Writing Freeway Actions:

1. Basic Actions

Acknowledgements

We gratefully acknowledge Tim Plumb's contribution to the preparation of this document. You may visit Tim's independent Freeway Actions Web site at

www.freewayactions.com

Here you'll find a circus of Actions that are both useful and fun. The Actions are available as both shareware and freeware. Additional resources are detailed in the 'Resources' section on page 45.

The Purpose of this Document

This purpose of this document is to enable you, a user of Freeway, to write your own Actions from scratch. As you probably know, Freeway Actions enable you to augment your Web pages with a wealth of available features. If you haven't yet used Actions with Freeway, we recommend that you do! Once you've used pre-existing Actions, the next step is to write your own.

For a comprehensive introduction to using Freeway Actions, take a look at the 'Actions' chapter in the *Freeway User Guide*. We won't go into detail here about using Actions, but in order to show you just how simple it is to use Actions, we'll introduce you to what they are and how to use them.

Then we'll take an in-depth look at how Actions work, and guide you through writing and implementing your own.

The Benefits of using Actions

From time to time it may be necessary to supplement, extent or modify the HTML output from Freeway. By using a Freeway Action, it is possible to make Freeway insert script into all the required places of the HTML output

using instructions held in one text file. The Action's script allows the author to specify exactly where in the HTML output any HTML script (or another kind of script – JavaScript) should be inserted.

This helps the author as the Action is held in a single location. Because the Action inserts script into a number of locations, there's no need for this to be done manually. If the script needs updating, the author edits the Action text file, and republishes the documents that use the Action.

What are Actions?

Actions are stored in text files

A Freeway Action is stored in a text file. It comprises the following components:

- HTML or JavaScript to be added to the HTML which Freeway outputs when you preview or publish your Freeway document.
- 'Action script' that determines *where* the HTML or JavaScript to be added is inserted.
- A definition of the variables used when the Action is executed. If an Action asks you (via the **Actions** palette) to input various parameters that determine its effects, these parameters are stored in variables.
- Script that controls how options appear in the **Actions** palette.

As you design your Web pages in Freeway, you call upon the Action in certain areas of your pages.

Actions are bounded by specific tags

In the Action script, the Action is bounded by `<action-html>` and `</action-html>` tags. The script to be added to the output lies between these two (outer) tags. If segments of this script have to be inserted into different parts of the HTML output by Freeway, they are enclosed within other `<action-html>` and `</action-html>` tags. The script contains some *identifiers* – script which defines where in Freeway's output HTML the Action is to insert additional script. For example, the following script

```
<action-html before-end-head> Some Text </action-html>
```

inserts the words ‘Some Text’ just before the end of the header of the HTML file as it is published.

The `<action-html>` tag includes the name of the Action. This appears in the **Insert** or **Item** options of the Main Menu. This tag is followed by script that displays options on the **Actions** palette.

Action script is similar to HTML—for example, tags are enclosed in angle brackets.

Using Actions

For comprehensive details on how to use Actions, read the ‘Actions’ chapter in the *Freeway User Guide*. Here we’ll offer the most basic instructions:

Copying Actions to the Freeway Actions folder

First copy the Action file into your ‘Freeway Actions’ folder, or a subfolder of this. For example, the Action **Slave Image** is located in the **Slave** subfolder, which holds all Actions of this type (Actions which invoke a change in an object when another object is changed).

We should distinguish here between two processes:

- Adding Actions to the ‘Freeway Actions’ folder (or subfolders) where Freeway can access them.
- Referring to Actions from within your Freeway document, which needs be done whenever the Action’s effects are to be applied.

You may load a Freeway document into Freeway running on a computer other than that used to create the document, which does not have a copy of those Actions that are referred to in that document. If you make changes to the document then preview or publish it, the Actions will work, because the Freeway document automatically makes and keeps a copy of any Actions it refers to.

Applying Actions to your document

There are three categories of Actions:

1. Item Actions, applied to an item on the page via the Actions entry on the Item drop-down menu on the Main Menu (**Item > Actions**).

2. Free-standing Actions, applied to the page via the **Sketch Action** tool in the toolbox.

3. Page Actions, applied through **Page > Actions**.

Click the Action to be applied from the list of Actions presented. The name of the Action is displayed in the top corner of the item (1) , sketched-out Action (2), or page (3).

Basic Actions and Advanced Actions

We recommend that you try writing basic Actions similar to those covered in this document before moving on to the advanced Actions covered in the second part of this document: 'Writing Freeway Actions: 2. Advanced Actions'. Once you're familiar with how the basic Actions presented in this document work, and have tried and implemented some of your own basic Actions, then you may wish to attempt to write advanced Actions that take full advantage of Freeway 3's Action capabilities. There are important differences in the way basic and advanced Actions modify pages in your document before they are output as HTML files, which are summarised below.

How Actions modify HTML

When a document is published or previewed, Freeway generates an HTML file for every page in your document. Once the HTML has been generated for a page by Freeway, but before it is output as an HTML file, the Actions that you have added to that page are applied to it. After all the Actions referred to in the page have been applied to it, the HTML file is generated.

When you come to work with advanced Actions, you'll appreciate that there are more stages to the process of generating an HTML file from a page in the document. In Freeway 3, a page in a document is converted into a hierarchical list of tags – a *tag tree*. The tag tree then interacts with any Actions that are referred to on the page. When all the Actions referred to on the page have been applied to the tag tree, an HTML file is generated from the tag tree. Don't worry about exactly what a tag tree is, just be aware that

it is an intermediate form of a page in your document before it is compiled into an HTML file.

Using more than one Action on an item or page

A basic Action adds HTML script to the HTML for a page after the HTML for that page has been generated by Freeway. Once this has been done, it may be difficult for another basic Action to modify the already-modified HTML script. This is because the Action simply looks for tags and inserts script – it is not ‘aware’ that the HTML text has already been altered. Because of this there are limitations on the number of Actions that may be run for a given object. Most commonly, only one Action may be applied to one object. Similarly in the case of page Actions, only one Action may be applied to any given page with complete confidence.

Advanced Actions are programs (written in JavaScript). The tag tree is the input for the program. A modifications to the tag tree represent the program’s output. The tag tree may be modified again and again by other advanced Actions. Once the tag tree has been modified by all the Actions referred to on the page, HTML is generated from it. Unlike basic Actions, there is no limitation on the number of different item Actions that may be referred to by a particular item, and similarly one page may refer to more than one page Action.

Creating a simple basic Action

Saving your new Actions

When you write a basic Action, you create a text file containing Action script and save it in the "Freeway Actions" folder. Once in this folder, Freeway can see it, and offer the Action name as an option on one of the Action menus. The menu the Action name is shown on depends upon which type of Action it is. Freeway knows this from the Action code – you do not need to specify if an Action that you have written is an item Action, a free-standing Action or a page Action. In summary:

- Item Actions: Go to **Item > Actions**.
- Free-standing Actions: Click-and-drag with the **Sketch Action** tool in the toolbox.

