YAHOO!
WIDGET ENGINE 3.0
Reference Manual

Version 3.0
December 7, 2005

Release History

First Release		February 10, 2003
Second Release		February 12, 2003
Third Release		February 15, 2003
Fourth Release		February 19, 2003
Fifth Release	Version 1.5	July 23, 2003
Sixth Release	Version 1.5.1	September 26, 2003
Seventh Release		October 8, 2003
Eighth Release	Version 1.6.2	June 6, 2004
Ninth Release	Version 1.8	November 8, 2004
Tenth Release	Version 1.8.1	November 24, 2004
Eleventh Release	Version 1.8.3	January 18, 2005
Twelfth Release	Version 2.1	July 23, 2005
Thirteeth Release	Version 2.1.1	August 3, 2005
Fourteenth Release	Version 3.0	December 7, 2005

Thanks to all who have submitted comments and corrections.

The Basics

The Yahoo! Widget Engine (or simply the 'Widget Engine' or at times 'engine' as used in this document) uses XML to define Widgets and the objects that make them up. This makes a clear hierarchy for what each object is, and the order it's drawn in as well as associating the correct attributes with each object.

A very simple Widget might look like this:

```
<widget>
  <debug>on</debug>

  <window title="Sample Yahoo! Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>


        <image src="Images/Sun.png" name="sun1">
              <hOffset>250</hOffset>
              <vOffset>250</vOffset>
              <alignment>center</alignment>
        </image>

        <text data="Click Here" size="36" style="bold">
              <name>text1</name>
              <hOffset>250</hOffset>
              <vOffset>100</vOffset>
              <alignment>center</alignment>
              <onMouseUp>
                    sun1.opacity = (sun1.opacity / 100) * 90;
              </onMouseUp>
        </text>
  </window>
</widget>
```

All it does is reduce the opacity of an image by 10% every time the user clicks on

the text that says "Click Here". Obviously this isn't terribly useful but we'll use this simplified example to illustrate a few points. This sample depends on one external file, `Images/Sun.png` if you run it without that it will display a "missing image" placeholder.

Firstly, note the structure of the Widget: XML is a symmetrical language in that each object specifier (e.g. `<text>`) has a corresponding terminator (`</text>`). Within these pairs the attributes of the objects are defined such as screen positions, alignments, etc. Also note that objects defined in XML (like `sun1`) can be manipulated in JavaScript (see the `onMouseDown` handler in the `text1` object). Name of objects must begin with a letter and only letters, numbers and underscores are allowed. The XML for a Widget is stored in a file with the extension `.kon` (see below for a discussion of the bundle this file lives in).

Real Widgets can have dozens of images and text objects, multiple JavaScript sections (often in external files) and will usually create new objects at runtime using JavaScript to implement complex functionality.

By far the best and easiest way to get started creating Yahoo! Widgets is to take an existing Widget and start making changes to it. The Widget Engine comes with a selection of Widgets which perform a variety of tasks, any of which would be a good place to start – just remember that although the XML and JavaScript in these Widgets is freely available for reuse, the art assets are not and they must not be redistributed.

XML Syntax

We have a robust XML parser, and this means that you can use either style of tag notation or mix and match. The two styles being:

```
<image>
  <src>images/image.png</src>
  <name>myImage</name>
</image>
```

or:

```
<image src="images/image.png" name="myImage"/>
```

Mixing and matching is okay too:

```
<image src="images/image.png">
  <name>myImage</name>
</image>
```

Entities

Entities are an XML construct that allow you to specify a character via a special escape sequence. Some characters are used to parse the XML syntax and are considered reserved. The symbol & is used as the entity escape start (and for that reason is also a reserved character). The standard set of entities are used to represent XML special characters:

```
&amp;      &
&quot;     "
&apos:     '
&lt        <
&gt        >
```

You can also use entities to specify a character by its unicode code point:

```
&#32;      <space character, decimal>
&#x20;     <space character, hex>
```

These arbitrary entities only work in version 2.1 or later.

JavaScript

Because the XML engine looks for the < and > symbols to mark blocks of XML data, our JavaScript engine needs to have these symbols replaced with `<` and `>` respectively. For example:

```
<onMouseUp>
  if (x &lt; 5)
    displayResults();
</onMouseUp>
```

Alternatively you can use XML comments to hide the JavaScript code from the XML engine just as is commonly done in HTML, like so:

```
<onMouseUp>
<!--
  if (x < 5)
    displayResults();
//-->
</onMouseUp>
```

This is generally preferred because it makes the code easier to read.

In version 2.1 or later, you can use CDATA sections (which are actually more correct to use these days, and largely necessary if you put the parser into strict mode):

```
<onMouseUp>
<![CDATA[
  if (x < 5)
    displayResults();
]]>
</onMouseUp>
```

You can also make references to external JavaScript which we will cover later.

Strict Mode

In version 2.1 and later, you can put the XML parser into a 'strict' mode. This means it enforces the rules of XML in ways the parser did not in the past. In fact, it was much too

lenient in many ways. To enable this, you can just add the following line to the top of your XML file:

```
<?konfabulator xml-strict="true"?>
```

In strict mode, the following things are enforced:

1) All attribute values must be put into quotes.
2) No stray "&" characters are allowed in a normal text section (i.e.use &).
3) Entities (things that start with "&") are evaluated inside attribute values.
4) No double dash ("--") allowed inside a comment. For this reason, it's best to put code into CDATA blocks.
5) If an external file is included, we do not replace entities such as < in that file.

CDATA blocks are available in version 2.1 or later.

File Paths

File paths in the engine are always relative to the location of the XML file. That means a file reference without a directory (e.g. `main.js`) will be looked for in the same directory as the XML file while one with a directory (e.g. `javascript/main.js`) will be looked for in the specified subdirectory of the directory the XML file resides in. It is not advised to use absolute paths (ones that begin with a `/`) since the disk layout of people's machines can differ quite markedly.

Widget Packaging

On Windows the files that make up a Widget are stored in a `.widget` file. This is a standard ZIP file that has had its extension changed to `.widget`.

On Mac OS X the files are packaged together in a *bundle,* which is a directory that is treated as a single unit by the operating system. You can control-click on one of the default Widgets and choose the **Show Package Contents** option to see this structure in use.

Both the Mac OS X and Windows versions of the engine can read the zipped up .widget files, so it is the best choice when doing cross-platform Widgets. There is a utility available on the Yahoo! Widgets web site (in Workshop) which can assist you in building or taking apart these .widget bundles called Widget Converter.

Whether on Windows or Mac OS X `.widget` bundle has the following structure:

```
myWidget.widget
  Contents
       myWidget.kon
       Resources
            <any files used by the Widget>
```

The `.kon` file contains the actual Widget code (similar to the sample Widget in the section above). At present, the .kon file must be contained in a folder called Contents. You can put resources like pictures, etc. anywhere you like, but typically they would be

put into a Resources folder, as shown above.

If you do not use the Widget Converter Widget and instead decide to zip these up manually, this is best done on a PC by right clicking the .widget folder and creating a ZIP file from that. On the Mac you can use something like DropZip.

It should be noted that while you are developing your Widget, you do not need to create a zipped up Widget file to test each time you make a change. You can merely double-click the .kon file.

You should never modify your Widget package at run time. That is, do not use your Widget package to store information inside of itself. While most Widgets use preferences to store their settings, there are some Widgets that have instead stored information inside its own package. With the advent of our zipped format, this has proven to be somewhat fruitless. When the Widget Engine runs a zipped Widget, it first unzips it into a special location and then runs it from there. In recent releases, this unzipping happens every time you run the Widget, so if you stored information in your package, it will be lost. To help accommodate Widgets that need to store permanent data, there is a `system.widgetDataFolder` folder path you can use to store your Widget's permanent info.

Widget Runtime

This section discusses how Widgets are run and some of the issues one needs to keep in mind to perhaps alleviate any confusion about how things work.

When a Widget is opened, it is run as a separate process. This is done to ensure one Widget's fate does not affect the rest of the Widgets a user might be running.

A Widget that is zipped up is unzipped into a special location (/tmp on the Mac, and C:\Documents and Settings\<user>\Local Settings\Application Data on the PC). A Widget that is not zipped is run right from where it is located. For this reason you can never rely on where your Widget is. Once we locate the .kon file in the Widget, we set the current directory to the directory in which we found the .kon file. So for example, if your .kon file is in the Contents folder as it normally would be, the current working directory would be Contents. This allows relative paths to resources to work. A .kon file would reference an image as Resources/Image1.png, for example, if its images were inside a Resources folder inside Contents.

Once the .kon file is located and our current directory is set, the file is parsed and the objects defined therein are created. Once everything is created successfully, the onLoad handler (see the 'action' object documentation) is called. At this point your Widget can do whatever it needs to do to initialize itself. Take care to not linger in your onLoad handler, because it is typically executed before your Widget is made visible (i.e. many Widgets set their window to be hidden at first and show it at the end of their onLoad handlers). Once the onLoad handler is run successfully, your Widget is now running!

The next time your Widget is run, the Widget is unzipped again. For this reason you cannot rely on storing information in your Widget bundle. Use `widgetDataFolder` as

mentioned previously.

The Widget Engine keeps track of what Widgets are open automatically. The next time the engine is launched, it will automatically reopen any Widgets that were running at the time it was last quit.

Actions

Actions are the lifeblood of Widgets. These are where you get to define how a Widget behaves when the user interacts with the Widget, etc. In versions prior to 3.0, the only way to specify an action was to set the action to some Javascript text. This text was evaluated and run when the user clicked, for example:

```
<onMouseUp>
    print( "hello" );
</onMouseUp>
```

The limitations are a) you were never able to use the Javascript 'this' object to refer generically to the object that the action was running for, and b) if you had several objects with the same code, you'd have to duplicate the Javascript and change the names of the objects to reflect each object you attached the code to.

To remedy this, in version 3.0 or later, the engine now supports using proper Javascript functions for these actions. In 3.0, no parameters are sent to the actions, but it is the intention to pass proper parameters in the future. For example, an onMouseUp handler would receive the x and y coordinates of the mouse instead of inspecting system.event.

To use functions, you can either tell the engine you want to use a function in the XML by using an attribute (and only attribute, a sub-element will not work), or by setting the property to the function to call in Javascript:

```
<!-- In XML -->
<onMouseUp function="_myMouseUp"/>

// in Javascript
myImage.onMouseUp = _myMouseUp;

// and someplace in your JS code, you must have the
// function defined:
function _myMouseUp()
{
   print( this.opacity );
}
```

Object Names

In the XML description, you can set a `<name>` property. This defines the global Javascript object that will be created and bound to the object the name is a part of. For example:

```
<window name="mainWindow" .../>
```

Will end up creating a JS variable at the global scope with the name `mainWindow`. Because of this, all names must be unique. Also, because internally these names are used to track objects, they cannot be changed. Version 3.0 enforces this by making all name properties read-only at present. Any time you create an object on the fly using Javascript, an object is given a generic name, such as `Image001`. In the future, we hope to make it such that these names can be mutable.

Debugging

Some provision is made for debugging your Widgets. There is an xml tag "debug" which you can set to "on" (see the reference below for exact details). When set to on, a debug output window will open when your Widget is started. Calls to log() or print() in your JavaScript code will be routed into this window. Any errors encountered inside the Widget Engine or your Widget will also be reported in this window.

While developing your Widget, you should always turn debug to on so you can tell what is perhaps going wrong. For example, if you spell an attribute wrong, the output window will tell you this, along with where in your code the problem is.

It should be noted that especially on the PC, the debug window will *never* open unless debugging is on. On the Mac, there might be times where the Debug window might open automatically, at present (especially if a Widget's onLoad handler fails).

In version 2.1 and later, you can access a debugging mode by holding down the control and shift keys and selecting the Gear menu in the menu bar (Mac) or system tray (Windows). Once you turn the option on, any Widget you launch will have debugging forced on and the debug window will open. Because of this, you really don't have to use the debug tag in your Widget definition any longer. You can also have your users use this mode to help you diagnose issues.

Also in version 2.1 and later, the debug window has been enhanced with a new command line field. In this field you can issues commands (see the complete list by typing "/help" in the field) or merely evaluate some JavaScript. This is handy for inspecting the values of varaibles, etc. You can also trace variables and functions using the built-in commands.

Exceptions

Starting in version 2.0, the Widget Engine will throw proper exceptions when things go awry. This is true particularly in the COM interfaces for the Windows release and for the filesystem object for both platforms. While not everything that could throw an exception is throwing at present, it's important to realize this and to use try/catch handlers in places. In using COM, it's pretty much a necessity as it can help you bail out and deal with failures to connect, etc. Currently the exception is merely a string, so worst case you can print the string in an alert or the like. The PIM Overview uses try/catch in its Outlook handling.

Widget Preferences

A Widget can provide a number of preference objects to allow itself to save out settings.

These settings are saved out in per-user preference storage. On the Mac, this is in ~/Library/Preferences/Konfabulator. On the PC, this is in HKEY_CURRENT_USER\Software\Yahoo\WidgetEngine.

MinimumVersion

The minimum version attribute of a <widget> tells the engine what version of the engine is required to run a Widget. But starting in version 3.0 it also tells the engine that this Widget has been revised for 3.0 and as such, we are using it to alter behaviors of certain aspects of how a Widget works. This is to help us move forward and fix things that are wrong without breaking any Widget that does not have the minimum version set appropriately.

If you set your minimum version to 3.0 (which you should if you are taking advantage of features in 3.0), the following behaviors come into play:

1) No views are auto-bound to the default window. This used to be the case, but with the advent of hierarchical views in 3.0, this became problematic. As a result, you must specify the window that an object belongs to, or use frame.addSubview() to embed an object into a frame. If your interface is mostly constructed via XML, the simplest thing to do is enclose your image/text/frame/scrollbar/textarea objects inside your window object:

```
<window ...>
  <image src=.../>
  <text data=.../>
  <frame .../>
        <image src=.../>
  </frame>
</window>
```

The most common error you would probably encounter as you migrate your Widget to 3.0 is to see some of your views not appearing. This is all due to this behavioral change. Simply double-check that all your views are bound to some window or parent frame.

2) Javascript lifetime changes. In prior releases, calling delete on an object or setting it to null would make the object disappear from the window. This will no longer occur. If you wish for an object to be removed, you must call <object>.removeFromSuperview(). The point of this change is to make it easier to code a Widget. In the past you'd have to maintain lists of all your objects just to ensure they didn't disappear with the window. With the advent of subviews, the number of objects can become very unmanageable very quickly. Now you no longer need to care if you have a reference to an object if you've added it to the window. This means items that would never change during the course of your Widget never need to be tracked by you. It will make your code more obvious in many ways and you can instead just concentrate on doing what you do best.

3) We no longer blindly replace XML entities in your .kon or .js files when files are loaded. If you want to ensure that Javascript code that has < or > in it doesn't trip up the parser, you should use CDATA sections, as mentioned earlier.

