Actions Tech. Note 4

Draft 1.0

Understanding Action Parameters

Freeway 3.0 supports a JavaScript API for Freeway actions. Using this API you can access parameters that the user sets in the interface and use values drawn from the parameters in your code.

Basic Parameter Access

If in your code you have a parameter

```
<action-number name="PNum">
```

You can access your parameters within your action as normal parameters as you can any JavaScript parameter.

```
var pNum = fwParameters.PNum
```

and also as

```
var pNum = fwParameters["PNum"]
```

The later form being very useful if you have actions with spaces in the names. For example if you have a parameter:

```
<action-number name="Pig Num">
```

Accessing it as follows

```
var pNum = fwParameters["Pig Num"]
```

Is both readable and convenient - and represents no speed penalty in access.

Being able to access the parameter as a string means that you can construct the string yourself if you need to. So for an action with the following parameters.

```
<action-number name="Pig 1">
<action-number name="Pig 2">
<action-number name="Pig 3">
<action-number name="Pig 4">
<action-number name="Pig 5">
<action-number name="Pig 6">
```

You can construct a loop that loops through all of them as follows:

```
for (var i = 1 ; i <=6 ; i++)
{
  var pNum = fwParameters["Pig "+i];
  :
  :
}
```

The Parameter list.

All the parameters that belong to an action are available from the `fwParameters` property of the action. This property is a list of all the parameters and facilitates indexed access, as do all the lists within Freeway. The list is contiguous and zero-based, so the first element in the list is element '0', and there are no gaps between the first and last element in the list.

To access the first parameter you could write.

```
var pNum = fwParameters[0];
```

The `fwLength` property of the list is the total number of elements in the list. You should bear this in mind when writing actions. If you use this property it is best to avoid giving any action parameter the name of `fwLength` as it is sure to cause you problems and confusion. Using the `fwLength` property you could iterate through all parameters as follows:

```
for (var i = 0 ; i < fwLength ; i++)
{
  var param = fwParameters[i];
  :
  :
}
```

You can also use the JavaScript loop construct designed for this purpose.

```
for (var i in fwParameters)
{
  var param = fwParameters[i];
  :
  :
}
```

The Parameters

The items in `fwParameters` the parameter list are parameter objects they are not values. This distinction is important as it is quite easy to mistakenly get the idea that say:

```
fwParameters["foobar"];
```

will give you the value of whatever the user entered into the parameter "foobar". It will not it will give you the parameter object that contains whatever the user entered. The situation is slightly confused by the fact that JavaScript is very good at coercing values to strings. If you do something like:

```
alert(fwParameters["foobar"]);
```

The JavaScript will take the parameter object `fwParameters["foobar"]`, and then call it's `toString` method automatically as the `alert` method requires a string. In 99% of cases this will be what you want - however it is not always.

An easy point of confusion is the example where the user has entered nothing into the parameter foobar, and you want to test for it in your code. The code

```
if (fwParameters["foobar"])
  alert("has value");
else
  alert("No value");
```

Will always return "has value" this is because the line `fwParameters["foobar"]`, will return the parameter object, that object will never be null, or the empty string or zero, so the `alert("No value")` code will never be executed. In this case you will have to either execute the `toString` method yourself

```
if (fwParameters["foobar"].toString())
  alert("has value");
else
  alert("No value");
```

or compare it with the empty string, in which case JavaScript will call the `toString` method it's self in order to compare it to the empty string.

```
if (fwParameters["foobar"]=="")
  alert("has value");
else
  alert("No value");
```

Parameter Types

The parameter object can be one of four types, according to how it has been specified in the action.

fwText	from	<action-text>
	&	<action-number>
	&	<action-popup>
	&	<action-checkbox>
fwLink	from	<action-url>
fwFile	from	<action-file>
fwColor	from	<action-color>
fwImage	from	<action-image>

You can discover the type of a particular parameter from the `fwType` property. So, for example, the following code will return all the `fwLink` parameter objects

```
for (var i in fwParameters)
{
  param = fwParameters[i];

  if (param.fwType=="fwLink")
    alert(param.fwName)
}
```

The actual value set by the user is accessible from the parameters `fwValue` property. The type of this property varies according to the type of the parameter.

fwText	string
fwLink	FWLink object
fwColor	FWColor object
fwFile	calls toString()
fwImage	calls toString()

You can see that for `fwFile` and `fwImage` the `toString` method is called. This is because neither the `fwFile` or `fwImage` have objects so this method will just do the best that it can.