- Page Actions: Go to **Page > Actions**.

Freeway checks the Freeway Actions folder (and subfolders) whenever it is launched or whenever it is switched into from another application. Note that if an existing Action in the Freeway Actions folder or subfolders has been modified, then the modified Action is applied to the document the next time it is previewed or compiled.

Writing your first Action

This is the script for a basic free-standing Action. When the page it is applied to is previewed, it shows the words "Hello world" on the page where the Action was applied:

```
<action name="Print Hello world">
<P>Hello World</P>
</action>
```

1. Launch a text editor, such as *Simple Text* or *BB Edit*, and start a new file. Enter the text above – and no other text.

2. Save the file in the **Freeway Actions** folder as “My Sample Actions” (without a file extension). Although you can call your Action file anything you like it’s always a good idea to give it the same (or at least a similar) name to the Action itself.

3. Either launch Freeway, or switch into Freeway from the text editor – note that you don't need to close “My Sample Actions” in the text editor in order to use it in Freeway, although you need to save any changes for them to be applied with the Action.

4. Start a new Freeway document (use the default size offered).

5. Move the pointer over the **Freeway Action** tool in the **Tools** palette and click-and-hold the mouse button. A pop-up menu appears listing the free-standing Actions available.

6. Click over the item "Print Hello world" from the pop-up menu.

7. Position the pointer anywhere on the page, and drag out a rectangle (as you would with the Graphic Item tool). On releasing the mouse button an ‘Action square’ is drawn, identified by the Freeway symbol in the top-right of the square.

8. Preview the page in a browser. You should see the words "Hello world" shown in the browser window, at the position you drew the Action item on the page.

9. Next, switch back into your text editor and change the text between the <P> and </P> tags, save the text file, switch back into Freeway and preview the page again. You should see the new text displayed on the page.

You have just created and used your first Freeway Action.

Adding code to different locations

Here's a more sophisticated Action. When your Freeway document is previewed or published it inserts the author's name at the head of the HTML script as a *meta tag*, and adds the text "Copyright (c) 2002, Your name here" where you drew Action item on the page.

1. In the text file “My Sample Actions”, insert a couple of blank lines beneath the text you added for the previous Action, then type or copy and paste the text below into it.

```
<action name="Sign the page">
  <action-html before-end-head>
        <META NAME="Author" CONTENT="Your name here">
  </action-html>
        <P><FONT SIZE="-2">Copyright (c) 200X Your name here
        </FONT></P>
  </action>
```

2. Save “My Sample Actions”.

3. Launch Freeway, or switch back into Freeway from the text editor.

4. Click-and-hold on the **Freeway Action** tool in the **Tools** palette until the pop-up menu appears.

5. Select the menu item **Sign the page**.

6. Click-and-drag an Action box anywhere on the page.

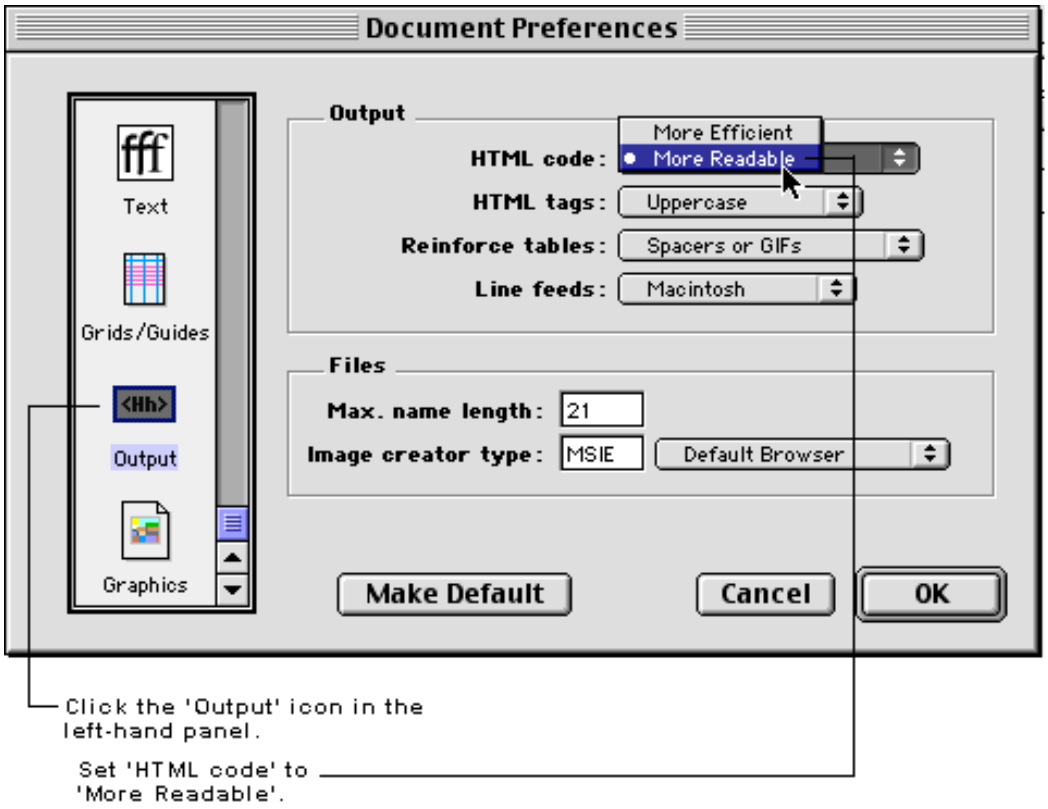
7. Preview the page. You should see the words "Copyright (c) 200X Your name here" (substituted with the appropriate information) appear on the page.

8. View the source code in your browser: In *Explorer* go to **View > Source**; in *Communicator* go to **View > Page Source**.

   You should be able to locate this text:

   ```
   <META NAME="Author" CONTENT="Your name here">
   ```

   You can make the code easier to read here: In Freeway, go to **File > Preferences**. In the 'Document Preferences' dialog, click the **Output** icon in the left-hand panel. In the **Output** cluster (see below), set **HTML code** to **More readable**, and click **OK**.



To be of use this Action needs to be placed on the master page of your document.

Note that when you need to change the copyright message, the formatting, or the meta tag, you simply edit the Action text file, then open and re-publish the documents that use it.

Using pre-defined properties

Freeway offers a selection of pre-defined properties for use in Actions. When the Action script refers to a property, the value of that property is inserted into the page when it is previewed or published. These properties include:

- `width:` The width of the item (to which the Action is applied), in points.
- `height:` The height of the item (to which the Action is applied), in points.
- `date:` the current date, in the format set up in the Control Panel.
- `time:` the current time, in the format set up in the Control Panel.
- `title:` the title of the item (to which the Action is applied).
- `link:` if the item is linked, this provides the URL (address) of the link.
- `pagetitle:` the title of the page as defined between the <TITLE> and </TITLE> tags.
- `pagefile:` the name of the file that holds the page (to which the Action is applied).