4) Rotation changes. We now properly rotate about the effective h and vOffset of an object. This means if you center an image using hAlign and vAlign and then rotate it, it will rotate around the center of the image.

5) Javascript code in an XML element will be read as just that, Javascript code. Previously, the engine would try to see if it was a file by trying to read a file with the given path. Not cool. We now only try to read a file if your action has the 'file' attribute. If you want to include a file in the element, use include(). This should improve loading performance as we won't hit the filesystem for every chunk of Javascript code in your Widget.

XML Services

Starting in version 3.0 we now provide new services to allow you to work with XML more easily. In 3.0 we now have a built-in XML parser which is significantly faster than using the external Javascript-based xmldom.js file we've recommended in the past. This XML parser always operates in 'strict' mode (see above notes on strict mode).

The output of the parser is a Level 1 W3C DOM and we follow the RFC for said DOM to the letter. There are a few omissions (entities, for one), but the important core is there. You can also create and mutate these DOM trees to make your own XML documents and output them.

The DOM API is nice, but in general it's not very convenient to traverse an XML tree to find the important bits. So we've also added XPath 1.0 support (minus namespace-specific functions). This makes it much easier to pull out pieces of a XML tree than using the DOM API.

To aid in moving code into Yahoo! Widgets, and just to help people get comfortable we've added a real XMLHttpRequest object. This follows the minimum core API at present. For POSTing files, we still recommend you use the URL object instead.

Yahoo! Login Support

Version 3.0 and later allow you to use APIs which require a Yahoo! login. The engine itself takes care of the details of logging in and storing credentials. Your Widget only has to check the current login state or request to login. Once logged in, when sending the API request to the server, the engine automatically adds the user's credentials for you.

To behave like a good citizen, you should first check to see whether you are logged in by calling `yahooCheckLogin()`. If this returns true, you are all set to access the Yahoo! API your Widget would call. If it returns false, you should display a placard or some other indication that your Widget cannot display its information because the user is not currently logged in and give them a button/link/something to click to enable them to login from your Widget.

In your `onLoad` handler, for example:

```
if ( yahooCheckLogin() )
  loggedIn(); // display your UI in the logged in state.
else
  loggedOut(); // display your UI in the logged out state.
```

It is considered bad form to blindly call `yahooLogin()` in your `onLoad` handler.

When the user clicks your button to login, call `yahooLogin()`. If this function returns true, you are already logged in, so behave as such. But more likely it will return false, meaning the user must authenticate. When `yahooLogin()` returns false, you must simply go about your business and wait for an `onYahooLoginChanged` event to come to your Widget (i.e. the function behaves asynchronously). You might also get your `onYahooLoginChanged` handler called if the user logs in or out from the Gear menu.

When your `onYahooLoginChanged` handler is called, you must call `yahooCheckLogin()` to see what your new state is (this call also loads up the necessary cookies, etc.). Based on the state returned, you would either behave logged in or out, just as shown above for onLoad.

Be warned that even if `yahooCheckLogin()` returns true, your request to the API server might fail due to expired credentials. In this case, is your Widget's responsibility to call `yahooLogout()` so that other Widgets are informed of the situation.

Subviews/Frames

Starting in version 3.0, the Widget Engine now supports hierarchical views. Prior to 3.0 you could only have a flat list of objects (images, text, etc.) in a window. 3.0 introduces the Frame object, which allows you to add objects to it and treat it as a group of items. If you move the Frame, the subviews move with it. If you fade a frame, everything within it fades.

When an object is put inside a frame, its hOffset and vOffset become frame-relative. Basically, the offsets are always relative to a view's parent. So an image with an h/vOffset of 10, 10 will appear 10 pixels down and to the right of the top left of its parent frame. This allows you to not have to care where it necessarily is in the window at all.

Even the objects that are at the top level of the window and not apparently in any frame are really in a root view of the window. You can access this root view through the window object. The root is a special view and only contains those attributes and functions necessary to allow you to traverse the tree of views successfully.

The other things that Frames bring is the ability to scroll their contents. This makes it possible to create scrolling lists of search results and various other things. The Widget Engine also provides a standard ScrollBar object which you can attach to a Frame to scroll its contents. When a scroll bar is bound to a frame, mouse wheel support is automatically enabled as well. The ScrollBar object can have its standard thumb colorized, or if that doesn't meet your needs, you can supply your own images for the track and thumb.

The other things that Frames bring is the ability to scroll their contents. This makes it

possible to create scrolling lists of search results and various other things. The Widget Engine also provides a standard ScrollBar object which you can attach to a Frame to scroll its contents. When a scroll bar is bound to a frame, mouse wheel support is automatically enabled as well. The ScrollBar object can have its standard thumb colorized, or if that doesn't meet your needs, you can supply your own images for the track and thumb.

Security Windows

There are two types of security windows that may appear in the Widget Engine, though they both look similar. The first is a first run/modification window. On first run of a Widget that the Widget Engine is not familiar with or has ever seen before, a window will appear telling the user they are about to open a new Widget and have them confirm the action. This is to protect against Widgets that might just run without the user's knowledge. Also, if the user allows a Widget to run and later on that Widget is somehow modified, another window will appear the next time the Widget is launched, telling them of this fact. Again, the user can confirm or deny the request to launch depending on whether or not the modification was expected.

If you are actively debugging a Widget, you can turn debug mode on (which is probably a good idea anyway) and first run/modified security windows will be surpressed, so as not to bug you ever time you tweak your code and reload the Widget.

The second type of window is a 'sandbox' window. Currently, the only sandboxed action is logging into a user's Yahoo! account (more actions will be sandboxed in future releases). The first time a Widget attempts to log into a user's Yahoo! account, a window will appear to alert the user of this fact and ask whether the Widget should be granted permission to use their Yahoo! data. Sandbox windows cannot be disabled.

XML Reference

The following sections describe the objects and attributes that make up Widgets. Objects are organized into a hierarchy as follows:

```
<widget>
  <about-box/>
  <action/>
  <frame/>
  <hotkey/>
  <image/>
  <preference/>
  <scrollbar/>
  <text/>
  <textarea/>
  <window/>
</widget>
```

Other blocks we read as subblocks:

```
<menuItem/>
<shadow/>
```

Starting in version 2.1, you can now nest objects inside their containing window. This means you can put objects like images, text, and textareas into the block for the window:

```
<window>
  ...
  <image name="foo"/>
  <text .../>
</window>
```

Using this method, you do not need to include the window property for any of the nested images since the window is known to be the containing window specified in the XML. If you do specify a window, you will get an error in the debug window warning you of this fact.

<about-box>

block to define images for an about box

Attributes

`image/about-image`
`about-text`
`about-version`

Description

If used, the `about-box` XML block must contain one or more references to a path to an image contained in an image block.

image/about-image

block containing a path to an image

Description

The image attribute of the about-box block must contain a valid path to an image.

If more than one image attribute is used the images will be shown sequentially to the user. When they are the same size, they will simply replace each other, when they are different sizes, the first will fade out and the next will fade in.

Example

```
<about-box>
  <image>Resources/About.png</image>
  <image>Resources/Thanks.png</image>
</about-box>
```

Version Notes

The about-image synonym first appeared in version 2.1.

about-text

text to display

Description

You can specify any number of text objects to be displayed in your about box. These text items at present only appear on the first page of your about box. They can have the following attributes:

`color/colour`
`data`

```
hOffset
font
size
style
shadow
url
vOffset
```

Except for `shadow` and `url`, these are all the same properties as can be used on a full-fledged text object. See the section on text objects for information on how these attributes are used. See the section on the shadow object for information about how that object is structured. The `url` property turns the text object into a clickable link which will open a browser targeted at the url you specify.

Availability

Available in version 2.1 or later. The `url` property is available in 3.0 or later.

about-version

element to describe where and how the version should be placed

Description

This is essentially a special case of the text element, described above. It has all the same attributes, and can only be placed on the first page of an about box. The only difference is that this tag represents where the Widget's version number will appear. The version number is taken right from the Widget definition's `version` attribute.

Availability

Available in version 2.1 or later.

<action>

code block not associated with an object

Attributes

`file`
`interval`
`trigger`

Description

The `action` XML block defines when and how a Widget will execute code that is triggered automatically rather than by a user.

file

the path to an external JavaScript file

Description

Embedding JavaScript code into an XML file may present unique problems for some developers. Your preferred text editor may not gracefully support syntax highlighting for both XML and JavaScript at the same time, your JavaScript code may be large and complex and need better management, or you may just be frustrated by the impositions of having to escape common characters that would confuse the XML portion of the Widget. In order to alleviate any or all of these we allow you to reference an external file.

You can reference files by specifying the `file` attribute for the `<action>` block. Alternatively, you can simply use `include()`.

Example

```
<action trigger="onLoad" file="main.js"/>

<action trigger="onLoad">
    include( "main.js" );
</action>
```

interval

time in seconds to wait between triggers

Description

The `interval` attribute for the `action` block is to be used with the `onTimer` `trigger` attribute. It defines how many seconds, or fractions of a second, to wait between `onTimer` code executions.

If no interval is defined for an on timer trigger, it will default to one minute.

Example

```
<!-- This will cause the Widget to beep every
     two minutes -->
<action trigger="onTimer" interval="120">
  beep();
</action>

<!-- This will cause the counter to increase ten
     times a second -->
<action trigger="onTimer" interval="0.1">
  counter ++;
</action>
```

Starting in version 2.0 this mechanism is deprecated in favor of the new Timer objects (see the section on Timers later in this manual).

trigger

the event that triggers the enclosed code

Values

```
onGainFocus
onIdle
onKeyDown
onKeyUp
onKonsposeActivated
onKonsposeDeactivated
onLoad
onLoseFocus
onMouseDown
onMouseEnter
onMouseExit
onMouseUp
onPreferencesChanged
onRunCommandInBgComplete
onScreenChanged
onTellWidget
onTimer
onUnload
onWakeFromSleep
onWillChangePreferences
onYahooLoginChanged
```

Description

The `trigger` attribute for the action block defines what will trigger the contained block of code.

`onGainFocus` will trigger when the Widget is activated by the user. This is useful if you want your Widget to have an active and inactive state. This action is typically triggered when the Widget first starts running. In version 2.0 and later, you should generally use the onGainFocus handler on each window and reserve the Widget onGainFocus handler for truly Widget-wide activation handling.

`onIdle` executes five times a second, but we do not suggest using it as it will cause your Widget to use excessive amounts of CPU time.

`onKonsposeActivated` and `onKonsposeDeactivated` execute when the user invokes and dismisses Konsposé mode. This gives the Widget the opportunity to change display modes or take other actions desired at this point (for example, some Widgets display their "focused" mode as if `onGainFocus` had been received when Konsposé is active).

`onLoad` executes when the Widget is first loaded and is used to define and store functions that might be used elsewhere in the Widget.

`onLoseFocus` will trigger when the Widget is deactivated by the user. This is useful if you want your Widget to have an active and inactive state. In version 2.0 and later, you should generally use the onLoseFocus handler on each window and reserve the Widget onLoseFocus handler for truly Widget-wide activation handling.

`onPreferencesChanged` is executed when the user saves the preferences. Note that nothing is executed if they cancel out of the preferences dialog as they didn't actually change the preferences in that case.

`onRunCommandInBgComplete` is executed when a command started with `runCommandInBg()` completes (see documentation for that call).

`onScreenChanged` fires if any screen size, arrangement or color depth changes are made using the **Displays** System Preference panel (note that the screen the Widget itself is on may or may not have been affected).

`onTellWidget` is called when another Widget or application calls the tellWidget interface to send your Widget a message. You should be very careful about what you decide to do with the message you receive. See the section on tellWidget later on in this document for more detail. This trigger is available in Widget Engine 2.0 or later.

`onTimer` executes at regular intervals based on what's defined in the `interval` attribute. If no `interval` attribute is defined, it will default to a one minute interval. Note that there can only be one `onTimer` trigger per Widget. For this reason, in version 2.0 and later we offer a new Timer object which allows you to have multiple timers running at different frequencies. See the section on Timers for more information.

`onUnload` executes when the Widget is closed. This is useful for doing any last minute manual preference saving (preferences set in the Widget Preferences dialog are saved automatically when they are changed by the user), as well as making sure any external applications you may be talking to are closed up and aware of your departure. Note that you should not perform any lengthy operations in this trigger as Widgets are encouraged to shutdown quickly (an example of a lengthy operation would be retrieving something from the network).

`onWakeFromSleep` executes when the machine wakes from a state of sleep. It should be noted that some desktops have a several second lag between waking up and reconnecting to the network, so you may want to add a `sleep()` call to your code if your Widget wants to connect to the internet. In version 3.0 or later, timers are stopped when the machine goes to sleep and are not restarted until onWakeFromSleep is called.

`onWillChangePreferences` executes when the user asks to edit the Widget's preferences (or when the `showWidgetPreferences()` JavaScript call is made).

`onYahooLoginChanged` executes when the user either logs in or logs out of their Yahoo! account. When called, you can check the current state of the user login by calling yahooCheckLogin().

The remaining triggers, `onKeyDown`, `onKeyUp`, `onMouseDown`, `onMouseUp`, `onMouseEnter` and `onMouseExit` execute when the corresponding user action is detected within the main window of the active Widget, and there is no other object to receive them. Note that using the global scope mouse actions will cause your Widget to no longer be draggable without having to hold down the command key.

Example

```
<!— Redraw the clock when we wake from sleep -->
<action trigger="onWakeFromSleep">
  updateClockFace();
</action>

<!— Update our info when the user changes the preferences -->
<action trigger="onPreferencesChanged">
  refreshTickerSymbols();
</action>
```

<frame>

block defining a frame object

Frame objects act as containers for other objects. As such, you can nest other view objects inside them in the XML, as well as use Javascript to place other objects inside them. When moved, all subviews of a frame move. Similarly, when the opacity of a subview changes, so does the effective opacity of everything in it.

Frames also allow scrolling. You can do so manually by adjusting the scrollX and scrollY properties, but you can also simply attach a scrollbar to a frame and have everything just work automatically.

Attributes

```
contextMenuItems
hAlign
height
hLineSize
hOffset
onContextMenu
onDragDrop
onDragEnter
onDragExit
onMouseDown
onMouseEnter
onMouseExit
onMouseMove
onMouseUp
onMouseWheel
onMultiClick
opacity
scrollX
scrollY
visible
vAlign
vLineSize
vOffset
width
window
zOrder
```

contextMenuItems

Specifies an array of context menu items.

Description

You can add items to the standard context menu that appears when the user right-clicks

the mouse button on your frame. You can also dynamically build your context items by specifying some JavaScript to execute on your onContextMenu tag (see onContextMenu for more information).

You specify your items by including an array of menuItem objects. See the section on menuItem for more information about them.

