row will be row '0'. If the table contains spans (i.e. cells that span more than a single roe or column) then the table cell will be repeated in the table.

*Example:*

The following code will list all the cells in a table, with information about their spans, in an alert.

```
<item-action name="Table Test">
<action-button name="List Cells" onclick="ListCells()">
<action-javascript>

   function ListCells()
   {
      var tableCells = fwItem.fwTableCells();
      if (tableCells)
      {
         var s ="";
         for (var iRow in tableCells)
         {
           for (var iColumn in tableCells[iRow])
           {
             cell = tableCells[iRow][iColumn];
             if (cell.fwRow == iRow && cell.fwColumn == iColumn)
             {
               s += iRow+","+iColumn;
               if (cell.fwColumnSpan > 1)
                 s += ', col span = '+cell.fwColumnSpan;
               if (cell.fwRowSpan > 1)
                 s += ', row span = '+cell.fwRowSpan;
               s += "\r";
             }
           }
         }

         alert(s);
      }
   }

</action-javascript>
```

## Property

### fwBgColor

*Description:*

The Fill colour of an item as a FWColor.

## Property

### fwName

*Description:*

If the item is a form item this is the 'name' that has been entered for the firm item by the user.

## Property

### fwText

*Description:*

If the item is a text field this is the text that has been entered by the user.

## Property

### fwValue

*Description:*

If the item is a form item this is the value that has been entered by the user.

## Property

### fwRows

*Description:*

This is only defined if the item is a table in which case it returns the number of rows in the table.

## Property

### fwColumns

*Description:*

This is only defined if the item is a table in which case it returns the number of columns in the table.

## Property

### fwBorder

*Description:*

This is only defined if the item is a table in which case it returns the width of the table border.

## Property

### fwCellPadding

*Description:*

This is only defined if the item is a table in which case it returns the table's cell padding.

## Property

### fwCellSpacing

*Description:*

This is only defined if the item is a table in which case it returns the table's cell spacing.

## Property

### fwRow

*Description:*

This is only defined if the item is a table cell in which case it returns the row that the table cell appears in. The row is '0' based so all row numbers start from '0'.

## Property

### fwColumn

*Description:*

This is only defined if the item is a table cell in which case it returns the column that the table cell appears in. The column is '0' based so all column numbers start from '0'.

## Property

### fwRowSpan

*Description:*

This is only defined if the item is a table cell in which case it returns number of rows that the table cell spans.

## Property

### fwColumnSpan

*Description:*

This is only defined if the item is a table cell in which case it returns number of columns that the table cell spans.

## Property

### fwExportType

*Description:*

This is the export-type of the item. It has the following values:

0 Other
1 HTML Item (including Table & Table Cell)
2 Export as GIF file(s)
3 Export as JPEG file(s)
4 Export as URL item
5 Export as PNG file(s)
6 Map Areathe item is an image map
101 Checkbox (Form Item)
102 Radio Button (Form Item)
103 Text Field (Form Item)
104 List/Popup (Form Item)
105 Text Area(Form Item)
106 Java Applet
107 PlugIn
108 Passthrough Graphic
109 QuickTime
110 Flash
111 Text
112 Markup Item

*Example:*

The following action will return the type of the item onto which it is applied in an alert.

```
<item-action name="Test Export Type">
<action-button name="Export Type"
onclick="alert(MakeMessage(fwItem.fwExportType))">
<action-javascript>

function MakeMessage(item)
{
    return ExportType2String(fwItem.fwExportType)
       +  ' (' + fwItem.fwExportType+ ')';
}

function ExportType2String(item)
{
    switch (item)
    {
       case 0 : return "Other";
       case 1 : return "HTML";
       case 2 : return "GIF";
       case 3 : return "JPEG";
       case 4 : return "URL";
       case 5 : return "PNG";
       case 6 : return "Map Area";

       case 100 : return "Checkbox";
       case 101 : return "Radio";
       case 102 : return "Button";
       case 103 : return "Text Field";
       case 104 : return "List";
       case 105 : return "Text Area";
       case 106 : return "Java Class";
       case 107 : return "PlugIn";
       case 108 : return "External Graphic [passthrough]";
       case 109 : return "QuickTime";
       case 110 : return "Flash";
       case 111 : return "Text";
       case 112 : return "Markup Item";
    }
}
</action-javascript>
</item-action>
```

## Class FWPage

This class represents a page within the Freeway document.

### Property

#### `fwTarget`

*Description:*

The base target specified for a page.

## Class FWAttribute

This class represents an attribute of a `FWTag`. Consider the following snippit of HTML.

```
<img src="Resources/_clear.gif" border=0 width=50 height=1
alt="">
```

In the tag tree this would be represented as one `FWTag`. The attributes of the tag:

```
src="Resources/_clear.gif"
border=0
width=50
height=1
alt=""
```

Are each represented as a `FWAttribute`.

When you add a an attribute to a tag (in JavaScript you do this by just adding a property)

```
myTag.onclick='"alert(1);'";
```

a new `FWAttribute` will be created.

### Property

#### `fwValue`

*Description:*

This is the text value of the attribute.

## Property

### `fwOwner`

*Description:*

This is the object that is responsible for creating this attribute. Generally it is `NULL`, however Freeway currently defines it for `href` and `color` attributes.