To call the value of a property in the Action code, enter the variable name in the form:

`&variablename;`

The above piece of code is entirely replaced by the contents of the variable when your Freeway document is published.

Here's a simple example of a couple of the built in variables in use:

```
<action name="Tell me about the page">
      <P><H3>Hello! This page is titled <I>"&_pagetitle;"</I>
      and was last published at &_time; on &_date;. </H3></P>
</action>
```

1. Add the script as shown above to the text file. Add a couple of blank lines between this Action and the previous Action, ‘Adding code to different locations’.

2. Save the changes to the text file.

3. Launch Freeway or switch back into Freeway.

4. Click-and-hold over the **Freeway Action** tool in the **Tools** palette until the pop-up menu appears.

5. Choose the item **Tell me about the page** from the menu.

6. Draw a box in Freeway using this Action.

7. Preview the page in a browser.

You should see a sentence that describes the page. This technique could be added to the previous example to create an Action which added the copyright information as well as the date and time the page was last published by Freeway.

Adding user input for Actions via Freeway's Actions palette

Using the Action from 'Adding code to different locations' (above), we're now going to examine how Actions can ask for user input via the **Actions** palette – i.e. at object creation time. We'll create two text fields in the **Actions** palette which allow the user to enter the name to be added for the Author meta tag, and the year and name for the copyright text displayed. Variables representing these fields are added to the output code, and will be substituted by the user's input when the page is published.

When you add an <action-text> tag, an editable text field with the given name appears in the **Actions** palette, and the contents of the field are represented by a variable with the same name.

Here's the Action:

```
<action name="Sign the page - 2">
<action-text name="Author">
<action-text name="Year">
<action-html before-end-head>
      <META NAME="Author" CONTENT="&Author;">
</action-html>
      <P><FONT SIZE="-2">Copyright (c) &Year;
      &Author;</FONT></P>
</action>
```

1. Add the code as shown above to the text file leaving another couple of blank lines between this code and that of the previous Action.

2. Save the changes to the text file.

3. Launch or switch back into Freeway.

4. Click-and-hold over the **Freeway Action** tool in the **Tools** palette until the pop-up menu appears.

5. Choose the item "Sign the page - 2" from the menu.

6. Draw a box in Freeway using this Action, and keep it selected.

7. In the **Actions** palette you'll see the two text fields, **Author** and **Year**.

8. Type your name and a year into the two fields.

9. Preview the page in a browser.

   You should see the year and name you entered displayed in the browser window, where the box was drawn.

10. Now view the source of the page in the browser, and look in the header of the HTML file—you should see the meta tag as defined in the <action-html before-end-head> section of the Action, but with the name you specified in the **Author** text field.

There are various ways of giving the user options—the default is the editable text field as used in this example, but you can also create pop-up menus with your own items in them, or create fields that allow the user to browse for files, and enter an appropriate file into the field.

Adding pop-ups and browse options

In this example, we're going to see how we could use an Action to allow the user to specify a QuickTime movie for output, and set up various options for it:

```
<action name="QuickTime">
<action-file name="Movie File">
      <value type="MooV">
</action-file>
<action-pop-up name="Loop">
      <value name="True">
      <value name="False" default>
</action-pop-up>
<action-pop-up name="Auto Play">
      <value name="True" default>
      <value name="False">
</action-pop-up>
<action-pop-up name="Controller">
      <value name="True">
      <value name="False">
</action-pop-up>
      <embed src="&Movie File;" width="&_width;"
      height="&_height;" loop="&Loop;" autoplay="&Auto Play;"
      controller="&Controller;">
</action>
```

There are two types of fields available for the user to enter data: *file* and *pop-up*.

The **file** option creates a pop-up menu with two choices on it, "None" and "Select...". Choosing **Select** displays a dialog with which the user can locate and choose a file. By default, this option would allow the user to choose *any* type of file. It is possible to restrict the available choices by specifying the Macintosh file types which should be displayed in the file selector dialog—in the above example, the file type of ‘MooV’ is specified, and only files with this type will be displayed.

The **pop-up** option creates a pop-up menu with the choices defined in the value tags between the <action-popup> and </action-popup> tags. Each choice is defined by its own value tag and can (optionally) contain the ‘default’ attribute. Menu items marked in this fashion will, as the name suggests, display the default item on the menu when shown on the Actions pallet. Only one item per menu should be defined in this way. Without a

default attribute the menu will display the first item defined in the list of values.

In the above example, the built-in width and height variables have been used to set the output dimensions of the QuickTime movie window to the same as the width and height of the object on the page.

Summary

In the previous examples, we've seen how Actions offer a convenient and powerful way to add custom HTML or scripting to pages created in Freeway. Because they are text files, it is easy to customize and experiment with them, and with a basic knowledge of HTML or JavaScript there should be no problems creating your own real-world solutions.

> Note
> You may use basic Actions to add JavaScript to Freeway's HTML output – even though basic Actions are not written in JavaScript and are not programs themselves.

Some References

To finish this introduction, here is a list of the tags which can be used in Actions (optional tag parameters are shown between the square brackets):

```
<action name=string [title=string] [region=regioncode
title=string …] [preview-type=checkbox | radio | button | textfield
| list | textarea | text] [preview-text=string] [width=number]
[height=number] [generates-form] [generates-link] [requires-form]>
</action>

<item-action name=string [title=string] [region=regioncode
title=string …] [generates-form] [generates-link] [requires-form]>
</item-action>

<page-action name=string [title=string] [region=regioncode
title=string …] [generates-form] [generates-link] [requires-form]>
</page-action>
```

`name:`
The name of the Action. If the name contains spaces the name needs to be enclosed in quotes.

`title:`
The title of the Action. If this is provided, it will be displayed within Freeway in place of the name. It can also be accompanied by a region code, to allow the setting of different titles for different localized versions of Freeway (see **region** below).

`region:`
A region code to accompany a title, which allows a different title to be displayed within the Freeway interface according to the localized version of Freeway which is running. A list of region codes is provided at the end of this section

`preview-type:`
Options are **checkbox**, **radio**, **button**, t**extfield**, **list**, **textarea** and **text** (default).

`preview-text:`
For use with preview-type=text. This specifies the text to appear on the layout item within Freeway. This can include variables (for example

```
preview-text=&_title
```

displays the title of the item on the Action).

`width/height:`
Width and height for the item within Freeway (always an integer).

`generates-form:`
The Action outputs a <form> tag. Freeway's automatic generation of the <form> tag (when there are form controls on the page) is suppressed if an Action is already handling it.

`generates-link:`
The Action outputs the <a> (link) tag. This is used to prevent Freeway from generating a second link tag if the Action already handles it internally.

`requires-form:`
The Action requires a <form> tag to be generated elsewhere. This would be used to ensure that Freeway automatically generates its default <form> tag when this Action is used, in the case when this Action will output a form element.