JavaScript

```
myObjectName.contextMenuItems
```

Example

```
<frame>
  ...
  <contextMenuItems>
    <menuItem title="Test" onSelect="beep();"/>
    <menuItem title="Another Test">
      <onSelect>alert( 'hello' );</onSelect>
    </menuItem>
  </contextMenuItems>
</frame>
```

See the onContextMenu section for an example of building a context menu in JavaScript.

Availability

Available in version 3.0 or later.

hAlign

control the horizontal alignment of a frame

Description

The hAlign property of an object defines the initial horizontal alignment with respect to its hOffset property. For example, an object with `right` alignment will be drawn so that its right edge appears at the `hOffset`. The default alignment is `"left"`.

Valid values are: `"left"`, `"right"` or `"center"`.

JavaScript

```
myObjectName.hAlign
```

Example

```
<frame>
  <hAlign>right</hAlign>
</frame>
myFrame.hAlign = "left";
```

Availability

Available in version 3.0 or later.

height

the height of the object

Description

The `height` attribute controls the vertical dimension of an object. If no height is specified for a frame, its height is determined automatically by the extent of its subviews.

JavaScript

`myObjectName``.height`

Example

```
<frame>
  <height>300</height>
</image>

myFrame.height = 300;
```

Availability

Available in version 3.0 or later.

hLineSize

the size of a line of data for use when scrolling

Description

The `hLineSize` property specifies how far a frame should scroll (in pixels) if the `lineLeft()` or `lineRight()` functions are called. It is also factored in when the frame reacts to the mouse wheel (if a scroll bar is attached). The default line size is 10 pixels.

JavaScript

`myObjectName``.hLineSize`

Example

```
<frame>
  <hLineSize>5</hLineSize>
</image>

myFrame.hLineSize = 5;
```

Availability

Available in version 3.0 or later.

hOffset

the horizontal offset of an object

Description

The `hOffset` attribute of an object defines the horizontal (left to right) offset for the image based on 0,0 being the upper left hand corner of the its parent view (superview). The greater the value assigned, the farther to the right the object will appear.

JavaScript

myObjectName.hOffset

Example

```
<frame>
  <hOffset>30</hOffset>
</frame>
```

Availability

Available in version 3.0 or later.

hScrollBar

the horizontal scroll bar for this frame

Description

The `hScrollBar` attribute of a frame defines what scroll bar object should control the horizontal scrolling for this frame. When expressed in XML, you specify the name of a `<scrollbar>` object you wish to bind to the frame for its `hScrollBar`. If the scroll bar object does not exist, an error will appear in the Widget's debug window.

Attaching a scroll bar will do all the automatic setup for communicating between the frame and the scroll bar.

JavaScript

myObjectName.hScrollBar

Example

```
<frame>
  <hScrollBar>my_scrollbar</hScrollBar>
</frame>
<scrollbar name="my_scrollbar" ... />

// in Javascript:
myFrame.hScrollBar = my_scrollbar;
```

Availability

Available in version 3.0 or later.

onContextMenu

called when a context menu is about to appear

Description

The simplest way to specify context menu items that get added to the standard context menu for a Widget is to use the contextMenuItems tag in the XML. However, for those Widgets that need to build their items dynamically, the onContextMenu handler is your hook to do so. When the menu is about to be presented, this is called for all elements under the mouse from front to back in the view order until some view responds. When handling this, you should simply build your context menu items and set your contextMenuItems property to the array of items.

JavaScript

myFrame.onContextMenu

Example

```
<onContextMenu>
var items = new Array();
items[0] = new MenuItem();
items[0].title = "This is the title";
items[0].enabled = false;
items[0].checked = true;
items[0].onSelect = "alert( 'you chose it!' );";

items[1] = new MenuItem();
items[1].title = "This is the second title";
items[1].onSelect = "beep();";

myImage.contextMenuItems = items;
</onContextMenu>
```

Availability

Available in version 3.0 or later.

onDragDrop

called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string is dragged from another application and dropped on the object.

In the `onDragDrop` action objects can access `system.event.data` to see what was dropped. This is an array of strings whose first element specifies what type of object was dropped: `"filenames"`, `"urls"` or `"string"` The remaining elements of the array are the items that were dropped.

JavaScript

myObjectName.onDragDrop

Example

```
<frame>
  <onDragDrop>
       if (system.event.data[0] == "filenames")
       {
         processDroppedFiles(system.event.data);
       }
  </onDragDrop>
</frame>

myFrame.onDragDrop = "handleDragDrop();";
```

Availability

Available in version 3.0 or later.

onDragEnter

called when an item is dragged into the object

Description

The `onDragEnter` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object. This happens before the item is dropped (indeed it may not be dropped as the user can change their mind).

This is useful for triggering a visual change of the object to indicate to the user that the dragged object will be accepted or rejected if it is dropped. Information about the item being dragged is contained in `system.event.data` (see `onDragDrop` for details).

JavaScript

myObjectName.onDragEnter

Example

```
<frame>
  <onDragEnter>
        highlightDropTarget(well);
  </onDragEnter>
</frame>

well.onDragEnter = "highlightDropTarget(well);";
```

Availability

Available in version 3.0 or later.

onDragExit

called when an item is dragged out of the object

Description

The `onDragExit` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object and then out again.

This is useful for undoing things that were done in `onDragEnter`.

JavaScript

myObjectName.onDragExit

Example

```
<frame>
  <onDragExit>
        unhighlightDropTarget(well);
  </onDragExit>
</frame>

well.onDragExit = "unhighlightDropTarget();";
```

Availability

Available in version 3.0 or later.

onMouseDown

called when the mouse button is down inside the object

Description

The `onMouseDown` property specifies JavaScript code that will execute when the user presses the mouse button down within the object.

JavaScript

`myObjectName``.onMouseDown`

Example

```
<frame>
  <onMouseDown>
        beep();
  </onMouseDown>
</frame>

myFrame.onMouseDown = "beep();";
```

Availability

Available in version 3.0 or later.

onMouseEnter

called when the mouse enters the object

Description

The `onMouseEnter` property specifies JavaScript code that will execute when the user has moved the cursor within the object.

This is useful for triggering a visual change of the object based on a rollover state.

JavaScript

`myObjectName``.onMouseEnter`

Example

```
<frame>
  <onMouseEnter>
        print( "Mouse entered!" );
  </onMouseEnter>
</frame>

myFrame.onMouseEnter = "handleEntered();";
```

Availability

Available in version 3.0 or later.

onMouseExit

called when the mouse exits an object

Description

The `onMouseExit` property specifies JavaScript code that will execute when the user has moved the cursor from within the object to outside the object.

This is useful for triggering a visual change of the object based on a rollover state.

JavaScript

myObjectName.onMouseExit

Example

```
<frame>
  <onMouseExit>
        print( "Sadly, the mouse has left us." );
  </onMouseExit>
</frame>

myFrame.onMouseExit = "handleMouseExit();";
```

Availability

Available in version 3.0 or later.

onMouseMove

called when the mouse moves within an object and the mouse is down

Description

The `onMouseMove` property specifies JavaScript code that will execute when the user drags the mouse cursor within the bounds of an object. The current mouse position is available in the `system.event` object.

JavaScript

myObjectName.onMouseMove

Example

```
<frame>
  <onMouseMove>
    print(system.event.x + ", " + system.event.y);
  </onMouseMove>
</frame>

myFrame.onMouseMove = "handleMouseMove();";
```

Availability

Available in version 3.0 or later.

onMouseUp

called on mouse up in an object

Description

The `onMouseUp` property specifies JavaScript code that will execute when the user has released the mouse after having it down within the object.

This is useful for triggering a visual change of the object based on a pressed state.

Please note that `onMouseUp` will trigger even if the mouse is not inside the object when the mouse is released. In order to create buttons which have correct mouse events you must employ the use of all four mouse event handlers in order to communicate the state of the mouse, and its intersection status (see the included Calendar Widget for an example of this).

JavaScript

`myObjectName``.onMouseUp`

Example

```
<frame>
  <onMouseUp>
       handleOnMouseUp();
  </onMouseUp>
</frame>

myFrame.onMouseUp = 'handleOnMouseUp();';
```

Availability

Available in version 3.0 or later.

onMouseWheel

called when the mouse wheel is moved while over the frame

Description

The `onMouseWheel` property specifies JavaScript code that will execute when the user moves the mouse wheel while hovering over the object. The delta can be gotten from system.event.scrollDelta.

You normally don't need to use this hook, as when a scroll bar is attached to a frame, the mouse wheel is handled for you automatically.

JavaScript

myObjectName.onMouseWheel

Example

```
<frame>
  <onMouseWheel>
        handleOnMouseWheel(system.event.scrollDelta);
  </onMouseWheel>
</frame>

myFrame.onMouseWheel = 'handleOnMouseWheel(
                        system.event.scrollDelta);';
```

Availability

Available in version 3.0 or later.

onMultiClick

a multiple click just occurred

Description

You can easily trap double-clicks (or triple-clicks, etc.) using the `onMultiClick` handler. Whenever your `onMultiClick` handler is called, you can inspect `system.event.clickCount` to see what the value is. It will always be 2 (for a double-click) or greater.

It is also possible to inspect this `system.event.clickCount` in an onMouseUp handler as well in lieu of using `onMultiClick`. However, the advantage to using `onMultiClick` is that it does not interfere with window dragging the way that `onMouseUp` does, i.e. a mouse up handler on an image will prevent a window from being dragged if you click that image. If your image only needs to respond to multi-clicks, you can use `onMultiClick` and the Widget will still be able to be dragged as usual.

```
<onMultiClick>
   if ( system.event.clickCount == 2 )
      alert( "Double Click!" );
</onMultiClick>
```

Availability

Available in version 3.0 or later.

opacity

the opacity of an object

Description

The `opacity` property allows you to specify a value from 0 to 255 which controls the alpha value with which the object is rendered. An opacity of 0 is completely transparent (invisible) and has such side effects as preventing the object from reacting to mouse events. A value of 255 will render the image at its natural opacity.

Example

```
<frame>
  <opacity>128</opacity>
</frame>

myFrame.opacity = 33;
```

Availability

Available in version 3.0 or later.

scrollX

the horizontal scrolling offset

Description

The `scrollX` property allows you to specify the horizontal scrolling offset. Setting this property to -10 would scroll a frame's contents to the left 10 pixels, for example. Normally you don't need to modify this property directly. Simply attaching a scroll bar to a frame will cause this property to get updated as necessary to scroll the contents.

Example

```
<frame>
  <scrollX>-10</scrollX>
</frame>

myFrame.scrollX = -20;
```

Availability

Available in version 3.0 or later.

scrollY

the vertical scrolling offset

Description

The `scrollY` property allows you to specify the vertical scrolling offset. Setting this property to -10 would scroll a frame's contents upward 10 pixels, for example. Normally you don't need to modify this property directly. Simply attaching a scroll bar to a frame will cause this property to get updated as necessary to scroll the contents.

Example

```
<frame>
  <scrollY>-10</scrollY>
</frame>

myFrame.scrollY = -20;
```

Availability

Available in version 3.0 or later.

vAlign

controls the vertical alignment of an object

Description

The vAlign property of an object defines how it is positioned vertically relative to its vOffset. For example, an image with a `bottom` alignment will be drawn so that its bottom edge appears at the `vOffset`. If this tag is not specified, the default value is `"top"`.

Valid values are: `"top", "bottom"` or `"center"`.

JavaScript

myObjectName.vAlign

Example

```
<frame>
  <vAlign>bottom</vAlign>
</frame>

myFrame.vAlign = "bottom";
```

Availability

Available in version 3.0 or later.

visible

controls the visibility of an image

Description

You can set the visible property of an image to show or hide it by setting it to true or false, respectively. This allows you to hide objects without affecting their opacity, or having to save off the current opacity to restore it later. The default visibility for any object if not specified is true.

JavaScript

myObjectName.visible

Example

```
<frame>
  <visible>false</visible>
</frame>

myFrame.visible = true;
```

Availability

Available in version 3.0 or later.

vLineSize

the size of a line of data for use when scrolling

Description

The vLineSize property specifies how far a frame should scroll (in pixels) if the lineUp() or lineDown() functions are called. It is also factored in when the frame reacts to the mouse wheel (if a scroll bar is attached). The default line size is 10 pixels.

JavaScript

myObjectName.vLineSize

Example

```
<frame>
  <vLineSize>5</vLineSize>
</image>
```

```
myFrame.vLineSize = 5;
```

Availability

Available in version 3.0 or later.

vOffset

the vertical offset of an image

Description

The `vOffset` property defines the vertical (top to bottom) offset for the object based on 0, 0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther down the object will appear.

JavaScript

object`.vOffset`

Example

```
<frame>
  <vOffset>20</vOffset>
</frame>
```

Availability

Available in version 3.0 or later.

vScrollBar

the vertical scroll bar for this frame

Description

The `vScrollBar` attribute of a frame defines what scroll bar object should control the vertical scrolling for this frame. When expressed in XML, you specify the name of a `<scrollbar>` object you wish to bind to the frame for its `vScrollBar`. If the scroll bar object does not exist, an error will appear in the Widget's debug window.

Attaching a scroll bar will do all the automatic setup for communicating between the frame and the scroll bar.

JavaScript

myObjectName`.vScrollBar`

Example

```
<frame>
  <vScrollBar>my_scrollbar</vScrollBar>
</frame>
<scrollbar name="my_scrollbar" ... />

// in Javascript:
myFrame.vScrollBar = my_scrollbar;
```

Availability

Available in version 3.0 or later.

width

the width of an object

Description

The `width` property controls the horizontal size of an object. If none is specified (or it's set to -1), a frame will use the vertical extent of its subviews to determine its size.

JavaScript

`myObjectName``.width`

Example

```
<frame>
  <width>300</width>
</frame>

myFrame.width = 200;
```

Availability

Available in version 3.0 or later.

window

the window to which this object belongs.

Description

You can specify the window an object belongs to by specifying its name in the XML or its variable in JavaScript. If you do not specify a window, the object is automatically attached to the first window found in the XML description of a Widget.

JavaScript

myObjectName.window

Example

```
<window name="fred" width="100" height="100"/>
<frame>
  <window>fred</window>
</frame>

// Or in code
var myWind = new Window();
myFrame.window = myWind;

// You can also specify it in the constructor

var myFrame = new Frame( myWind );
```

Availability

Available in version 3.0 or later.

zOrder

the stacking order of an object

Description

The `zOrder` property defines the stacking order of an object. Objects with a higher `zOrder` are drawn on top of those with lesser `zOrder`s. Normally the `zOrder` is determined by the order in which objects are defined in the XML file with earlier objects being drawn under later ones but it can also be manipulated using JavaScript at runtime.

JavaScript

myObjectName.zOrder

Example

```
<frame>
  <zOrder>10</zOrder>
</frame>

myFrame.zOrder = customZOrder++;
```

Availability

Available in version 3.0 or later.