*Example:*

The following action will post an alert with the links that are defined on a page.

```
<page-action name="Test Owner">
<action-javascript>

    function fwAfterEndBody()
    {
        var bodyTag = fwDocument.fwTags.fwFind("body");
        var aTags = bodyTag.fwFindAll("a");
        var message = new Array;

        if (aTags.length)
        {
           message.push("LINKS");
           for (var i in aTags)
           {
              var a = aTags[i];
              href = a.href;
              message.push(href.fwValue+" - "+href.fwOwner);
           }
        }
        aTags = bodyTag.fwFindAll("area");
        if (aTags.length)
        {
           message.push("MAP AREAS");
           for (var i in aTags)
           {
              var a = aTags[i];
              href = a.href;
              message.push(href.fwValue+" - "+href.fwOwner);
           }
        }

        if (message.length)
           alert(message.join("\r"));
        else
           alert("no links");
    }

</action-javascript>
</page-action>
```

## Class FWOutput (additional information)

### Property

**fwIsRaw**

*Description:*

This retirns if a particular tag is raw text data.

**Method**

**fwFindEnclosing([tag/tagarray][,name[,value]])**

*Arguments:*

this can now take an array of tags in which case it will look for the first of any of the tags in that array.

**Method**

**fwFindEnclosing([tag/tagarray][,name[,value]])**

*Arguments:*

this can now take an array of tags in which case it will look for the first of any of the tags in that array.

**Method**

**fwFind([owner][after tag][type/typearray [,name [,value]]])**

*Arguments:*

This can now take an array of tag types in which case it will look for the first of any of the tags in that array.

**Method**

**fwFindAll([owner][after tag][type/typearray [,name [,value]]])**

*Arguments:*

This can now take an array of tag types in which case it will look for all of any of the tags in that array.

### Method

**fwAddCSSStyles([,after tag][string])**

*Arguments:*

- after tag - the tag to add this after.
- string - the name if the script tag to add

*Returns:*

A `<style>` tag

*Description:*

This is used to add a CSS style definition to the HTML output. You optionally specify the style type but by default it is `"text/css"`. Freeway will look for a tag of the type you request. If it finds one it will return that. If it does not it will create one. This method is very useful for adding additional CSS style definitions to your page. In addition to creating the tag freeway will correctly enclose it in the correct hiding mechanism to prevent early browsers that do not understand CSS style definitions from having problems.

*Example:*

The following function will add the style definition:

```
#divText { position: absolute; top:0; left:0 }
```
creating the tags

```
<style type="text/css">
<!--
-->
</style>
```

If they are not there

```
   function fwAfterEndBody()
   {
      var head = fwDocument.fwTags.fwFind("head",fwItem);
      if (head)
      {
         var cssStyles = head.fwAddCSSStyles();
         cssStyles.fwAddRawln("#divText { position: absolute;
top:0; left:0 }");
      }
   }
```

## Method (additional information)

**fwAdd(name/htmltag[,after][,hasClose])**

*Arguments:*

after - this is the content object that it should be placed after. If there is no content object then it will be appended. If the same content object as the method object is passed then the new tag will be inserted at the head of the list.

*Example:*

```
var tag = myTag.fwAdd("option",myTag);
```

Will add a new "option" tag as the first tag within the myTag.

## Method (additional information)

**fwAddRaw([text1][,after][,text2])**

*Arguments:*

`after` - this is the content object that it should be placed after. If there is no content object then it will be appended. If the same content object as the method object is passed then the raw text will be inserted at the head of the list.

## Class FWListData (additional information)

### Method

**`fwNameAt(index)`**

*Arguments:*

`index` - the zero-based index of the name.

*Description:*

The `FWListData` is used to represent name value pairs. This enables yuou to retrieve the name of a given item in the list.

*Example:*

The following action will list all the user-tags when you click on the button. It uses `fwNameAt` to get the name of the tag.

```
<page-action name="User Meta Tags Test">
<action-button name="User Meta Tags" onclick="listTags()"/>
<action-javascript>

   function listTags()
   {
      var result = new Array;

      // get the list of tags
      var userTags = fwPage.fwUserMetaTags;

      // go through all items in the list
      for (var i in userTags)
      {
         var tagName = userTags.fwNameAt(i);
         var tagValue = userTags[i];

         alert(tagName, '=', tagValue);
      }
   }

</action-javascript>
</page-action>
```

## Method

### fwFindAll(string)

*Arguments:*

`string` - the name of the tag to find.

*Description:*

This will return an array of all the tag values that have this particular name.

*Example:*

The following action will list all the tag values of a particular name, which is by default the GENERATOR.

```
<page-action name="Get Meta Tags Test">
<action-text name="tag" default="GENERATOR" />
<action-button name="Get" onclick="listTags()"/>
<action-javascript>

    function listTags()
    {
        // get the list of tags
        var userTags = fwPage.fwUserMetaTags;

        var tagList = userTags.fwFindAll(fwParameters["tag"]);

        alert(tagList.join("\r"));
    }

</action-javascript>
</page-action>
```

Please note the following disclaimer specifically with respect to this document:

SOFTPRESS SYSTEMS LIMITED LICENSOR(S) MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. SOFTPRESS SYSTEMS LIMITED LICENSOR(S) DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.
IN NO EVENT WILL SOFTPRESS SYSTEMS LIMITED OR ITS DEVELOPERS, DIRECTORS, OFFICERS, EMPLOYEES OR AFFILIATES BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL, OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR ACCOMPANYING WRITTEN MATERIALS, EVEN IF SOFTPRESS OR AN AUTHORISED SOFTPRESS REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.