Examples:

```
<action name="my submit button" preview-type="button"
height=20 generates-form>
```

```
<action name="my name" preview-type="text" preview-
text="fred">
```

`version:`
This is a decimal number which will be displayed in the Actions palette (for example: 1.0)

```
<action-version version=decimal>
    [Some text]
</action-version><action-text name=string [title=string]
[region=regioncode title=string...] [default=string]
[display] [required] [var] [script=scriptcode]>
```

`name:`
Name of this text argument.

`title:`
Title of the argument. If this is provided, it will be displayed within Freeway in place of the name. It can also be accompanied by a region code, to allow the setting of different titles for different localized versions of Freeway (see ‘region’ below).

`region:`
A region code to accompany a title, which allows a different title to be displayed within the Freeway interface according to the localized version of Freeway which is running. A list of region codes are provided at the end of this document.

`default:`
The default value for this argument.

`display:`
Whether to display the value of this argument on the Action item on the Freeway page.

`required:`
This argument must be filled in. If not, a warning will be flagged when publishing.

`var:`
This argument is an internal field only – it will not appear in the Freeway interface.

`script:`
This can be used the specify the Macintosh script code (or keyboard) which should be used when typing into this field in the Actions palette. For example, if you want this field to always accept roman text even on a Japanese page, you would set the script to US. The default is for text to be entered according to the encoding of the page. A list of script codes is given below.

Example:

```
<action-text name="copyright" default="copyright Bilbo
Baggins 2002" display>

<action-number name=string [title=string] [region=regioncode
title=string...] [default=number] [minimum=number]
[maximum=number] [display] [required] [var]/>
```

`minimum:`
minimum value allowed.

`maximum:`
maximum value allowed.

Other parameters: see `<action-text>` above

Example:

```
<action-number name="counter" minimum=1 maximum=100>
<action-file name=string [title=string]
      [region=regioncode title=string...] [display]
      [required] [var] [keepwithhtml]>
      [<value type=file-type>]
</action-file>
```

`keepwithhtml:`
This parameter specifies that any non-HTML files that are generated on publishing are placed in the same folder as the HTML files, rather than the **Resources** folder, regardless of the choice made in the **Document Setup** dialog for the site. This is useful in the case of Java Applets where they will not function if they are placed in a different folder.

The <value type=file-type> settings allow you to specify a list of possible Macintosh file types to display in the file selector when choosing the file. For example, to specify that only GIF, JPEG and PNG files should be displayed in the file selector, use:

```
<value type=GIFf> <value type=JPEG> <value type=PNGf>.
```

Up to 4 types can be specified. If no types are specified then any file will be permitted.

```
<action-image name=string [title=string]
      [region=regioncode title=string...] [display]
      [required] [var] [keepwithhtml]>
      [<value type=file-type>]
</action-image>
```

This is similar to <action-file>, but in this case the file can also be generated using the **Graphic Parameters** view in the **Actions** palette, using graphic items on the Freeway page.

```
<action-pop-up name=string [title=string]
      [region=regioncode title=string...] [display]
      [required] [var]>
      <value name=string [value=string] [default]>
</action-pop-up>
```

`default:`
This is the default value of the pop-up.

The <value name=...> settings specify the items to be displayed in the pop-up within Freeway. The name appears in the pop-up. The value (if specified) is what will be substituted in the code where this parameter is used. If no value is specified, the name will be used. The value can contain other variables. They will be expanded before substitution occurs. In the simple example below, one parameter is used to specify a link, and a pop-up is used to determine whether or not to use that link.

```
<action-url name=mylink>
      <action-pop-up name=uselink>
      <value name=yes value='<A HREF="&mylink">'
      <value name=no value="">
      </action-pop-up>
</action-url>
<action-url name=string [title=string] [region=regioncode
title=string...] [display] [required] [var]>
```

This will generate a pop-up menu in the **Actions** palette, allowing a link to be set – either an internal or external hyperlink.

```
<action-checkbox name=string [title=string]
      [region=regioncode title=string...] [display]
      [required] [var]>
</action-checkbox>

<action-button name=string [title=string]
      [region=regioncode title=string...]
      [onclick=function()] [display]>
</action-button>
```

onclick:
A JavaScript function to execute when the button is clicked. This function must have been defined in an <action-javascript> tag somewhere within the Action file.

```
<action-color name=string [title=string]
      [region=regioncode title=string...]
      [display]>
</action-color>
```

This generates a color pop-up in the **Actions** palette, to allow selection from the custom or Internet colors, or creating a new color.

```
<action-label name=string [title=string]
      [region=regioncode title=string...]>
</action-label>
```

This doesn't actually allow user input, but displays text within the **Actions** palette.

```
<action! [any-text]/>
```

The contents of this are ignored by Freeway. It may be used to divide the controls that are shown in the **Actions** palette.

Example:

```
<action! This Action was written by Joe Bloggs/>
<action-html [position]> … </action-html>
```

Position can be one of the following:

`before-start-html`	`after-start-html`
`before-end-html`	`after-end-html`
`before-start-head`	`after-start-head`
`before-end-head`	`after-end-head`
`before-start-noframes`	`after-start-noframes`
`before-end-noframes`	`after-end-noframes`
`before-start-body`	`after-start-body`
`before-end-body`	`after-end-body`
`before-start-table`	`after-start-table`
`before-end-table`	`after-end-table`
`before-start-form`	`after-start-form`
`before-end-form`	`after-end-form`

`once-before-start-html`	`once-after-start-html`
`once-before-start-head`	`once-after-start-head`
`once-before-end-head`	`once-after-end-head`
`once-before-start-noframes`	`once-after-start-noframes`
`once-before-start-body`	`once-after-start-body`
`once-before-end-body`	`once-after-end-body`
`once-before-end-noframes`	`once-after-end-noframes`
`once-before-end-html`	`once-after-end-html`
`once-before-start-table`	`once-after-start-table`
`once-before-end-table`	`once-after-end-table`
`once-before-start-form`	`once-after-start-form`
`once-before-end-form`	`once-after-end-form`

`custom`
This code is not inserted automatically, but can be accessed using Freeway's JavaScript language to insert the code as required.

`<action-javascript>`
This is used to enclose JavaScript code used in the Action. Writing Actions using JavaScript is discussed in a separate document.

Built-in variables

The following built-in arguments can be used in Actions, and will be replaced with the relevant information on output:

`top`
Top coordinate of item on page.

`left`
Left coordinate of item on page.

`height`
Height of the item in Freeway.

`width`
Width of the item in Freeway.

`path`
Comma separated list of all coordinates of this item. This can be used as an image map.

`textarea.rows`
Number of rows occupied by textarea item.

`textarea.cols`
Number of columns occupied by textarea item.

`textfield.size`
Number of characters this textfield can display.

`select.size`
Number of lines to display for this menu/list item.

`date`
Date the page is published.

`time`
Time the page is published.

`title`
Title of the item in Freeway.

`layer`
Layer name of this item.

`link`
The link applied to this item (e.g. http://www.softpress.com).

`link.href`
The link applied to this item including "href=" or "nohref" and all extended attributes applied to the link.