<hotkey>

block defining a hotkey and associated default properties

Attributes

```
key
modifier
name
onKeyDown
onKeyUp
```

Description

The `hotkey` block in the XML file defines the initial key and modifier for a hotkey in a Widget. Hotkeys are system level key triggers which allow Widgets to be accessed via the keyboard. So, for example, a search Widget could be coded to come to the foreground with a sequence like `Control+Shift+F2`.

Hotkey objects can also be created and destroyed dynamically via the JavaScript engine. This can be useful if you allow the user to customize your Widget's hotkeys.

Note that some key combinations are reserved by the system (e.g. `Control+Tab` on Windows or `Command+Tab` on Mac OS X). On Mac OS X, if more than one Widget or application uses the same hotkey then all receive a notification when the user presses those keys. On Windows, only the first to try gets the hotkey.

JavaScript

```
newObjectName = new HotKey()
delete newObjectName
```

Example

```
<hotkey name="hkey1">
  <key>F4</key>
  <modifier>control+shift</modifier>
  <onKeyDown>focusWidget();</onKeyDown>
</hotkey>
```

key

the name of the function key

Description

On Mac OS X hotkeys can be defined for any of the following keys:

```
Delete, End, Escape, ForwardDelete, F1, F2, F3, F4, F5, F6, F7,
F8, F9, F10, F11, F12, F13, F14, F15, F16, Help, Home, PageDown,
PageUp, Space, Tab
```

On Windows the following keys can be used:

```
UpArrow, DownArrow, LeftArrow, RightArrow, F1, F2, F3, F4, F5,
F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16, Insert,
ForwardDelete, Home, End, PageUp, PageDown, Help, Clear,
PrintScreen, ScrollLock, Pause, Enter, Return, Backspace, Delete,
Space, Tab, Escape
```

At least one modifier is required which is **Command** on Mac OS X and **Control** on Windows by default.

Hotkeys can also be defined for any letter or punctuation key but **two** modifiers must be specified in this case (to avoid confusing users by having familiar key combinations have unexpected effects).

JavaScript

myObjectName.key

Example

```
<hotkey name="hkey1">
  <key>F2</key>
</hotkey>

hkey1.key = "F2";
```

modifier

the modifier keys for the hotkey

Description

The modifier attribute can be any combination of:

On Mac OS X:	`command, control, option, shift`
On Windows:	`control, alt, shift`

A modifier is always used and is Command on Mac OS X or Control on Windows by default.

JavaScript

myObjectName.key

Example

```
<hotkey name="hkey1">
  <key>Home</key>
  <modifier>control+shift</modifier>
</hotkey>

hkey1.key = "F2";
```

name

the reference name of an hotkey

Description

The `name` attribute of the `hotkey` block defines the name of the key when referenced by JavaScript. Since the name is used for reference in code, it must not contain any spaces or non ASCII characters.

The name of an object cannot be changed once it's assigned.

When creating a dynamic object via JavaScript, you use the name of the variable to represent the new name of the object.

JavaScript

```
newObjectName = new HotKey()
```

Example

```
<hotkey name="hkey1">
  <key>F2</key>
</hotkey>

hkey1.key = "F2";
```

onKeyDown

the code that is activated when a hotkey is pressed

Description

The code to be run when the hotkey is pressed is specified with the `onKeyDown` attribute. Note, on Mac OS X it is generally best to attach key code to the `onKeyUp` action as that is what users expect. However, note that only `onKeyDown` fires on Windows.

JavaScript

```
newObjectName = new HotKey()
```

Example

```
<hotkey name="hkey1">
  <key>F10</key>
  <modifier>control</modifier>
  <onKeyDown>
    print("Hotkey " + system.event.keyString +
          " pressed");
  </onKeyDown>
</hotkey>
```

onKeyUp

the code that is activated when a hotkey is released

Description

The code to be run when the hotkey is released is specified with the `onKeyUp` attribute.

A common action to perform when a Widget's hotkey is pressed is `focusWidget()`.

JavaScript

```
newObjectName = new HotKey()
```

Example

```
<hotkey name="hkey1">
  <key>F10</key>
  <modifier>control</modifier>
  <onKeyUp>focusWidget();</onKeyUp>
</hotkey>
```

Windows Note

This trigger is not available on Windows.

<image>

block defining an image and associated default properties

Attributes

alignment
clipRect
colorize
contextMenuItems
fillMode
height
hAlign
hOffset
hRegistrationPoint
hslAdjustment
hslTinting
loadingSrc
missingSrc
name
onContextMenu
onDragDrop
onDragEnter
onDragExit
onImageLoaded
onMouseDown
onMouseEnter
onMouseExit
onMouseMove
onMouseUp
onMultiClick
opacity
remoteAsync
rotation
src
srcHeight
srcWidth
tileOrigin
useFileIcon
visible
vAlign
vOffset
vRegistrationPoint
width
window
zOrder

Description

The `image` block in the XML file defines the initial placement and mouse event scripts for a static image object in a Widget.

Image objects can also be created and destroyed dynamically via the JavaScript engine. This can be useful if you're creating a Widget that lists an indeterminate number of items.

When you create more than one dynamic object with the same name, only the last object created will receive property changing events via JavaScript. As a result you should make sure that you're calling each dynamic object a unique name so they can be referenced properly (using a JavaScript Array is often a good way to achieve this).

For more information on how to do this, look at how this works in our Stock Ticker Widget.

You can remove a dynamic object once you create it using the JavaScript `delete` instruction.

JavaScript

```
newObjectName = new Image()
delete newObjectName
```

Example

```
<image src="Images/Sun.png" name="sun1">
  <hOffset>250</hOffset>
  <vOffset>250</vOffset>
  <height>20</height>
  <width>30</width>
  <alignment>center</alignment>
</image>
```

alignment

direction the image is drawn from the defined origin point

Description

The alignment property of an image defines the initial horizontal alignment of the image being rendered. For example, an image with a `right` alignment will be drawn so that its right edge appears at the `hOffset` (see below). The default alignment is `left`.

Valid values are: `left, right` or `center`.

JavaScript

```
myObjectName.alignment
```

Example

```
<image src="button.png">
  <alignment>right</alignment>
</image>

myButton.alignment = "left";
```

clipRect

Controls what part of an image is visible.

Description

You can limit what part of an image is drawn by applying a clip rectangle to it. Coordinates are given in X, Y, width, height order.

JavaScript

myObjectName.clipRect

Example

If you have a 100x100 image and only want to show the area starting at 20, 20 and extending to 50, 50, you would add this tag to your image:

```
<image>
  ...
  <clipRect>20, 20, 30, 30</clipRect>
</image>
```

You can set or clear it at any time in Javascript as well:

```
myImage.clipRect = "20, 20, 30, 30";
myImage.clipRect = null;
```

You can clear by setting to an empty string as well as null.

Availability

Available in version 2.0 or later.

colorize

Controls the overall colorization of an image.

Description

Colorize essentially turns an image into grayscale and, given a color, maps the color onto the gray ramp. The image turns all shades of whatever color you specify. You can

do a sort of sepia tone effect with this, as well as various other very interesting and surprisingly useful things.

JavaScript

```
myObjectName.colorize
```

Example

```
<image>
  ...
  colorize>#993333</colorize>
</image>
```

To clear any colorization, just set it to null or an empty string in your code:

```
myImage.colorize = "";
```

Windows note: some 8-bit image formats (GIF) may not play well with colorization.

Availability

Available in version 2.0 or later. In version 3.0 or later, the format "r:0; g:0; b:0" can be used.

contextMenuItems

Specifies an array of context menu items.

Description

You can add items to the standard context menu that appears when the user right-clicks the mouse button on your Widget by adding contextMenuItems to your image. This tag is actually valid for text, textArea, and window objects as well. You can also dynamically build your context items by specifying some JavaScript to execute on your onContextMenu tag (see onContextMenu for more information).

You specify your items by including an array of menuItem objects. See the section on menuItem for more information about them.

JavaScript

```
myObjectName.contextMenuItems
```

Example

```
<image>
  ...
  <contextMenuItems>
    <menuItem title="Test" onSelect="beep();"/>
    <menuItem title="Another Test">
      <onSelect>alert( 'hello' );</onSelect>
    </menuItem>
  </contextMenuItems>
</image>
```

See the onContextMenu section for an example of building a context menu in JavaScript.

Availability

Available in version 2.0 or later.

fillMode

Controls how an image fills its area.

Description

Normally, an image will always stretch to fill the area it should occupy if you specify a width and height for the image. This tag allows you to override this and instead either stretch or tile the image by specifying either "tile" or "stretch". If you were to use tiling, you also might need to use the tileOrigin tag (described later).

If this tag is not specified, the default fill mode is "stretch".

JavaScript

myObjectName.fillMode

Example

```
<image>
  ...
  <fillMode>tile</fillMode>
  <tileOrigin>bottomLeft</tileOrigin>
</image>
```

You can also set these attributes of an image in JavaScript.

Availability

Available in version 2.0 or later.

height

how tall the image is made

Description

The `height` attribute controls the vertical dimension of the image. If none is specified, the image is drawn at its "natural" height (i.e. whatever the height of the source image is). If the height is larger than the natural height, the fillMode attribute controls what happens (the default is to stretch the image).

JavaScript

myObjectName.height

Example

```
<image src="button.png">
  <height>30</height>
</image>

myButton.height = 30;
```

hAlign

Control the horizontal alignment of an image.

Description

This is a synonym for the alignment tag. See the description of that tag for information.

Availability

Available in version 2.0 or later.

hOffset

the horizontal offset of an image

Description

The `hOffset` attribute of the image block defines the horizontal (left to right) offset for the image based on 0,0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther to the right the image will be drawn.

JavaScript

myObjectName.hOffset

Example

```
<image src="button.png">
  <hOffset>30</hOffset>
</image>
```

hRegistrationPoint

the horizontal offset for defining a registration point

Description

The `hRegistrationPoint` attribute of the image block defines the horizontal offset to use for placing and/or rotating the image. For example, if you have an 8x8 image, and you set the `hRegistrationPoint` to be 4, the image would draw centered based on the `hOffset` you gave it.

JavaScript

`myObjectName``.hRegistrationPoint`

Example

```
<image src="hourHand.png">
  <hRegistrationPoint>4</hRegistrationPoint>
</image>
```

Note

This attribute does not work correctly with hAlign/vAlign. Please use those tags if you are trying to align something to an edge or to center it and reserve this tag for rotation purposes.

hslAdjustment

Adjusts an image by adjusting it by HSL (Hue-Saturation-Lightness).

Description

HSL Adjustment works basically like you'd see in Photoshop's Adjust Hue/Saturation dialog when "Colorize" is not checked. You can shift the hue as well as increase color saturation and lightness. Hue can be adjusted from -180 to +180, saturation can be adjusted from -100 to +100. And lightness can be adjusted from -100 to +100. Adjusting the lightness upward may affect the saturation as well, so keep that in mind. You might use this for a throbbing effect where you need to shift all pixel hues evenly. This is also highly useful for doing things like changing an image to look selected by darkening it (decrease the lightness by about -55).

JavaScript

myObjectName.hslAdjustment

Example

```
<image>
  ...
  <hslAdjustment>-20, 5, 0</hslAdjustment>
</image>
```

To clear any adjustment, just set it to null or an empty string in your code:

```
myImage.hslAdjustment = "";
```

Windows note: some 8-bit image formats (GIF) may not play well with this feature.

Availability

Available in version 2.0 or later.

hslTinting

Colorize an image using HSL tweaking.

Description

HSL Tinting works basically like you'd see in Photoshop's Adjust Hue/Saturation dialog when "Colorize" is checked. You can set the hue and color saturation while adjusting lightness. Hue can be set from 0 to 360, saturation can be adjusted from 0 to +100. And lightness can be adjusted from -100 to +100. Adjusting the lightness upward may affect the saturation as well.

JavaScript

myObjectName.hslTinting

Example

```
<image>
  ...
  <hslTinting>-20, 5, 0</hslTinting>
</image>
```

To clear any tinting, just set it to null or an empty string in your code:

```
myImage.hslTinting = "";
```

Windows note: some 8-bit image formats (GIF) may not play well with this feature.

Availability

Available in version 2.0 or later.

loadingSrc

path to an image to display while an image loads asynchronously

Description

If a remote image is loaded asynchronously (by setting remoteAsync to true), you can display an alternate image in its place while the image is fetched from the server by setting this property. When the image is finally loaded, it replaces the loadingSrc automatically. If you wish to be informed when this happens, specify an action to happen via the onImageLoaded property.

JavaScript

```
myImage.loadingSrc = "images/loading.png";
```

Example

```
<image src="http://www.imadethisup.com/remote.jpg">
  <loadingSrc>images/loading.png</loadingSrc>
  <remoteAsync>true</remoteAsync>
</image>
```

Availability

Available in version 3.0 or later.

missingSrc

path to an image to display if the src cannot be loaded

Description

This property is used to customize the image that is displayed when an image's src attribute cannot be loaded. The Widget Engine has a default 'missing' image for this situation, but it might not be adequate for all situations. Typically you'd use this when loading a remote source which might not exist or be accessible.

JavaScript

```
myImage.missingSrc = "images/missing.png";
```

Example

```
<image src="http://www.imadethisup.com/notthere.jpg">
  <missingSrc>images/missing.png</missingSrc>
</image>
```

Availability

Available in version 3.0 or later.

name

the reference name of an image

Description

The `name` attribute of the `image` block defines the name of the image when referenced by JavaScript. Since the name is used for reference in code, it must not contain any spaces or non ASCII characters.

The name of an object cannot be changed once it's assigned.

When creating a dynamic object via JavaScript, you use the name of the variable to represent the new name of the object.

JavaScript

```
newObjectName = new Image()
```

Example

```
<image src="button.png">
  <name>myButton</name>
</image>

myButton.hOffset = 22;
```

onContextMenu

A context menu is about to appear. Time to add your items.

Description

The simplest way to specify context menu items that get added to the standard context menu for a Widget is to use the contextMenuItems tag in the XML. However, for those Widgets that need to build their items dynamically, the onContextMenu handler is your hook to do so. When the menu is about to be presented, this is called for all elements under the mouse from front to back in the view order until some view responds. When handling this, you should simply build your context menu items and set your contextMenuItems property to the array of items.

JavaScript

```
myImage.onContextMenu
```

Example

```
<onContextMenu>
var items = new Array();
items[0] = new MenuItem();
items[0].title = "This is the title";
items[0].enabled = false;
items[0].checked = true;
items[0].onSelect = "alert( 'you chose it!' );";

items[1] = new MenuItem();
items[1].title = "This is the second title";
items[1].onSelect = "beep();";

myImage.contextMenuItems = items;
</onContextMenu>
```

Availability

Available in version 2.0 or later.

onDragDrop

the script called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string is dragged from another application (e.g. the Finder) and dropped on the object.

In the `onDragDrop` action objects can access `system.event.data` to see what was dropped. This is an array of strings whose first element specifies what type of object was dropped.: `filenames`, `urls` or `string`. The remaining elements of the array are the items that were dropped.