Text Parameters

The parameters of `fwText` behave much as you would expect. If you want to get the value of the parameter the `fwValue` will return the text, and you can set it by setting the value. So if you wanted for a parameter to have the value of "goat" you would do this.

```
fwParameters["foobar"]="goat";
```

The user interface would then reflect this next time the action interface is updated, and this parameter will be saved with the document.

Link Parameters

These parameters represent URLs. The `fwValue` will return a `FWLink` object. You can not change the object that the `fwValue` will return, but you can change the values within this object - this means, for example, you can change the parameter so that it will link to a particular page. Consider the following code fragment - it will change the link so that it is linked to the page after the current one.

```
// get the next page
  var nextPage = fwFolder.fwNextPage(fwPage);

  // set "foobar" parameter to that page
  fwParameters["foobar"].fwValue.fwInternal = nextPage ;
```

If you want to get the page object the is linked to you would do the following:

```
var currPage = fwParameters["foobar"].fwValue.fwInternal;
```

This will give you the actual page - and you can then (if you need to) traverse the objects on that page, getting text (or whatever else you need) from that page for use in your action.

If however you want the string required to make a link relative to the page on which the action is can do:

```
var linkStr = fwParameters["foobar"].fwValue.toString();
```

or

```
var linkStr = fwParameters["foobar"].toString();
```

or use the `fwParameters["foobar"]` parameter in a situation where JavaScript will coerce the link automatically and you will get the value that you need.

Color Parameters

Color parameters are a little different again. The `fwValue` will return the color object. All color objects are objects within the Freeway document and you can not change these objects within an action. What you can do is change the `fwValue` so that it accesses a different color object. You could, for example, get this color object from the documents `fwColors`.

If you call the `toString` method of a color object it will return the name of the color. This also means that of you call the `toString` method of the parameter object you will get the name of the colour. Unfortunately in most situations this is not useful.

If you want to get the hex string of a color you will have to access the `fwHex` method of the color parameter

```
var param = fwParameters["my colour"]; // get the parameter object
var colourStr = param.fwValue.fwHex; // get the colour
```

It is worth pointing out that a color does not always have a hex value. You can have a colour that is set to none. You can most easily check for this by checking the `fwHasValue` property of the colour.

```
var param = fwParameters["my colour"]; // get the parameter object
var colour = param.fwValue; // get the colour

if (colour.fwHasValue)
  alert("hex value = ", colour.fwHex);
else
  alert("no value for this colour");
```

If you need the RGB values of a colour you can get them from the `fwValue` property. This will return a JavaScript object with the properties red, green and blue that you can then use.

```
var param = fwParameters["my colour"]; // get the parameter object
var colour = param.fwValue; // get the colour
var rgb = colour.fwValue;
alert( '"'+rgb.red+','+rgb.green+','+rgb.blue+'"');
```

File Parameters

The file parameter allows the user to specify a file in the interface. There is no file-object associated with this. Accessing the `fwValue` property will effectively call the `toString` method of the object. There is no provision to set the object through the `fwValue` property.

The `toString` method of the object will return the relative path of the file relative to the page on which the action resides. If you need to specify a file programatically you can do so using the `fwSpecify` method. This method is only defined for the file parameter. It is most useful in that it allows you to specify a file that is either another file parameter or if your code has created a `FWFile` object.

File parameters can upload files but they will not do so unless you trigger this. A file specified in a file parameter will not be uploaded unless the action resolves the file name, or the file name is resolved within the action. This resolution will be caused by the name of the file being asked for. So you can easily trigger this by calling the `toString` method of the parameter. Additionally if the parameter is referenced outside of JavaScript this will also cause it to be uploaded. In practice this means that files are generally uploaded automatically as you will generally want to use the filepath of an action within the action. What this means in practice is that even if you do not want to use the file-path in the action you should still call `toString` even though you will not use the results.

Image Parameters

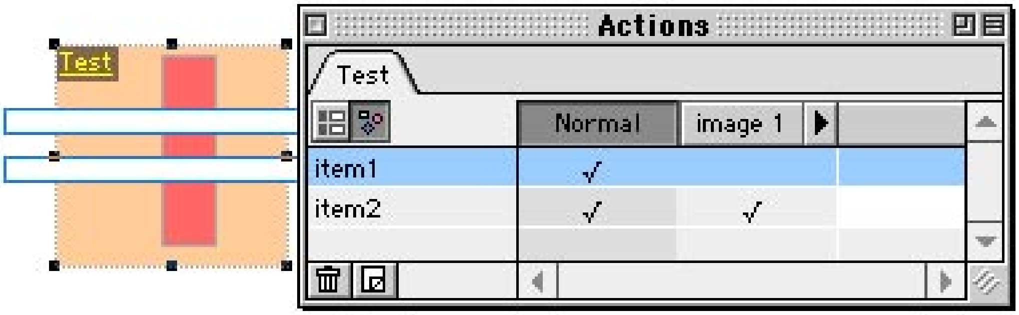
Image parameters are a sub-class of file parameters. As a user you can use them as file parameters (to specify GIF or JPEG files), but you can also use them to specify image that are generated within freeway. The highest profile use of them is the rollovers in Freeway. These allow you to specify images by turning on and off images that overlap each other.