`link.target`
The target applied to this link.

`link.extended`
All the extended attributes applied to this link.

`href`
Full HTML for link (e.g. <a href="http://www.softpress.com" target="_blank">)

`page.title`
Title of the page.

`page.file`
HTML filename of the page.

`cleargif`
Returns the relative path of the "_clear.gif" file.

`action.title`
The title of the Action.

`action.name`
The name of the Action.

`r`
Insert new line.

`gt`
Output “>” (similarly for other HTML escape characters).

The following options apply to the relevant multimedia types supported by Freeway, when an Action is applied (for example the QuickTime and Flash Extras).

`quicktime.autoplay`

`quicktime.controller`

`quicktime.loop`

`flash.loop`

`flash.quality`

The following can only be used within <action-html> in an <item-action> applied to a graphic item in Freeway:

`filename`
Image file name.

`alttext`
Alt text assigned to this image.

`mapname`
The map name.

Region codes for Action region setting

The region codes used for providing localized names for Actions and Action parameters are the standard Macintosh region codes. The localized name will be used when a localized version of Freeway is running – currently that only applies to the Japanese version, but here is a short list of possible values:

Language	**Code**	**Language**	**Code**
US English	0	Hebrew	13
French	1	Japanese	14
English	2	Australian	15
German	3	Arabic	16
Italian	4	Finnish	17
Dutch	5	Swiss French	18
Flemish	6	Swiss German	19
Swedish	7	Greek	20
Spanish	8	Icelandic	21
Danish	9	Maltese	22
Portuguese	10	Cyprian	23
French Canadian	11	Turkish	24
Norwegian	12		

Script codes for Action script setting

The script codes used in Freeway are standard Macintosh script codes. Here is a short list of possible values:

Script	Code
Roman	0
Japanese	1
Traditional Chinese	2
Korean	3
Arabic	4
Hebrew	5
Greek	6
Cyrillic	7
Simplified Chinese	8
Central European Roman	29
Turkish	35

Writing Freeway Actions:

2. Advanced Actions for Freeway 3

So far we have introduced the concept of Actions as pieces of HTML code that are inserted into the HTML that is output by Freeway. Just where HTML code is inserted depends upon how the Action is coded. However, this is a fairly blunt approach which doesn't exploit the potential of JavaScript to its full. If you've mastered the basic Actions explained in the first part of this document, 'Writing Freeway Actions: 1: Basic Actions for Freeway 2 and Freeway 3', you're now ready to adopt a new and more powerful approach to how you write Actions.

At the heart of Freeway 3 is the very same JavaScript 'engine' that drives modern Web browsers. Because of this, Actions can interact with a Freeway document at a very 'low level' and can, through a number of predefined terms, work with Freeway to query, add or remove content from the final pages. We'll now look at how you can control the interaction of Actions with your Freeway document as it generates HTML code.

The evolution of Actions

Let's think of Freeway's Actions technology as some kind of physical construction: This 'building' has three distinct floors. As you go higher, each level is defined by a distinct increase in sophistication, capability, and consequently, productivity.

The earliest 'Actions' are represented by the ground floor: these are not true Actions, but forerunners – which are called the 'include' (or '.inc') format.

Basic Actions

On the first floor we encounter Actions, introduced in Freeway 2. These Actions allow HTML code to be merged with the HTML code that Freeway generates, in a process not unlike that of the 'mail merging' of data. All the Actions that you have explored so far – in the first part of this document – are of this type. They are what we have termed 'basic Actions'. They run in both Freeway 2 and Freeway 3, but do not take full advantage of the ability

of Freeway 3 to allow Actions to interact with the *intermediate stages* of HTML code generation.

Advanced Actions

On the top floor we find advanced Actions, which will only run on Freeway 3. At the heart of Freeway 3 lies a JavaScript engine, and through the use of Actions, we can interrogate this engine to find out what code it is generating, and depending on what's being generated, Actions can make changes to the HTML output.

By using JavaScript in the Action code to generate output, there are two ways in which Actions modify the output that Freeway generates. We have already explored the first of these: that of adding HTML to the Freeway's HTML output. The second involves interacting with the intermediate stages of HTML generation from the Freeway document. Actions are able to do this, because Freeway 3 converts Freeway documents to HTML text via a number of distinct stages. The key intermediate stage is the generation of a *tag tree*. Advanced Actions then interact with this tree of HTML tags; after this, the tag tree is converted into HTML text.

Advanced Actions are programs, written in JavaScript, that run at specific stages in the generation of the tag tree. In running, they may generate HTML or JavaScript (not to be confused with the JavaScript which comprises the Actions themselves). The code generated by advanced Actions is then inserted into the tag tree as the tag tree is, stage by stage, converted into HTML.

This is quite difficult to conceptualise – the key point here is that once an advanced Action has interacted with the tag tree, the tag tree remains, essentially, so that further advanced Actions may interact with it. This gives tremendous versatility to the Actions, allowing them to modify each other's effects, and allowing a number of Actions to interact with the same object.

Tags and tag trees

Now let's look at exactly what we mean by 'tag tree'. You may already be aware that HTML code tells the browser how to format a page by the use of *tags*. We'll first look at tags.

Tags

HTML tags are often English words (such as ‘blockquote’) or abbreviations (such as "p" for paragraph), but are distinguished from other text as through being bounded by angle brackets. For example, the ‘paragraph’ tag is represented by <p>. The ‘blockquote’ tag is represented by <blockquote>. Some tags determine how the page is formatted when viewed on a browser; for example, <p> begins a new paragraph. Other tags determine how text is formatted; for example, <b> makes text bold. Some tags provide information to the browser that doesn't appear on the page itself; for example, the page title is bounded by <title> tags.

When Freeway generates HTML code from your Freeway document, it first creates these HTML tags. Each tag represents an element or characteristic of the page when the HTML is interpreted by a browser. It is these tags that are modified by advanced Actions. To see how Actions do this, one first needs to appreciate the concept of a *tag tree*.

Tag trees

Every Web page contains certain items, and possesses certain properties. such as a background, images, buttons and so on. These items or properties in turn contain their own items, or possess their own properties. The overall layout of the page is most clearly represented as a tree – a *tag tree*.

Freeway generates HTML code in the same way that you would generate HTML if you had to do it manually – if you were given an accurate diagram of the page to be converted into HTML. Freeway first makes a list of all the items to be created on the page. This list is the start of a *tag tree*, and comprises a list of pairs of tags, each pair of tags representing an item. Some items contain other items, in which case Freeway then inserts another pair of tags for each new item between the pair of tags that define the parent item. These pairs of tags may be thought of as branches on the tree.