JavaScript

`myObjectName``.onDragDrop`

Example

```
<image src="button.png">
  <name>myButton</name>
  <onDragDrop>
       if (system.event.data[0] == "filenames")
      {
        processDroppedFiles(system.event.data);
      }
  </onDragDrop>
</image>
```

```
<image src="button.png" name="myButton">
  <onDragDrop>dragCode.js</onDragDrop>
</image>

myButton.onDragDrop = "handleDragDrop();";
```

onDragEnter

the script that gets called when an item is dragged into the object

Description

The `onDragEnter` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object. This happens before the item is dropped (indeed it may not be dropped as the user can change their mind).

This is useful for triggering a visual change of the object to indicate to the user that the dragged object will be accepted or rejected if it is dropped. Information about the item being dragged is contained in `system.event.data` (see `onDragDrop` for details).

JavaScript

`myObjectName``.onDragEnter`

Example

```
<image src="well.png">
  <name>well</name>
  <onDragEnter>
        highlightDropTarget(well);
  </onDragEnter>
</image>

well.onDragEnter = "highlightDropTarget(well);";
```

onDragExit

the script that gets called when an item is dragged out of the object

Description

The `onDragExit` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object and then out again.

This is useful for undoing things that were done in `onDragEnter`.

JavaScript

myObjectName.onDragExit

Example

```
<image src="well.png">
  <name>well</name>
  <onDragExit>
        unhighlightDropTarget(well);
  </onDragExit>
</image>

well.onDragExit = "unhighlightDropTarget(well);";
```

onImageLoaded

called when an asynchronously loaded image is finally loaded

Description

If the src property of an image points to a remote image, and the remoteAsync property is set to true, images are fetched asynchronously. If you need to know when the image finally loads, you can use this action to get notified when the image is done loading. You might resize the image to the current native size of the image, or proportionally size it, for example.

JavaScript

myObjectName.onImageLoaded

Example

```
<image src="http://www.some.remote.image.com/image.png">
  <onImageLoaded>
        print( "image done loading" );
  </onImageLoaded>
</text>
myImage.onImageLoaded = "print( 'image loaded' );";
```

Availability

Available in version 3.0 or later.

onMouseDown

the script called when the mouse button is down inside the object

Description

The onMouseDown attribute of the image block is a wrapper for JavaScript code that

will execute when the user presses the mouse button down within the object.

This is useful for triggering a visual change of the object based on a pressed state.

JavaScript

myObjectName.onMouseDown

Example

```
<image src="button.png">
  <name>myButton</name>
  <onMouseDown>
        myButton.src = "buttonDown.png";
  </onMouseDown>
</image>

<image src="button.png" name="myButton">
  <onMouseDown>buttonCode.js</onMouseDown>
</image>

myButton.onMouseDown = "doButtonHighlight(myButton);";
```

onMouseEnter

the script that gets called when the mouse rolls into the object

Description

The `onMouseEnter` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has moved the cursor within the object.

This is useful for triggering a visual change of the object based on a rolled over state, or for showing an object that's hidden unless you're hovering over the Widget.

JavaScript

myObjectName.onMouseEnter

Example

```
<image src="button.png">
  <name>myButton</name>
  <onMouseEnter>
        myButton.src = "buttonOver.png";
  </onMouseEnter>
</image>

myButton.onMouseEnter = "handleMouseEnter(myButton);";
```

onMouseExit

the script that gets called when the mouse rolls out of an object

Description

The `onMouseExit` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has moved the cursor from within the object to outside the object.

This is useful for triggering a visual change of the object based on a rolled over state, or for re-hiding an object that's hidden unless you're hovering over the Widget.

avaScript

`myObjectName``.onMouseExit`

Example

```
<image src="button.png">
  <name>myButton</name>
  <onMouseExit>
        myButton.src = "button.png";
  </onMouseExit>
</image>

myButton.onMouseExit = "handleMouseExit(myButton);";
```

onMouseMove

the script that gets called when the mouse moves within an object

Description

The `onMouseMove` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user drags the mouse cursor within the bounds of an object. The current mouse position is available in the `system.event` object.

This is useful for moving an object around the Widget. The volume slider in the **iTunes Remote** Widget is implemented using this action.

JavaScript

`myObjectName``.onMouseMove`

Example

```
<image src="button.png">
  <name>myButton</name>
  <onMouseMove>
    print(system.event.x + ", " + system.event.y);
  </onMouseMove>
</image>

myButton.onMouseMove = "handleMouseMove(myButton);";
```

onMouseUp

the script that gets called on mouse up in an object

Description

The `onMouseUp` attribute of the `image` block is a wrapper for JavaScript code that will execute when the user has released the mouse after having it down within the object.

This is useful for triggering a visual change of the object based on a pressed state.

Please note that `onMouseUp` will trigger even if the mouse is not inside the object with the mouse is released. In order to create buttons which have correct mouse events you must employ the use of all four mouse event handlers in order to communicate the state of the mouse, and its intersection status (see the included Calendar Widget for an example of this).

JavaScript

`myObjectName``.onMouseUp`

Example

```
<image src="button.png">
  <name>myButton</name>
  <onMouseUp>
       myButton.src = "button.png";
  </onMouseUp>
</image>

myButton.onMouseUp = 'handleOnMouseUp(myButton);';
```

onMultiClick

A multiple click just occurred.

Description

You can easily trap double-clicks (or triple-clicks, etc.) using the onMultiClick handler.

This handler can be set on image, text, text area, and window objects. Whenever your onMultiClick handler is called, you can inspect system.event.clickCount to see what the value is. It will always be 2 (for a double-click) or greater.

It is also possible to inspect this system.event.clickCount in an onMouseUp handler as well in lieu of using onMultiClick. However, the advantage to using onMultiClick is that it does not interfere with window dragging the way that onMouseUp does, i.e. a mouse up handler on an image will prevent a window from being dragged if you click that image. If your image only needs to respond to a multi-clicks, you can use onMultiClick and the Widget will still be able to be dragged as usual.

```
<onMultiClick>
   if ( system.event.clickCount == 2 )
      alert( "Double Click!" );
</onMultiClick>
```

Availability

Available in version 2.0 or later.

opacity

how translucently the image displays

Description

The `opacity` attribute allows you to specify a value from 0 to 255 which controls the alpha value with which the image is rendered. An opacity of 0 is completely transparent (invisible) and has such side effects as preventing the object from reacting to mouse events. A value of 255 will render the image at its natural opacity.

Example

```
<image src="button.png">
  <name>myButton</name>
  <opacity>128</opacity>
</image>

myButton.opacity = 33;
```

remoteAsync

specifies whether remote images should be fetched asynchronously

Description

When set to true, remoteAsync tells the image object to load the image source in the background, allowing your Widget to do other things in the meantime. If you wish to specify an image to display while the image is being fetched, you can set the loadingSrc

property. If you wish to be informed when the image is finally loaded, set the onImageLoaded property with some appropriate Javascript.

Example

```
<image src="http://www.a.remote.server.com/image.png">
  <loadingSrc>images/loading.png</loadingSrc>
  <remoteAsync>true</remoteAsync>
</image>


myImage.remoteAsync = true;
```

Availability

Available in version 3.0 or later.

rotation

the degrees clockwise in which the image is rotated

Description

The `rotation` attribute of the image block defines by what degree, or fraction of a degree, the `image` is rotated.

Rotation can be used for any number of purposes, but the most obvious example is to accurately represent the hands of an analog clock.

JavaScript

`myObjectName``.rotation`

Example

```
<image src="hourHand.png">
    <rotation>
      180
    </rotation>
</image>
```

src

the path to the image being displayed

Description

The `src` attribute for the `image` block defines the source of the image. It takes a path to the file on your hard drive relative to the XML file of the Widget it's referenced from.

In version 2.0 and later, you can specify a URL as the source for the image.

JavaScript

myObjectName.src

Example

```
<image src="Resources/Buttons/button.png">
```

srcHeight

the original height of the source image

Description

The `srcHeight` attribute gives the original height of the image as it was when it was read from the disk before any resizing has been done. This attribute is read-only, setting it will have no effect.

JavaScript

myObjectName.srcHeight

Example

```
origHeight = myButton.srcHeight;
```

srcWidth

the original width of the source image

Description

The `srcWidth` attribute gives the original width of the image as it was when it was read from the disk before any resizing has been done. This attribute is read-only, setting it will have no effect.

JavaScript

myObjectName.srcWidth

Example

```
origWidth = myButton.srcWidth;
```

tileOrigin

Controls how an image is tiled.

Description

This tag is used with the fillMode tag, described above. If fillMode is set to "tile", an image is tiled into its width and height (assuming they are larger than the natural size of the image). This tag controls what corner of an image the tiling starts from. Valid values are "topLeft", "topRight", "bottomLeft", and "bottomRight". If this tag is not specified, the default is "topLeft".

JavaScript

myObjectName.tileOrigin

Example

```
<image>
  ...
  <fillMode>tile</fillMode>
  <tileOrigin>bottomLeft</tileOrigin>
</image>
```

They are also perfectly settable in Javascript.

Availability

Available in version 2.0 or later.

tooltip

the tooltip for an image object

Description

The `tooltip` attribute defines the text displayed in a popup tooltip window when the mouse cursor rests over an `image` object.

JavaScript

object.tooltip

Example

```
<image src="Example.png">
  <tooltip>Example tooltip</tooltip>
</image>
```

tracking

the cursor tracking style of the image

Description

The `tracking` attribute specifies whether the image's opacity should be used to determine the clickable portions of the image rather than the bounding rectangle. By default transparent parts of an image are not clickable but this can be changed by setting the `tracking` attribute to `rectangle` which makes the entire image respond to mouse clicks.

JavaScript

`myObjectName``.tracking`

Example

```
<tracking>rectangle</tracking>
```

See Also

`defaultTracking`

useFileIcon

retrieve the icon for the file

Description

The `useFileIcon` attribute for the `image` block specifies that this image will be initialized using the icon of the file specified in `src`. Note that, in this case, `src` can refer to any file not just one containing image data.

JavaScript

`myObjectName``.src`

Example

```
<image useFileIcon="true">
  <src>/Applications/iChat.app</src>
</image>
```

vAlign

Controls the vertical alignment of an image

Description

The vAlign property of an image defines how it is positioned vertically relative to its

vOffset. For example, an image with a `bottom` alignment will be drawn so that its bottom edge appears at the `vOffset` (see below). If this tag is not specified, the default value is `top`.

Valid values are: `top, bottom` or `center`.

JavaScript

myObjectName.vAlign

Example

```
<image src="button.png">
  <vAlign>bottom</vAlign>
</image>

myButton.vAlign = "bottom";
```

Availability

Available in version 2.0 or later.

visible

Controls the visibility of an image

Description

You can set the visible property of an image to show or hide it by setting it to true or false, respectively. This allows you to hide objects without affecting their opacity, or having to save off the current opacity to restore it later. The default visibility for any object if not specified is true.

JavaScript

myObjectName.visible

Example

```
<image src="button.png">
  <visible>false</visible>
</image>

myButton.visible = true;
```

Availability

Available in version 3.0 or later.

vOffset

the vertical offset of an image

Description

The `vOffset` attribute of the `image` block defines the vertical (top to bottom) offset for the image based on 0,0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther down the image will be drawn.

The vOffset specifies where the baseline of the text will be located. If you don't set this, you might not see the text, as the baseline will be zero.

JavaScript

object.vOffset

Example

```
<image src="button.png">
  <vOffset>20</vOffset>
</image>
```

vRegistrationPoint

the vertical offset for defining a registration point

Description

The `vRegistrationPoint` attribute of the `image` block defines the vertical offset to use for placing and/or rotating the image. For example, if you have an 8x8 image, and you set the `vRegistrationPoint` to be 4, the image would draw centered on the `vOffset` you gave it.

JavaScript

myObjectName.vRegistrationPoint

Example

```
<image src="hourHand.png">
  <vRegistrationPoint>36</vRegistrationPoint>
</image>
```

Note

This attribute does not work correctly with hAlign/vAlign. Please use those tags if you are trying to align something to an edge or to center it and reserve this tag for rotation purposes.

width

how wide the image is made

Description

The `width` attribute controls the horizontal dimension of the image. If none is specified, the image is drawn at its "natural" width (i.e. whatever the width of the source image is). If the width is larger than the natural width, the fillMode attribute controls what happens (the default is to stretch the image).

JavaScript

myObjectName.width

Example

```
<image src="button.png">
  <width>30</width>
</image>

myButton.width = 20;
```

window

The window to which this image belongs.

Description

You can specify the window an image belongs to by specifying its name in the XML or its variable in JavaScript. If you do not specify a window, the image is automatically attached to the first window found in the XML description of a Widget.

JavaScript

myObjectName.window

Example

```
<window name="fred" width="100" height="100"/>
<image src="button.png">
  <window>fred</window>
</image>

// Or in code
var myWind = new Window();
myImage.window = myWind;

// You can also specify it in the constructor for an image

var myImage = new Image( myWind );
```

Availability

Available in version 2.0 or later.

zOrder

the stacking order of an image

Description

The `zOrder` attribute of the `image` block defines the stacking order of the image. Objects with a higher `zOrder` are drawn on top of those with lesser `zOrder`s. Normally the `zOrder` is determined by the order in which objects are defined in the XML file with earlier objects being drawn under later ones but it can also be manipulated using JavaScript at runtime.

JavaScript

myObjectName.zOrder

Example

```
<image src="button.png">
  <zOrder>10</zOrder>
</image>

myButton.zOrder = customZOrder++;
```

<menuItem>

block to define a menu item

Attributes

```
checked
enabled
onSelect
title
```

Description

Menu items are used by the context menu arrays and handlers to provide extra items for the standard Widget context menu.

Availability

Available in version 2.0 or later.

checked

Specifies an item is checked

Description

This attribute merely specifies that the item should have a check mark next to it when displayed in the menu. If this attribute is not specified the default is false.

Example

```
<menuItem name="myItem"
   title="Widgetz R0x0r!!!11" checked="true"/>

// or in JavaScript
myItem.checked = true;
```

Availability

Available in version 2.0 or later.

enabled

Specifies an item is enabled

Description

This attribute merely specifies that the item should be enabled in the menu. If set to false, the item appears grayed out and is not choosable by the user. If this attribute is not specified, the default is true.

Example

```
<menuItem title="Recent Locations" enabled="false"/>

// or in JavaScript
myItem.enabled = false;
```

Availability

Available in version 2.0 or later.

onSelect

Specifies the JavaScript to run when an item is chosen.

Description

This attribute provides the action to carry out when an item is chosen from the context menu.

Example

```
<menuItem title="Recent Locations" enabled="false"
   onSelect="beep();"/>

// or in JavaScript
myItem.onSelect = "beep();";
```

Availability

Available in version 2.0 or later.

title

Specifies the text of a menu item.

Description

This attribute provides the text to display for a menu item.