The image parameter supports all the method of the file parameter so it is possible for you to use them in the same way - however the story of using these things properly is a bit more complex.

The additional complexity with image parameters is that an image can be sliced. If you consider the following action.

```
<item-action name="Test">
<action-image name="image 1">
</item-action>
```

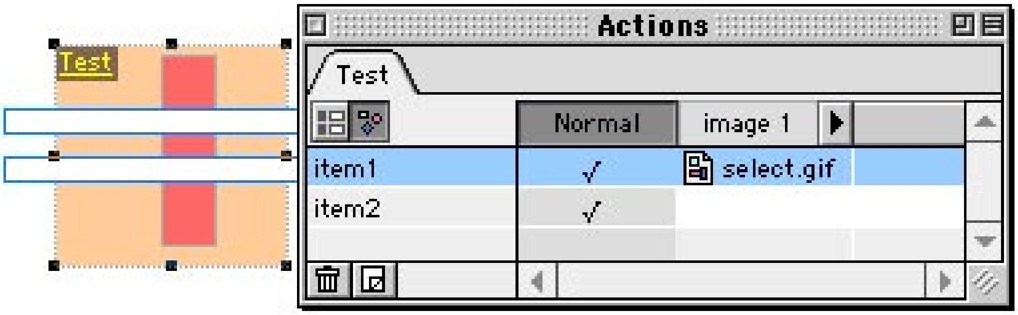
applied to a graphic that is cut into three by overlapping HTML boxes



	Normal	image 1
item1	√	
item2	√	√

You can see that the image parameter will actually have three graphics associated with it - one for each slice.

On the other hand if you associate a file with the parameter there will be just a single file associated with it.



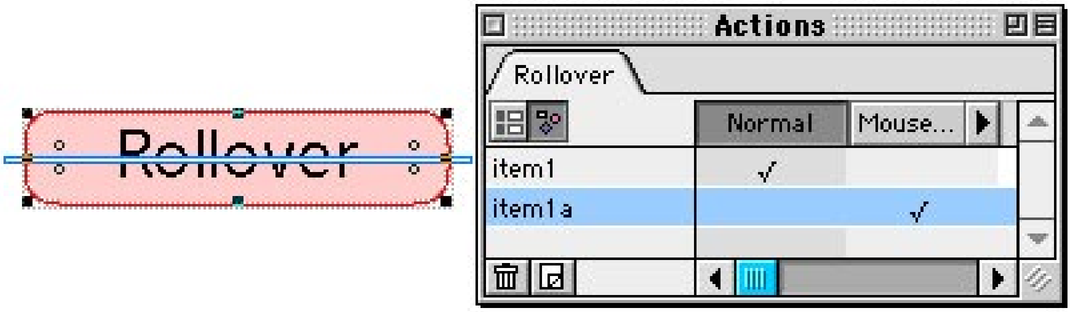
	Normal	image 1
item1	√	select.gif
item2	√	

Within your Freeway Action the way to handle this situation is to use the `fwFindAllImages` method.

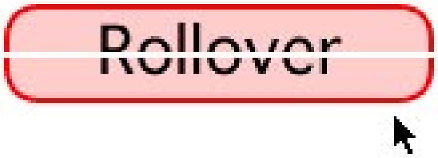
```
var images = fwParameters["image 1"].fwFindAllImages();
```

This method will return an array of all the images that are defined for the parameter. If the parameter has no image set the array will be empty (will have no elements), if the parameter has a file set then the array will have the path of just that one image. If the parameter has a number of images (due to the image being sliced) then it will contain an element for each slice.

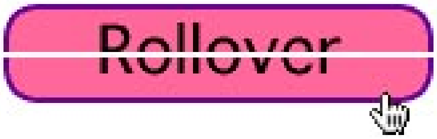
In your code you should account, and test, that your action works correctly when the image is sliced and when the image has a single image file. If you are careful you can do things like write a rollover that will work properly even when an image is sliced - so that, for example, all parts of the rollover will rollover together.



A sliced Rollover



Normal "rest" state



MouseOver state - both slices rolover together.