Let’s build a tag tree for a background page has three buttons. The language of our tree isn’t HTML – it’s English – so it allows you to see what’s going on:

```
<:open tag for background page:>
  <:open tag for first button:>
  <:close tag for first button:>
  <:open tag for second button:>
  <:close tag for second button:>
  <:open tag for third button:>
  <:close tag for third button:>
<:close tag for background page:>
```

However, each button may have three images apportioned to it: one image for the button's 'resting' state, one for when the mouse pointer is rolled over it, and one for when the button is pressed:

```
<:open tag for background page:>
  <:open tag for first button:>
  <:open tag for first image:>
  <:close tag for first image:>
  <:open tag for second image:>
  <:close tag for second image:>
  <:open tag for third image:>
  <:close tag for third image:>
  <:open tag for first button:>
       <:open tag for first image:>
       <:close tag for first image:>
       <:open tag for second image:>
       <:close tag for second image :>
       <:open tag for third image:>
       <:close tag for third image:>
  <:close tag for first button:>
  <:open tag for third button
       <:open tag for first image:>
       <:close tag for first image:>
       <:open tag for second image:>
       <:close tag for second image:>
       <:open tag for third image:>
       <:close tag for third image:>
  <:close tag for third button:>
<:close tag for background page:>
```

Of course, this is extremely simple, but we hope that it conveys to you what is meant by a 'tag tree'.

How Freeway generates a tag tree

The first tags, in the above example, '< : open tag for background page, close tag for background page: >' , represent the first level of the *tag hierarchy*. In practice, these would be the items and properties of the background page. As the HTML is generated, more tags are added to certain 'branches' in the tree. This means that more tags are added between existing tags. Freeway populates all the way down to the end of each 'branch' as it writes each part of the tag structure – so the tag structure in our example above would be output by Freeway as '< : open tag for background page < : open tag for first button <:open tag for first image…etc.'

Freeway needs to do this so that it didn't have to go back after having written each hierarchy of tags in order add another hierarchy, then go back again to add another hierarchy.

Once Freeway 3 has assembled the complete tag tree, advanced Actions are able to question tag structures, and Action after Action can work on the same item. The output from one Action instantly becomes available to the next one to be launched, and as each Action is run on the tag tree, complex code structures can be built up, with the Actions effectively being the building blocks of this code.

How the tag tree is coded in Freeway

In the tag tree, HTML tags are represented by `'fwTags'`.

The presence `'fwTags'` in the tag tree indicates the presence of tags. As an Action writer we can directly manipulate these tags.

Actions act at different stages of HTML production from the tag tree

The generation of HTML from a tag tree is carried out over a number (over 20!) stages. At any one of these stages an Action can take control of the code generation.

Any tags that are present in the tree are available for inspection and modification by an Action; however, just *when* an Action runs on the tag tree is important: An Action will not see tags that have yet to be created, neither will it see tags that are represented as HTML.

Actions can only be viewed on publishing or previewing

Whenever the page is published or previewed the tag tree is generated from your Freeway document. It is at this point that Actions are able to act on the tags. This is why the effects of Actions can't be viewed on your Freeway document – the Actions have nothing to act upon, as the tags are simply not present before the tag tree is generated.

As was stated at the start of this section, A tag tree isn't generated from your Freeway document in one, simple stage – in fact, the tag tree is generated in more than twenty distinct stages, and Actions may act at each of these stages. The diagram below shows each of these stages, and where they may be interrupted by Actions:

Walking through a simple advanced Action

Let's have a look at a specific Action to see how the tag tree is modified by its presence.

In this example we've written a basic Action that replaces the object it is applied to with an HTML comment.

```
<item-action name="Replace image with comment">
<action-javascript>

function fwAtContent()
{
  fwDocument.fwWrite("<!-- Image replaced by the 'Replace image
with comment' action –->");
}

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

`<item-action name="Replace image with comment">`
tells Freeway that this is indeed an Action file and gives it a name. With this information Freeway can place the Action item correctly in your Freeway document

`<action-javascript>`
tells the JavaScript engine in Freeway to wake up and get ready for the next line.

`function fwAtContent() {`
Is the start of the main code, and it tells Freeway's JavaScript engine that the next bit of code:

```
fwDocument.fwWrite("<!-- Image replaced by
the 'Replace image with comment' action --
>");
```

should replace whatever the Action is applied to. This is quite powerful as we can use this to easily swap one item for another.

`fwDocument.fwWrite`
tells the application that the text in quotes needs to be written to the document.

`</action-javascript>`

tells Freeway that we are finished with this block of code.

`</item-action>`
tells the JavaScript engine and Freeway that the Action has finished so that they can go and get on with other matters.

First the Action has to identify the object that is to have its HTML output overridden. For example:

`fwDocument.fwWrite()`
This line of code, if present in an Action, will write data *without* adding a new line.

`fwDocument.fwWritelnopt()`
This line of code will write data followed by a new line if the **more readable** option within Freeway is turned on.

`fwDocument.fwWriteln()`
This line of code will write data followed by a new line *irrespective* of whether the **more readable** option within Freeway is turned on.

If the Action code calls one of the above methods then it will be able to add text to the output. Anything that you write using these methods will appear directly in the HTML file.

For example, this Action will add a 1-pt border to an image:

The code here may be broken down as follows:

Each time

```
fwDocument.fwWrite
```

is called, the location and width of the border to be applied is determined. The

```
fwAtContent
```

method will be called when the `<img>` tag is written.

`<item-action name="Image Border1">`
This tells the Freeway HTML compiler that it has encountered an Action – an *item Action* (one that is applied to an item in the document) – and that the item Action has the name ‘Image Border1’.

`<action-javascript>`
This tells then Freeway compiler to insert HTML code that tells the browser that it is about to encounter JavaScript.

`function fwAtContent()`
This line tells Freeway to replace the item’s code with the code we are about to supply.

`{fwDocument.fwWrite('<img');`
This line tells Freeway to write to the object **img**

Each of these lines concatenates parts of the final line of code together into their final form. We’ve split the lines in this example, but you could have all of this code in one single `fwWrite` statement.

```
fwDocument.fwWrite(' src="', fwFileName(), '"');
```
This line tells Freeway to write…for all occurrences in the Freeway document file as named in the closed brackets. Take the name of the image that the Action is applied to and use this as the source for our new image.

```
fwDocument.fwWrite(' border=1');
```
Set the borders that bound the object ‘<img’ to a width of one point.

```
fwDocument.fwWrite(' height=', fwHeight);
```
Set the height of the object ‘<img’ to the height specified in ‘height=’. Take the height of the item that the Action is applied to and use this value for the height attribute of the new image.

```
fwDocument.fwWrite(' width=', fwWidth);
```
Set the width of the object ‘<img’ to the width specified in ‘width=’.

```
fwDocument.fwWrite(' alt="', fwAltText(), '"');
```
Take the alt text of the item that the Action is applied to and use this value for the alt attribute of the new image.

```
fwDocument.fwWrite('>');
```
Add the closing angle bracket to the code.

```
fwDocument.fwWritelnopt();}
```
This adds a carriage return if the application preferences are set to more readable.

```
</action-javascript>
```
This terminates the JavaScript part of the Action.