Example

```
<menuItem title="I am the title"/>

// or in JavaScript
myItem.title = "Choose me!";
```

Availability

Available in version 2.0 or later.

<preference>

block defining a preference setting and associated properties

Attributes

```
defaultValue
description
directory
extension
file
group
hidden
kind
maxLength
minLength
name
notSaved
option
optionValue
secure
style
ticks
tickLabel
title
type
value
```

Description

The preference block defines a block of information that is to be stored by the Widget between open/closed sessions, as well as user entered data.

There are two preferences that are provided automatically:

`windowLevel`	the level the Widget window displays at on the user's screen `floating`, `topMost`, `normal`, `below` or `desktop`
`windowOpacity`	the opacity of the Widget's window

These preferences allow the user to control how the Widget displays on their desktop. If you want to provide this functionality yourself, all you have to do is call your preferences the same names, `windowLevel` and `windowOpacity`. If you want to disable this feature, just define two preferences as follows in your Widget:

```
<preference name="windowLevel">
  <hidden>true</hidden>
</preference>

<preference name="windowOpacity">
  <hidden>true</hidden>
</preference>
```

defaultValue

the default value of the preference

Description

The `defaultValue` attribute of the preference block specifies what the value should be by default. This makes it possible to pre-populate your preferences as well as have placeholders until the user enters proper data. This is the value your JavaScript code will see if it accesses the preferences before the user has customized them.

Example

```
<preference name="colorPref">
  <defaultValue>red</defaultValue>
</preference>

colorPref.defaultValue = "red";
```

description

the descriptive text displayed in the preference panel

Description

The description attribute of the preference block defines the descriptive text that goes underneath a preference when being displayed in the preference panel's user interface.

It's optional, but highly recommended, to explain the preference and its usage to your users.

Example

```
<preference name="colorPref">
  <description>
       Enter the desired color
  </description>
</preference>
```

directory

the default starting directory for a preference of type `selector`

Description

Preferences of type `selector` can have their starting directory set using this attribute.

Example

```
<preference>
  <type>selector</type>
  <style>open</style>
  <directory>~/Documents</directory>
</preference>
```

extension

the kind of file for a preference of type `selector`

Description

Preferences of type `selector` displaying an `open` system dialog can be limited to returning only files with certain extensions using this attribute.

Example

```
<preference>
  <type>selector</type>
  <style>open</style>
  <extension>.jpg</extension>
  <extension>.gif</extension>
  <extension>.png</extension>
</preference>
```

file

the default filename for a preference of type `selector`

Description

Preferences of type `selector` can have their default filename set using this attribute.

Example

```
<preference>
  <type>selector</type>
  <style>save</style>
  <file>~/Documents/myfile.foo</file>
</preference>
```

group

Group this preference belongs to

Description

As of version 2.0, the Preferences dialog for a Widget is broken into groups, and displayed in a multi-pane dialog. This attribute tells the Widget Engine which preference group this particular preference belongs to. If this attribute is not specified, the preference is rolled into a 'General' group automatically.

Example

```
<preference>
  <type>selector</type>
  <style>save</style>
  <group>my_group</group>
</preference>
```

The above example assumes that you've defined an appropriate preference group called my_group in your XML someplace. See the section on preferenceGroup for more information.

Availability

Available in version 2.0 or later.

hidden

is the preference presented to the user

Description

If a preference has the `hidden` attribute, the ability to edit or see that preference is not offered to the end user. The preference can still be manipulated in JavaScript but it isn't displayed on the Widget Preferences dialog. If a Widget has only hidden preferences, the user is not offered the Widget Preferences option on the context menu. Hidden preferences are often used to implement settings the user makes using controls on the Widget rather than by opening the Widget Preferences dialog.

Example

```
<preference name="colorPref">
  <hidden>true</hidden>
  <type>text</type>
  <defaultValue>red</defaultValue>
</preference>
```

kind

the kind of item for a preference of type `selector`

Description

Preferences of type `selector` displaying an `open` system dialog can be limited to `files`, `folders` or `both` using this attribute.

Example

```
<preference>
  <type>selector</type>
  <style>open</style>
  <kind>folders</kind>
</preference>
```

maxLength

the maximum value of a `slider` preference

Description

Used only for slider preferences currently, this is the maximum value the slider can represent.

Example

```
<preference>
  <maxLength>200</maxLength>
</preference>
```

minLength

the minimum value of a `slider` preference

Description

Used only for slider preferences currently, this is the minimum value the slider can represent.

Example

```
<preference>
  <minLength>1</minLength>
</preference>
```

name

the reference name of a preference

Description

The name attribute of the preference block defines the name of the preference as reference by JavaScript. Since the name is used for reference in code, it should not contain any spaces or non ASCII characters.

Example

```
<preference>
  <name>colorPref</name>
</preference>
```

notSaved

prevents a preference value from being automatically saved

Description

The `notSaved` attribute causes the preference not to be automatically saved in the user's preference file for the Widget. This can be useful if you want to display a control on the preferences panel but handle the value returned in code. In a way, this attribute is the opposite of `hidden`.

Example

```
<preference>
  <notSaved>true</notSaved>
</preference>
```

option

the choices for a preference of type `popup`

Description

Preferences of type `popup` are displayed as a popup menu in the Widget Preferences dialog. Several `option` attributes may be used to provide a set of choices for a popup menu.

Specifying the string `(-` for an option causes a separator to be displayed at that point in the popup (which cannot be selected by the user).

Example

```
<preference name="colorPref">
  <type>popup</type>
  <option>Red</option>
  <option>White</option>
  <option>(-</option>
  <option>Blue</option>
</preference>
```

optionValue

the values corresponding to the choices for a preference of type popup

Description

If you want the value returned when an `option` (see above) is choosen to be different, specify an `optionValue` for each `option`. There should be an `optionValue` for every `option` (note that if you use a "separator" `option`, see above, you will need to give it a corresponding dummy `optionValue` even though this value can never be returned).

Example

```
<preference name="colorPref">
  <type>popup</type>
  <option>Red</option>
  <optionValue>#FF0000</optionValue>
  <option>White</option>
  <optionValue>#FFFFFF</optionValue>
  <option>(-</option>
  <optionValue>none</optionValue>
  <option>Blue</option>
  <optionValue>#0000FF</optionValue>
</preference>
```

secure

specify that an attribute value should be saved securely

Description

Any type of preference can be `secure` which causes it's data to be saved in a manner that cannot easily be read. This is useful for saving items such as passwords. `text` preferences additionally display a "password" style user interface (bullets appear instead of typed characters).

If the code that reads a previously `secure` preference is changed to be non-secure, the value of the preference is reset to the `defaultValue`.

Example

```
<preference>
  <type>text</type>
  <secure>yes</secure>
</preference>
```

style

the dialog style for a preference of type `selector`

Description

Preferences of type `selector` can display either `open` or `save` system dialogs. The former allows the user to choose existing files, the latter a place to save or create a new file.

Example

```
<preference>
  <type>selector</type>
  <style>open</style>
</preference>
```

ticks

the number of tick marks to display on a `slider` preference

Description

To make the slider display tick marks use the `ticks` attribute. A side effect of this is that the slider also only returns values corresponding to the ticks.

For sliders, the `minLength` and `maxLength` attributes define the minimum and maximum values that can be set. The first tick will correspond to `minLength` and the last to `maxLength`.

Example

```
<preference>
  <type>slider</type>
  <ticks>10</ticks>
  <minLength>0</minLength>
  <maxLength>100</maxLength>
</preference>
```

tickLabel

labels for `slider` preferences

Description

To make the slider display labels under the track specify one or many `tickLabels`. The labels are evenly distributed along the length of the slider.

Example

```
<preference>
  <type>slider</type>
  <tickLabel>One</tickLabel>
  <tickLabel>Volume</tickLabel>
  <tickLabel>Eleven</tickLabel>
</preference>
```

title

the label displayed in the preference panel

Description

The title attribute of the preference block defines the label title that is displayed to the user via the built in preference interface.

Example

```
<preference>
  <title>Color:</title>
</preference>
```

type

the type of data and control to display

Description

The type attribute of the preference block defines what type of user interface object is used to display the data choices.

Type can be one of:

`checkbox` display a checkbox to gather yes/no input. The value returned to the Widget is either `0` or `1`.

`color` display a color swatch and allow colors to be picked. The value returned to the Widget is a standard color specifier like `#123456`.

`font`	display a font name and allow a font to be picked from those available on the system. The font attribute of a Text object can be set to the value returned.
`hotkey`	display a hotkey and its modifier and allow alternative key combinations to be chosen. The value returned can be used to set the modifier and key for a Hotkey object.
`popup`	display a choice and allow alternatives to be chosen from a Widget specified list. A string is returned (either one of the `options` or, if specified, one of the `optionValues`).
`selector`	display a file name and allow other file names to be chosen. The value returned to the Widget is the fully qualified pathname of the file (a web style path with / separators).
`slider`	display a slider and allow numeric values to be input. A numeric value is returned.
`text`	a standard text field in which the user can type text. A string is returned (see the `secure` attribute for information on displaying password style text fields).

Example

```
<preference>
  <type>checkbox</type>
</preference>
```

value

the current value of the preference

Description

The value attribute of the preference contains the current value assigned to the preference. This may have just been entered by the user or may have been read from the Widget's preference file at startup time.

Note that the value attribute is always treated as a string even if it contains a number. If you want to use a preference value as a number, use the appropriate conversion routine when accessing it. For instance:

```
numberOfItems = int(preferences.numItems.value) + 1;
```

<preferenceGroup>

A group to organize preferences

Attributes

```
name
icon
order
title
```

Description

Preference groups allow you to organize your preferences when displayed in the Preferences dialog. The dialog is displayed as a multi-pane dialog in version 2.0 and later. You define your groups using preferenceGroups and then set the group attribute of each preference you want in a particular group.

Availability

Preference groups were introduced in version 2.0

name

the name of this group

Description

This attribute defines the group name. This name is merely an identifier and should be unique among all preference groups. When defining a preference item that belongs to a group, it is this name you use to identify the group to which it belongs. It should not be confused with the title attribute, which is the user-visible name that is shown in the preferences window toolbar.

Example

```
<preferenceGroup name="colors" title="Colors"/>
```

Availability

Available in version 2.0 or later.

icon

The image to display for the group

Description

You can specify the image that is displayed in the dialog to represent your group. This image must be 32x32 maximum at present. If you do not specify an icon, a default one

will be provided for your group automatically.

Example

```
<preferenceGroup icon="Resources/myPrefIcon.png"/>
```

Availability

Available in version 2.0 or later.

order

Defines which order your groups appear

Description

This property is used to help you control the order in which your preference groups appear in the dialog. The numbering is completely up to you, but the lowest number is displayed in the leftmost position.

Example

```
<preferenceGroup>
  title="First Group"
  order="0"
</preferenceGroup>
<preferenceGroup>
  title="Second Group"
  order="1"
</preferenceGroup>
```

Availability

Available in version 2.0 or later.

title

The title of your preference group

Description

This property defines what text should appear below the icon of your preference group in the dialog. These titles should generally be short and one or two words long.

Example

```
<preferenceGroup>
  title="General"
  order="0"
</preferenceGroup>
<preferenceGroup>
  title="Special"
  order="1"
</preferenceGroup>
```

Availability

Available in version 2.0 or later.

<scrollbar>

specifies a scroll bar object

Properties

```
autoHide
hAlign
height
hOffset
max
min
onValueChanged
opacity
orientation
pageSize
thumbColor
vAlign
value
visible
width
window
zOrder
```

autoHide

specifies whether a scroll bar should hide when there's nothing to scroll

Description

This property is used to set the scroll bar into a mode where when there is nothing to scroll the scroll bar will hide. The default of this property is false. If the scroll bar is not set to auto-hide it instead becomes 50% transparent when there is nothing to scroll.

Example

```
<scrollbar>
  <autoHide>true</autoHide>
</scrollbar>

myScrollbar.autoHide = true;
```

Availability

Available in version 3.0 or later.

hAlign

control the horizontal alignment of an object

Description

The hAlign property of an object defines the initial horizontal alignment with respect to its hOffset property. For example, an object with `right` alignment will be drawn so that its right edge appears at the `hOffset`. The default alignment is `"left"`.

Valid values are: `"left", "right"` or `"center"`.

JavaScript

myObjectName.alignment

Example

```
<scrollbar>
  <alignment>right</alignment>
</scrollbar>

myScrollbar.alignment = "left";
```

Availability

Available in version 3.0 or later.

height

the height of the object

Description

The `height` attribute controls the vertical dimension of an object. For horizontal scroll bars, you do not need to specify a height. The size of the scroll bar images will determine it. In general, you should not specify a height and instead ask the scrollbar what its height is to layout your interface. This will insulate you from changes in the scroll bar appearance in the future. Obviously, for a vertical scroll bar, you must set the height to whatever your interface demands.

JavaScript

myObjectName.height

Example

```
<scrollbar>
  <height>300</height>
</scrollbar>

myScrollbar.height = 300;
```

Availability

Available in version 3.0 or later.

hOffset

the horizontal offset of an object

Description

The `hOffset` attribute of an object defines the horizontal (left to right) offset for the image based on 0,0 being the upper left hand corner of the its parent view (superview). The greater the value assigned, the farther to the right the object will appear.

JavaScript

myObjectName.hOffset

Example

```
<scrollbar>
  <hOffset>30</hOffset>
</scrollbar>
```

Availability

Available in version 3.0 or later.

max

the maximum value of a scroll bar

Description

The `max` property defines the maximum value of a scroll bar. Together with min, it defines the range of values the scroll bar can have. Values are pinned between min and max.

If you merely attach a scroll bar to a frame, you would normally never need to deal with this property. It all gets set up automatically in that situation.

If you do have a standalone scroll bar and wish to set the min, you must use the `setRange()` function. You cannot modify this property directly in Javascript. However, you can specify it in the XML for a scrollbar.

JavaScript

myObjectName.max

Example

```
<scrollbar>
  <min>0</min>
  <max>100</max>
</scrollbar>
```

Availability

Available in version 3.0 or later.

min

the minimum value of a scroll bar

Description

The `min` property defines the minimum value of a scroll bar. Together with max, it defines the range of values the scroll bar can have. Values are pinned between min and max.

If you merely attach a scroll bar to a frame, you would normally never need to deal with this property. It all gets set up automatically in that situation.

If you do have a standalone scroll bar and wish to set the max, you must use the `setRange()` function. You cannot modify this property directly in Javascript. However, you can specify it in the XML for a scrollbar.

JavaScript

myObjectName`.min`

Example

```
<scrollbar>
  <min>0</min>
  <max>100</max>
</scrollbar>
```

Availability

Available in version 3.0 or later.

onValueChanged

called when a scroll bar's value changes

Description

This property contains the Javascript that is called whenever a scroll bar's value

changes.

If you merely attach a scroll bar to a frame, you would normally never need to specify anything for this property. The frame will react to the scroll bar being dragged, etc. automatically.

JavaScript

myObjectName.min

Example

```
<scrollbar name="sb">
  <onValueChanged>
       print( "Whoa! value is now " + sb.value );
  </onValueChanged>
</scrollbar>
```

Availability

Available in version 3.0 or later.

opacity

the opacity of an object

Description

The `opacity` property allows you to specify a value from 0 to 255 which controls the alpha value with which the object is rendered. An opacity of 0 is completely transparent (invisible) and has such side effects as preventing the object from reacting to mouse events. A value of 255 will render the image at its natural opacity.

Example

```
<scrollbar>
  <opacity>128</opacity>
</scrollbar>

myScrollbar.opacity = 33;
```

Availability

Available in version 3.0 or later.

orientation

the orientation of a scrollbar

Description

The `orientation` property allows you to specify the orientation of a scrollbar. It's possible values are "vertical" and "horizontal". The default is "vertical".

Example

```
<scrollbar>
  <orientation>vertical</orientation>
</scrollbar>

myScrollbar.orientation = "horizontal";
```

Availability

Available in version 3.0 or later.

pageSize

the page size of a scrollbar

Description

The `pageSize` property is used to help determine the size of the thumb for a proportional scroll bar. Typically, this is the height of the view being scrolled (assuming a vertical scroll bar).

If you have attached a scroll bar to a Frame for scrolling, you do not need to deal with this property directly. It is all set up and handled automatically.

Example

```
<scrollbar>
  <pageSize>140</pageSize>
</scrollbar>

myScrollbar.pageSize = 100;
```

Availability

Available in version 3.0 or later.

thumbColor

the thumb color of a scrollbar

Description

The `thumbColor` property is used to control the tint of the thumb. The default thumb in the standard scroll bar is a medium gray. The color you specified is applied via colorization.

To clear the current color completely, you can set it to null in Javascript.

Example

```
<scrollbar>
  <thumbColor>#333366</thumbColor>
</scrollbar>

myScrollbar.thumbColor = "#333366";
myScrollbar.thumbColor = null;
```

Availability

Available in version 3.0 or later.

vAlign

controls the vertical alignment of an object

Description

The vAlign property of an object defines how it is positioned vertically relative to its vOffset. For example, an image with a `bottom` alignment will be drawn so that its bottom edge appears at the `vOffset`. If this tag is not specified, the default value is `"top"`.

Valid values are: `"top", "bottom"` or `"center"`.

JavaScript

`myObjectName``.vAlign`

Example

```
<scrollbar>
  <vAlign>bottom</vAlign>
</scrollbar>

myScrollbar.vAlign = "bottom";
```

Availability

Available in version 3.0 or later.

value

the current value of the scroll bar

Description

The value property contains the current value of the scroll bar. You can also use it to set the value to some value between the scroll bar's minimum and maximum values. If you specify a value less than the minimum or greater than the maximum, the value is pinned to those values.

JavaScript

```
myObjectName.value
```

Example

```
<scrollbar>
  <min>-100</min>
  <max>100</max>
  <value>0</value>
</scrollbar>

myScrollbar.value = 10;
```

Availability

Available in version 3.0 or later.

visible

controls the visibility of an image

Description

You can set the visible property of an image to show or hide it by setting it to true or false, respectively. This allows you to hide objects without affecting their opacity, or having to save off the current opacity to restore it later. The default visibility for any object if not specified is true.

JavaScript

```
myObjectName.visible
```

Example

```
<scrollbar>
  <visible>false</visible>
</scrollbar>

myScrollbar.visible = true;
```

Availability

Available in version 3.0 or later.

vOffset

the vertical offset of an image

Description

The `vOffset` property defines the vertical (top to bottom) offset for the object based on 0, 0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther down the object will appear.

JavaScript

object`.vOffset`

Example

```
<scrollbar>
  <vOffset>20</vOffset>
</scrollbar>
```

Availability

Available in version 3.0 or later.

width

the width of an object

Description

The `width` attribute controls the horizontal dimension of an object. For vertical scroll bars, you do not need to specify a width. The size of the scroll bar images will determine it. In general, you should not specify a width and instead ask the scrollbar what its width is to layout your interface. This will insulate you from changes in the scroll bar appearance in the future. Obviously, for a horizontal scroll bar, you must set the width to whatever your interface demands.

JavaScript

myObjectName`.width`

Example

```
<scrollbar>
  <width>300</width>
</scrollbar>
```

```
myScrollbar.width = 200;
```

Availability

Available in version 3.0 or later.

window

the window to which this object belongs.

Description

You can specify the window an object belongs to by specifying its name in the XML or its variable in JavaScript. If you do not specify a window, the object is automatically attached to the first window found in the XML description of a Widget.

JavaScript

myObjectName.window

Example

```
<window name="fred" width="100" height="100"/>
<scrollbar>
  <window>fred</window>
</scrollbar>

// Or in code
var myWind = new Window();
myScrollbar.window = myWind;

// You can also specify it in the constructor

var myFrame = new ScrollBar( myWind );
```

Availability

Available in version 3.0 or later.

zOrder

the stacking order of an object

Description

The `zOrder` property defines the stacking order of an object. Objects with a higher `zOrder` are drawn on top of those with lesser `zOrder`s. Normally the `zOrder` is determined by the order in which objects are defined in the XML file with earlier objects being drawn under later ones but it can also be manipulated using JavaScript at runtime.

JavaScript

myObjectName.zOrder

Example

```
<scrollbar>
  <zOrder>10</zOrder>
</scrollbar>

myScrollbar.zOrder = customZOrder++;
```

Availability

Available in version 3.0 or later.

<shadow>

specifies shadow parameters for an object

Attributes

```
color/colour
hOffset
opacity
vOffset
```

Description

The shadow element is currently only used in about-box text items. It allows you to set a solid shadow on an item with a certain color and opacity. The h and vOffsets you specify are offsets from the object you are shadowing (currently, text).

Availability

Available in version 2.1 or later.

color/colour

the color of the shadow

Description

Specifies the color of the shadow to cast.

Example

```
<shadow color="#333333"/>
```

Availability

Available in version 2.1 or later.

hOffset

the horizontal offset of the shadow

Description

Specifies the horizontal offset from the original object to cast the shadow. A value of 1 would mean the shadow was offset 1 pixel to the right of the object.

Example

```
<shadow hOffset="1"/>
```

Availability

Available in version 2.1 or later.

opacity

the opacity of the shadow

Description

Specifies the opacity of the shadow from 0 to 255, where 0 is completely transparent and 255 is completely opaque.

Example

```
<shadow opacity="255"/>
```

Availability

Available in version 2.1 or later.

vOffset

the vertical offset of the shadow

Description

Specifies the vertical offset from the original object to cast the shadow. A value of 1 would mean the shadow was offset 1 pixel to below the object.

Example

```
<shadow vOffset="1"/>
```

Availability

Available in version 2.1 or later.

<text>

block defining a text object and associated default properties

Attributes

alignment
bgColor
bgOpacity
color
contextMenuItems
data
font
height
hAlign
hOffset
name
onContextMenu
onDragDrop
onDragEnter
onDragExit
onKeyUp
onKeyDown
onMouseDown
onMouseEnter
onMouseExit
onMouseMove
onMouseUp
onMultiClick
opacity
shadow
size
style
truncation
visible
vOffset
width
window
zOrder

Description

The `text` block in the XML file defines the initial placement and mouse event scripts for a static text object in a Widget.

Text objects can also be created and destroyed dynamically via the JavaScript engine. This can be useful if you're creating a Widget that lists an indeterminate number of items.

When you create more than one dynamic object with the same name, only the last

object created will receive property changing events via JavaScript. As a result you should make sure that you're calling each dynamic object a unique name so they can be referenced properly (using a JavaScript Array is often a good way to achieve this). For more information on how to do this, look at how this works in our Stock Ticker Widget.

You can remove a dynamic object once you create it using the JavaScript `delete` instruction.

JavaScript

```
newObjectName = new Text()
delete newObjectName
```

alignment

direction the text draws from the defined origin point

Description

The alignment property of the text block defines the initial horizontal alignment of the text being rendered.

Valid values are: `left, right` or `center`.

Example

```
<text data="Example Text">
  <alignment>right</alignment>
</text>
```

bgColor

the background color of a text object

Description

Sets the color of the background of a text object. Colors are specified as browser style hex RGB triplets. For example:

```
#FF0000
```

is red.

Note that this property is closely linked with the `bgOpacity` property – both should be set to get a visible result.

Example

```
<text data="Example Text">
  <bgColor>#FFFFFF</bgColor>
  <bgOpacity>150</bgOpacity>
</text>
```

bgOpacity

the opacity of the background of a text object

Description

Set the opacity of the background of a text object. Opacities are specified as a number between 0 and 255.

Note that this property is closely linked with the `bgColor` property – both should be set to get a visible result.

Example

```
<text data="Example Text">
  <bgColor>#FFFFFF</bgColor>
  <bgOpacity>150</bgOpacity>
</text>
```

color

color that the text object draws in

Description

Set the color of the text object. Colors are specified as browser style hex RGB triplets. For example:

```
#FF0000
```

is red.

Example

```
<text data="Example Text">
  <color>#F42DA6</color>
</text>
```

contextMenuItems

Specifies an array of context menu items.

Description

You can add items to the standard context menu that appears when the user right-clicks the mouse button on your Widget by adding contextMenuItems to your text object. This tag is actually valid for image, textArea, and window objects as well. You can also dynamically build your context items by specifying some JavaScript to execute on your onContextMenu tag (see onContextMenu for more information).

You specify your items by including an array of menuItem objects. See the section on menuItem for more information about them.

JavaScript

myObjectName.contextMenuItems

Example

```
<text>
...
  <contextMenuItems>
    <menuItem title="Test" onSelect="beep();"/>
    <menuItem title="Another Test">
      <onSelect>alert( 'hello' );</onSelect>
    </menuItem>
  </contextMenuItems>
</text>
```

See the onContextMenu section for an example of building a context menu in JavaScript.

Availability

Available in version 2.0 or later.

data

the text that the text object draws

Description

The text to be displayed. Note that any new lines or carriage returns in the text will be converted to spaces before display.

Example

```
<text>
  <data>Example Text</data>
</text>
```

font

the font that the text object draws using

Description

The name of the font to be used to render the text. If the specified font cannot be found then the default System Font is used. Separate multiple font names with commas to specify *fallbacks* (fonts that used in the event a preceding font isn't found on the user's system).

Example

```
<text data="Example Text">
  <font>Palatino</font>
</text>

text1.font = "Monaco, Courier";
```

hAlign

Control the horizontal alignment of a text object.

Description

This is a synonym for the alignment tag. See the description of that tag for information.

Availability

Available in version 2.0 or later.

height

how tall the text object is made

Description

The `height` attribute controls the vertical dimension of the text object. If none is specified, the object occupies just enough space to fit the text (rendered in the specified font, size, etc). It is not usually necessary to specify the `height` of a text object.

JavaScript

`myObjectName``.height`

Example

```
<text data="Example Text">
  <height>30</height>
</text>
```

```
myLabel.height = 30;
```

hOffset

the horizontal offset of a text object

Description

The `hOffset` attribute of the text block defines the horizontal (left to right) offset for the text based on 0,0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther to the right the text will be drawn.

JavaScript

myObjectName`.hOffset`

Example

```
<text data="Example Text">
  <hOffset>30</hOffset>
</text>
```

name

the reference name of a text object

Description

The `name` attribute of the `text` block defines the name of the text object when referenced by JavaScript. Since the name is used for reference in code, it must not contain any spaces or non ASCII characters.

The name of an object cannot be changed once it's assigned.

When creating a dynamic object via JavaScript, you use the name of the variable to represent the new name of the object.

JavaScript

newObjectName `= new Image()`

Example

```
<text data="Example Text">
  <name>myText</name>
</text>

myText.hOffset = 22;
```

onContextMenu

A context menu is about to appear. Time to add your items.

Description

The simplest way to specify context menu items that get added to the standard context menu for a Widget is to use the contextMenuItems tag in the XML. However, for those Widgets that need to build their items dynamically, the onContextMenu handler is your hook to do so. When the menu is about to be presented, this is called for all elements under the mouse from front to back in the view order until some view responds. When handling this, you should simply build your context menu items and set your contextMenuItems property to the array of items.

JavaScript

myText.onContextMenu

Example

```
<onContextMenu>
var items = new Array();
items[0] = new MenuItem();
items[0].title = "This is the title";
items[0].enabled = false;
items[0].checked = true;
items[0].onSelect = "alert( 'you chose it!' );";

items[1] = new MenuItem();
items[1].title = "This is the second title";
items[1].onSelect = "beep();";

myText.contextMenuItems = items;
</onContextMenu>
```

Availability

Available in version 2.0 or later.

onDragDrop

the script called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string is dragged from another application (e.g. the Finder) and dropped on the object.

In the `onDragDrop` action objects can access `system.event.data` to see what was dropped. This is an array of strings the first element of which tells you what kind of thing was dropped: `filenames`, `urls` or `string`. The remaining elements of the array

are the items that were dropped.

JavaScript

myObjectName.onDragDrop

Example

```
<text data="Drop Stuff Here">
  <name>dropper</name>
  <onDragDrop>
       if (system.event.data[0] == "filenames")
       {
         processDroppedFiles(system.event.data);
       }
  </onDragDrop>
</text>

<text data="Drop Stuff Here">
  <onDragDrop>dragCode.js</onDragDrop>
</text>

dropper.onDragDrop = "handleDragDrop();";
```

onDragEnter

the script that gets called when an item is dragged into the object

Description

The `onDragEnter` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object. This happens before the item is dropped (indeed it may not be dropped as the user can change their mind).

This is useful for triggering a visual change of the object to indicate to the user that the dragged object will be accepted or rejected if it is dropped. Information about the item being dragged is contained in `system.event.data` (see `onDragDrop` for details).

JavaScript

myObjectName.onDragEnter

Example

```
<text data="Drop Stuff Here">
  <name>dropper</name>
  <onDragEnter>
       highlightDropTarget(dropper);
  </onDragEnter>
</text>
```

```
well.onDragEnter = "highlightDropTarget(well);";
```

onDragExit

the script that gets called when an item is dragged out of the object

Description

The `onDragExit` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object and then out again.

This is useful for undoing things that were done in `onDragEnter`.

JavaScript

myObjectName`.onDragExit`

Example

```
<text data="Drop Stuff Here">
  <name>dropper</name>
  <onDragExit>
        unhighlightDropTarget(dropper);
  </onDragExit>
</text>

dropper.onDragExit = "unhighlightDropTarget(dropper);";
```

onMouseDown

the script called when the mouse button is down inside the object

Description

The `onMouseDown` attribute of a `text` block is a wrapper for JavaScript code that will execute when the user presses the mouse button down within the object.