```
</item-action>
```
Terminates the Action itself.

The above method, although quick and effective is generally regarded as a destructive method of generating output with an Action. This is because Freeway is unaware of the code that the Action writes to the tag tree. All Freeway is aware of is that code is added to the output stream when the document is published.

A better way of doing this is to specify the tags that are to be added to the tag tree. Writing Actions takes longer this way, but it is better as further Actions can alter these tags at any point in the publishing process. Because the tags are inserted in the tag tree at a ‘root level’ Freeway treats any extra code as if it was its own and therefore allows other Actions full access to it.

If we look at a simple HTML page we can start to break this down into items that our Action will ‘see’:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
	<head>
		<title>My sample HTML file</title>
	</head>
	<body>
		Hello world!
	</body>
</html>
```

Looking at the tags that make up the above page we can see how we can access any part of the code by searching for specific tags.

For example to extract the title of the page and place this information into a window that alerts the user of the title (i.e. an *alert*), we could write the following Action (presented here line-by-line):

```
<page-action name="Alert page name">
```

The above line tells Freeway that the following code is a page-based Action – an Action that can only be applied to whole pages – and specifies the name of the page-based Action.

```
<action-javascript>
```

The above line activates the JavaScript engine in Freeway.

```
function fwAfterEndHead(){
```

This tells Freeway where it should run this code.

In the above example the code runs directly after the application has written the closing HEAD tag.

As we are only after the TITLE tag in the head anyway we don't need to wait for the application to generate all of the page before we start working on the file. In general it's best to jump in as soon as you can to alter the code. Jump in too soon and the code won't have been created yet (and the Action will return nothing!). Jump in too late and you risk any corrections you make being missed by any other Actions applied to the page.

```
var myTitle = fwDocument.fwTags.fwFind("TITLE");
```

This line sets up a variable that holds a reference to the TITLE tag.

```
var myTitleText = myTitle.fwFindAllContent();
```

Again we define a variable and store the contents of the TITLE tag inside it.

```
alert(myTitleText);
```

Finally alert the user with the name of the page.

```
}
```

This line tells the application that we are done with our function.

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

This tells the application that we have now finished with the JavaScript engine inside Freeway.

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

This line closes the Action and tells Freeway that it can get on with other matters.

Each branch represents an object – and in HTML code this is represented as a tag. Note that some branches lead to further branches, just as some objects on the Web page contain other objects.

Attributes

If tags are the branches of the HTML tree then attributes are the leaves that bring character and form to the structure. Each attribute defines how a tag will display its data. For example the image tag which is represented by `<IMG>` in HTML. By adding the 'src' (source) attribute to it and giving the source attribute a value we get `<IMG SRC=″myimage.jpg″>`. When a Web browser is faced with this line of HTML it downloads the image and displays it.

Once further attributes of width and height are added to the IMG tag, the look and representation of this item is further defined.

How Actions work with tags

In order to modify the tags in a Freeway document, an Action must first locate which tags are to be modified. In the first part of this document we simply added to the Freeway's HTML output without looking for specific tags. However, as you require more of your Actions you'll code them so as to search for specific tags.

Part of the coding in most Actions searches the tag tree of Freeway's output for specific tags.

For example, this JavaScript fragment - part of an Action - will locate the HTML <body> tag so an *attribute* of the body tag – the background colour of the HTML page – can be modified.

```
<body bgcolor=”#ffffff”>
```

The above line of HTML sets the background colour (‘bgcolor’) of the page (‘body’) to ”#ffffff”. If an Action is to modify this, the ‘body’ tag needs to be located; once located, the attribute of the tag can be changed. The following line in an Action does all this:

```
var bodyTag = fwDocument.fwTags.fwFind("body")
```

First the search:

fwDocument: This means Let’s look at the current document.

fwTags: In the document, let’s look at the tag tree.

fwFind: In the tree let’s search for ‘body’.

Once the search is complete, the attributes of the <body> tag are set to whatever value the variable bodyTag holds.

Let’s look at another line from an Action:

```
var imageTag = fwDocument.fwTags.fwFind("img", fwItem)
```

This sets the attributes of the <img> tag to the value of the variable imageTag. The <img> tag affected is the first <img> tag that belongs to the item onto which the Action is applied.

Modifying the tags

Action code such as that described immediately above means that by setting the properties within the variables used in the code, you can set or clear the attributes – i.e. properties - of tags.

For example, the HTML tag <body bgcolor=”#ffffff”> has one attribute, bgcolor. The value held by the variable bodytag.bgcolor is thus “#ffffff”

Let’s look at another example: This Action will initially locate the correct tag. This is the <img> tag that belongs to the item where this Action is to be applied. Having located the correct tag, it then first sets the border to a thickness of 1 point.

```
// find the first <img> tag that belongs to item
// onto which this Action is applied
var imageTag = fwDocument.fwTags.fwFind("img", fwItem);
// and set the border to 1
imageTag.border=1;
```

Some attributes are not designed to have a value, such as the 'no resize' tag of a frame. If you wish to have an attribute in a tag that does not have a value, then you need to set the attribute (which must have the same name as the tag) to no characters at all – i.e. an empty *string*.

The following Action sets all the <frame> tags so that they have a 'no resize' attribute.

```
<frame src="top.html" name="top" scrolling=no noresize>
<page-action name="Make All NoResize">
<action-javascript>

function fwAfterEndBody()
{
  var frameTags=fwDocument.fwTags.fwFindAll("frame");
  for (var i in frameTags)
       frameTags[i].noresize="";
}

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

Let's dissect this Action:

<frame src="top.html" name="top" scrolling=no noresize>

Set the frame search variable 'frame src' to 'top.html'; the top component refers to the HTML tag <top>; the 'html' component means that we're searching the HTML text. The remainder of the line sets the 'name' variable to 'top'.

<page-action name="Make All NoResize">

This identifies the name of the Action.

<action-javascript>

This informs Freeway that the Action is in JavaScript.

```
function fwAfterEndBody()
```

This defines the function fwAfterEndBody() to be applied to the document. This function, like all functions is defined between the '{' and '}' brackets which follow.

```
{
var frameTags=fwDocument.fwTags.fwFindAll("frame");
```

This locates all instances of `<frame>` tags in your document, and sets them to the value of the 'frameTags' variable.

```
for (var i in frameTags)
frameTags[i].noresize="";}
```

This ensures that the function is applied for each occurrence of the of `<frame>` tags.

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

This is the closing tag for the Action – text following this is not a part of the Action.

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

This is the closing tags for the area of the page that holds all the Actions. Text following this is not a part of any Action.

In this example, for each image that is generated the Action will set the border for the item to 1.

```
<item-action name="Image Border1">
<action-javascript>

function fwAfterEndBody()
{
  // find all <img> tags that belong to the item
  // onto which this action is applied
  var imageTags=fwDocument.fwTags.fwFindAll("img", fwItem);

  // set the border to 1
  for (var i in imageTags)
       imageTags[i].border=1;
}

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

The example below sets the border tag of successive `<img>` tags on the page to an incrementing amount. That is, the first `<img>` will have a border of 1, the second `<img>` will have a border of 2, and so on.

```
<page-action name="Border++">
      <action-javascript>
            function fwAfterEndBody()
                  {// find all <img> tags on this page
                  var imageTags =
                  fwDocument.fwTags.fwFindAll("img");
                  var j = 1;
                  for (var i in imageTags)
                  // set the border
                  {imageTags[i].border=j;
                  // increase the border width
                  j++;}}
      </action-javascript>
</page-action>
```

Removing attributes of tags

If you wish to remove an attribute entirely you can do this by setting the property of the same name to `'null'` This will delete the attribute.

The following Action will remove the width and height attributes from all the `<img>` tags on the page.

```
<item-action name="Remove Width & Height">
<action-javascript>

function fwAfterEndBody()
{
  // find all <img> tags that belong to the item
  // onto which this action is applied
  var imageTags = fwDocument.fwTags.fwFindAll("img", fwItem);
  // remove the width and height
  for (var i in imageTags)
  {
       imageTags[i].width=null;
       imageTags[i].height=null;
  }
}

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

The following Action will remove the `‘align’` attribute from all the images that have been generated from the item to which the Action is applied.

```
<item-action name="Strip Align">
<action-javascript>

function fwAfterEndBody()
{
  var imgTags=fwDocument.fwTags.fwFindAll("img", fwItem);
  for (var i in imgTags)
  {
       var img = imgTags[i];
       img.align = null;
  }
}

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

Interaction of body tag attributes with item tags attributes

Sometimes it is useful to be able to affect the page tags from an Action applied to an item.

Example: This Action will set the page to have a background image that is the same as an image in a specified item.

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

function fwAfterEndBody()
{
  // find the body tag
  var bodyTag = fwDocument.fwTags.fwFind("body");

  // find the first image tag applied to this Action
  var imageTag = bodyTag.fwFind("img", fwItem);

  // set page background image to source file of the image
  bodyTag.background=imageTag.src;
}

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

Qualities of the tag tree

About the tag tree

The tag tree generated by Freeway 3 is a *partial* tree. This means that not *all* the tags that appear in the output can be searched for and found by `fwFind` or `fwFindAll`. For example if you call the method, `fwFind("img"),` Freeway will not return any of the `<img>` tags for `_ClearGifs`. `_ClearGifs` are the invisible GIF images Freeway uses for reinforcing tables in the layout. Generally these layout tables are not available as tags to the Actions writer. The reason is to prevent Actions from being able to destroy the layout tables which Freeway creates. Under some circumstances you may need to do things such as remove all the content of a page, but note that you can do this by deleting all the content in the body area. Similarly the tags that are associated with text, the formatting of text and the text itself are not accessible in the tag tree.

As an Actions writer it’s important that you’re aware that not all tags are available for modification.

Marker Tags

Whenever output from an item is generated, Freeway adds special empty tags as markers in the tag tree, to help you locate the output for that item. These are termed *marker tags*. Typically a given item on the Freeway page results in one or more table cells being generated. When Freeway

generates the code for the item it encloses the content in a marker tag. You can locate this tag using `fwFind()` in the normal way. These tags are not output as HTML.

The following example will generate an **alert** dialog showing the HTML that will be generated by the item to which the Action is applied.

```
<item-action name="Show Content">
<action-javascript>

function fwAfterEndHTML()
{
  itemContents = fwDocument.fwTags.fwFindAll("", fwItem);
  for (var i in itemContents)
       alert(itemContents[i].fwToHTML());
}

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

The example below finds the marker tag that encloses the code output for an item, converts the tag to text and then within that text substitutes any occurrence of the text "`_test_`" with the parameter "`MyText`" and outputs the processed text.

```
<item-action name="Replace Text">
<action-text name="My Text" default="Hobson">
<action-javascript>

function fwAfterEndHTML()
{
  firstTag = fwDocument.fwTags.fwFind("", fwItem);
  if (firstTag)
  {
       // find the enclosing tag
       enclosing = firstTag.fwEnclosing;

       // convert it to text
       var tagAsText = firstTag.fwToHTML();

       // replace anything with _test_
       tagAsText = tagAsText.replace(/_test_/g,fwParameters["My
Text"]);

       // insert it after "firstTag"
       enclosing.fwAddRaw(firstTag, tagAsText);

       // delete the original tag
       firstTag.fwDelete();
  }
}
</action-javascript>
</item-action>
```

Timing out

Freeway will automatically terminate the running of any Action code that becomes caught in an infinite loop, when the Freeway document which refers to the Action is previewed or published. It does this by maintaining an internal timer that monitors how long the Action code has been running. The amount of time that any item of Action code can run before it is terminated is determined by the global property `fwTimeout`. This time-out facility is included to prevent Freeway from locking up should it try to run an Action which has an infinite loop in its coding.

Each time a *user method* (such as `fwAtContent`) is called the timer is reset. So if your page has several copies of the same Action, the timer will be

reset each time the same user method is called, regardless of whether it has been called before.

The timer will be paused whenever user intervention is required (for instance to confirm an Alert dialog or to locate a file). The timer will also pause if you execute an OSA script (such as AppleScript).

In this example, the Action is written to loop endlessly when published. However when Freeway detects that the `fwAtContent` method has executed for longer than the period specified by `fwTimeout` , so Freeway will stop executing the Action code and display an error dialog. The Action definition will be marked as having an error.

```
<item-action name="Infinite Loop">
<action-javascript>

function fwAtContent()
{
  for (var i = 0; true; i++)
  {
       var b = i;
  }
}

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

To reset the Action after it has been marked as having an error, you will need to force Freeway to reload the Action definition, either by quitting or modifying the Action file.

Persistence

Persistence defines how long properties persist (or remain) after they have been defined.

None of the properties that you set as an Actions writer are persistent – once the Action code has executed, they no longer exist. You need to be aware that you can’t guarantee that properties you set will persist across the generation of different pages while publishing the site, or when publishing subsequently. Freeway will dispose of properties when it is required to free up memory. However, any properties required to generate the current page will be secure.

If you need to store values and have those values saved in the Freeway document, you can do this through internal Action parameters that are invisible to the user. You can ensure that a parameter is not visible by adding a `var` property.

In the following example, the parameter called `'Last Page'` will not be present in the Actions palette, but values can be stored by setting the `fwValue` property of the parameter using JavaScript:

```
<action-url var name="Last Page"/>
```

You may find it useful to remove the `var` property while developing your Action, so you can see the values that you are storing in the **Actions** palette.

Resources:

Additional resources for the Actions writer are:

- Writing your own Actions: Reference Guide. This is a detailed listing of each of the Freeway functions, what they do, with examples of their implementation.
- The Freewaytalk mailing list (to join visit http://www.softpress.com/Freewaytalk/)
- The Actions Developer mailing list (to join visit http://www.softpress.com/actions/
- Independent Freeway Actions site with lots of useful freeware or shareware Actions: http://www.Freewayactions.com