This is useful for triggering a visual change of the object based on a pressed state.

JavaScript

myObjectName`.onMouseDown`

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <onMouseDown>
       myLabel.color = "#FF0000";
  </onMouseDown>
</text>

<text data="Example Text" name="myLabel">
  <onMouseDown>labelCode.js</onMouseDown>
</text>

myLabel.onMouseDown = "doLabelHighlight(myLabel);";
```

onMouseEnter

the script that gets called when the mouse rolls into the object

Description

The `onMouseEnter` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has moved the cursor within the object.

This is useful for triggering a visual change of the object based on a rolled over state, or for showing an object that that's hidden unless you're hovering over the Widget.

JavaScript

`myObjectName``.onMouseEnter`

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <onMouseEnter>
       myLabel.color = "#EEEEEE";
  </onMouseEnter>
</text>

myLabel.onMouseEnter = "handleMouseEnter(myLabel);";
```

onMouseExit

the script that gets called when the mouse rolls out of an object

Description

The `onMouseExit` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has moved the cursor from within the object to outside the

object.

This is useful for triggering a visual change of the object based on a rolled over state, or for re-hiding an object that that's hidden unless you're hovering over the Widget.

JavaScript

myObjectName.onMouseExit

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <onMouseExit>
       myLabel.color = "#FFFFFF";
  </onMouseExit>
</text>

myLabel.onMouseExit = "handleMouseExit(myLabel);";
```

onMouseMove

the script that gets called when the mouse moves within an object

Description

The `onMouseMove` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user drags the mouse cursor within the bounds of an object. The current mouse position is available in the `system.event` object.

This is useful for moving an object around the Widget. The volume slider in the **iTunes Remote** Widget is implemented using this action.

JavaScript

myObjectName.onMouseMove

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <onMouseMove>
    print(system.event.x + ", " + system.event.y);
  </onMouseMove>
</text>

myLabel.onMouseMove = "handleMouseMove(myLabel);";
```

onMouseUp

the script that gets called on mouse up in an object

Description

The `onMouseUp` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has released the mouse after having it down within the object.

This is useful for triggering a visual change of the object based on a pressed state.

Please note that `onMouseUp` will trigger even if the mouse is not inside the object with the mouse is released. In order to create buttons which have correct mouse events you must employ the use of all four mouse event handlers in order to communicate the state of the mouse, and its intersection status (see the included Calendar Widget for an example of this).

JavaScript

myObjectName.onMouseUp

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <onMouseUp>
        myLabel.color = "#FFFFFF";
  </onMouseUp>
</text>

myLabel.onMouseUp = 'handleOnMouseUp(myLabel);';
```

onMultiClick

A multiple click just occurred.

Description

You can easily trap double-clicks (or triple-clicks, etc.) using the onMultiClick handler. This handler can be set on image, text, text area, and window objects. Whenever your onMultiClick handler is called, you can inspect system.event.clickCount to see what the value is. It will always be 2 (for a double-click) or greater.

It is also possible to inspect this system.event.clickCount in an onMouseUp handler as well in lieu of using onMultiClick. However, the advantage to using onMultiClick is that it does not interfere with window dragging the way that onMouseUp does, i.e. a mouse up handler on a text item will prevent a window from being dragged if you click that item. If your text item only needs to respond to a multi-clicks, you can use onMultiClick and the Widget will still be able to be dragged as usual.

```
<onMultiClick>
   if ( system.event.clickCount == 2 )
      alert( "Double Click!" );
</onMultiClick>
```

Availability

Available in version 2.0 or later.

opacity

how translucently the text displays

Description

The `opacity` attribute allows you to specify a value from 0 to 255 which controls the alpha value with which the text is rendered. An opacity of 0 is completely transparent (invisible) and has such side effects as preventing the object from reacting to mouse events. A value of 255 will render the text 100% opaque.

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <opacity>128</opacity>
</text>

myLabel.opacity = 33;
```

scrolling

direction and type of animated scrolling

Description

The `scrolling` attribute can take values of `off` (the default), `left`, `right`, `autoLeft`, or `autoRight`. If set, the text in the object scrolls continuously in the direction specified reappearing on the opposite edge as it disappears.

The "auto" variants only scroll if the text is too big for the area specified for its display (this is the most common use of scrolling, to make long text visible in a small space).

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <scrolling>autoLeft</scrolling>
</text>

myLabel.scrolling = "off";
```

shadow

sets shadow parameters for a text object

Description

You can specify a shadow to be displayed underneath a text object using the shadow attribute. To clear it, just set the shadow property to null. The shadow XML is the same as that described in the <shadow> section.

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <shadow hOffset="1" vOffset="1" color="#000000"/>
</text>

var s = new Shadow();
s.hOffset = 1;
s.vOffset = 1;
s.color = "#000000";
myLabel.shadow = s;
myLabel2.shadow = s;
```

Availability

Available in version 3.0 or later.

size

font size for the text block

Description

The point size for the text object.

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <size>22</size>
</text>

myLabel.size = 33;
```

style

the style of the text to display

Description

The style in which to render the text. Style can be any combination of:

```
italic, bold, narrow, expanded, condensed, smallcap, poster,
compressed, fixed
```

For example:

```
textObject.style = "bold;italic";
```

requests a bold, italic variation of the font named in the `font` attribute.

Note that the font must have the requested variation or else the style is ignored. Most fonts support only two or three variations.

Windows Note: only Bold and Italic are valid styles.

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <style>bold</style>
</text>

myLabel.style = 'italic';
```

truncation

Specifies whether to truncate text with an ellipsis or not.

Description

Normally, a text object will draw without any truncation. If there is no room for the entire text object to draw, it merely gets clipped. This tag allows you to instead specify that if the width of the text object is too small for the text, truncate it intelligently using an ellipse.

This tag only takes effect if there is a width specified for the text item, the text item is longer than the width, and no scrolling attribute is specified. Valid values are "none", "center", and "end".

Example

```
<text data="Example Text">
  <name>myLabel</name>
  <width>50</width>
  <truncation>end</truncation>
</text>
myLabel.truncation = "none";
```

Availability

Available in version 2.1 or later. "center" truncation is available in version 3.0 or later only.

tooltip

the tooltip for a text object

Description

The `tooltip` attribute defines the text displayed in a popup tooltip window when the mouse cursor rests over a `text` object.

JavaScript

`object``.tooltip`

Example

```
<text data="Example Text">
  <tooltip>Example tooltip</tooltip>
</text>
```

visible

Controls the visibility of a text object

Description

You can set the visible property of a text object to show or hide it by setting it to true or false, respectively. This allows you to hide objects without affecting their opacity, or having to save off the current opacity to restore it later. The default visibility for any object if not specified is true.

JavaScript

`myObjectName``.visible`

Example

```
<text data="Example Text">
  <visible>false</visible>
</text>

myText.visible = true;
```

Availability

Available in version 3.0 or later.

vOffset

the vertical offset of a text object

Description

The `vOffset` attribute of the `text` block defines the vertical (top to bottom) offset for the text based on 0,0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther down the text will be drawn.

JavaScript

object`.vOffset`

Example

```
<text data="Example Text">
  <vOffset>20</vOffset>
</text>
```

width

how wide the text object is made

Description

The `width` attribute controls the horizontal dimension of the text object. If none is specified, the object occupies just enough space to fit the text (rendered in the specified font, size, etc). It is sometimes useful to specify the width of a text object when using the `scrolling` attribute.

JavaScript

myObjectName`.width`

Example

```
<text data="Example Text">
  <width>30</width>
</text>

myLabel.width = 30;
```

window

The window to which this text belongs.

Description

You can specify the window a text object belongs to by specifying its name in the XML or its variable in JavaScript. If you do not specify a window, the object is automatically attached to the first window found in the XML description of a Widget.

JavaScript

myObjectName.window

Example

```
<window name="fred" width="100" height="100"/>
<text>
  <window>fred</window>
</text>

// Or in code
var myWind = new Window();
myText.window = myWind;

// You can also specify it in the constructor
var myText = new Image( myWind );
```

Availability

```
Available in version 2.0 or later.
```

zOrder

the stacking order of a text object

Description

The `zOrder` attribute of the `text` block defines the stacking order of the text. Objects with a higher `zOrder` are drawn on top of those with lesser `zOrder`s. Normally the `zOrder` is determined by the order in which objects are defined in the XML file with earlier objects being drawn under later ones but it can also be manipulated using

JavaScript at runtime.

JavaScript

myObjectName.zOrder

Example

```
<text data="Example Text">
  <zOrder>10</zOrder>
</text>

myLabel.zOrder = customZOrder++;
```

<textarea>

block defining a textarea object and associated default properties

Attributes

alignment
bgColor
bgOpacity
color
columns
contextMenuItems
bgColor
bgOpacity
data
editable
font
height
hAlign
hOffset
lines
name
onContextMenu
onDragDrop
onDragEnter
onDragExit
onGainFocus
onKeyUp
onKeyDown
onKeyPress
onLoseFocus
onMouseDown
onMouseEnter
onMouseExit
onMouseUp
onMultiClick
opacity
secure
scrollbar
size
spellcheck
style
tooltip
visible
vAlign
vOffset
width
window
zOrder

Description

The `textarea` block in the XML file defines the initial placement and mouse event scripts for an editable text object in a Widget.

textarea objects can also be created and destroyed dynamically via the JavaScript engine.

When you create more than one dynamic object with the same name, only the last object created will receive property changing events via JavaScript. As a result you should make sure that you're calling each dynamic object a unique name so they can be referenced properly (using a JavaScript Array is often a good way to achieve this).

You can remove a dynamic object once you create it using the JavaScript `delete` instruction.

JavaScript

```
newObjectName = new TextArea()
delete newObjectName
```

alignment

how the object is positioned relative to the given origin

Description

The alignment property of the textarea block defines how the object is positioned relative to its `hOffset` and `vOffset`.

Valid values are: `left, right` or `center`.

Note that this does not define the alignment of text within the textarea, rather how the object is positioned within the Widget. `left` is the most usual value for this property.

Example

```
<textarea data="Example Text">
  <alignment>left</alignment>
</textarea>
```

bgColor

the background color of a textarea object

Description

Set the color of the background of a textarea object. Colors are specified as browser style hex RGB triplets. For example:

```
#FF0000
```

is red.

Note that this property is closely linked with the `bgOpacity` property – both should be set to get a visible result.

Example

```
<textarea data="Example Text">
  <bgColor>#FFFFFF</bgColor>
  <bgOpacity>150</bgOpacity>
</textarea>
```

bgOpacity

the opacity of the background of a textarea object

Description

Set the opacity of the background of a textarea object. Opacities are specified as a number between 0 and 255.

Note that this property is closely linked with the `bgColor` property – both should be set to get a visible result.

Example

```
<textarea data="Example Text">
  <bgColor>#FFFFFF</bgColor>
  <bgOpacity>150</bgOpacity>
</textarea>
```

color

color that the text draws in

Description

Set the color of the text. Colors are specified as browser style hex RGB triplets. For example:

```
#00FF00
```

is green.

If you set the `color` and `bgColor` to the same value you won't be able to see the text.

Example

```
<textarea data="Example Text">
  <color>#F42DA6</color>
</textarea>
```

columns

number of columns wide to make the object

Description

Instead of giving a `width` and `height` for textarea objects, their size can be specified in terms of a number of `columns` and `lines` of text in the current font.

Note that using a proportional font makes the number of columns approximate.

Example

```
<textarea>
  <columns>40</columns>
  <lines>10</lines>
</textarea>
```

contextMenuItems

Specifies an array of context menu items.

Description

You can add items to the standard context menu that appears when the user right-clicks the mouse button on your Widget by adding contextMenuItems to your text area. This tag is actually valid for image, text, and window objects as well. You can also dynamically build your context items by specifying some JavaScript to execute on your onContextMenu tag (see onContextMenu for more information).

You specify your items by including an array of menuItem objects. See the section on menuItem for more information about them.

JavaScript

myObjectName.contextMenuItems

Example

```
<textarea>
  ...
  <contextMenuItems>
    <menuItem title="Test" onSelect="beep();"/>
    <menuItem title="Another Test">
      <onSelect>alert( 'hello' );</onSelect>
    </menuItem>
  </contextMenuItems>
</textarea>
```

See the onContextMenu section for an example of building a context menu in JavaScript.

Availability

Available in version 2.0 or later.

data

the text that the textarea object contains

Description

The text to be edited. This is optional. If omitted, the user will be presented with an empty text entry field.

Example

```
<textarea>
  <data>Example Text</data>
</textarea>
```

editable

sets whether the text can be edited

Description

Set `editable` to false to make the textarea display only.

Example

```
<textarea data="Example Text">
  <editable>false</editable>
</textarea>

ta1.editable = true;
```

font

the font that the textarea uses

Description

The name of the font to be used to render the text. If the specified font cannot be found then the default System Font is used. Separate multiple font names with commas to specify *fallbacks* (fonts that used in the event a preceding font isn't found on the user's system).

Example

```
<textarea data="Example Text">
  <font>Palatino</font>
</textarea>

ta1.font = "Palatino, Times";
```

hAlign

Control the horizontal alignment of a textarea object.

Description

This is a synonym for the alignment tag. See the description of that tag for information.

Availability

Available in version 2.0 or later.

height

how tall the textarea object is made

Description

The `height` attribute controls the vertical dimension of the textarea object.

JavaScript

myObjectName`.height`

Example

```
<textarea data="Example Text">
  <height>30</height>
</textarea>

ta1.height = 30;
```

hOffset

the horizontal offset of a textarea object

Description

The `hOffset` attribute of the text block defines the horizontal (left to right) offset for the text based on 0,0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther to the right the text will be drawn.

JavaScript

myObjectName`.hOffset`

Example

```
<textarea data="Example Text">
  <hOffset>30</hOffset>
</textarea>
```

lines

number of lines high to make the object

Description

Instead of giving a `width` and `height` for textarea objects their size can be specified in terms of a number of `columns` and `lines` of text in the current font.

Specifying a value of `1` for `lines` changes the behavior of the textarea object slightly. Instead of wrapping, the text scrolls sideways when then the edge of the object is reached while typing.

Example

```
<textarea>
  <columns>40</columns>
  <lines>10</lines>
</textarea>
```

name

the reference name of a textarea object

Description

The `name` attribute of the `text` block defines the name of the textarea object when referenced by JavaScript. Since the name is used for reference in code, it must not contain any spaces or non ASCII characters.

The name of an object cannot be changed once it's assigned.

When creating a dynamic object via JavaScript, you use the name of the variable to represent the new name of the object.

JavaScript

```
newObjectName = new TextArea()
```

Example

```
<textarea data="Example Text">
  <name>myText</name>
</textarea>

myText.hOffset = 22;
```

onContextMenu

A context menu is about to appear. Time to add your items.

Description

The simplest way to specify context menu items that get added to the standard context menu for a Widget is to use the contextMenuItems tag in the XML. However, for those Widgets that need to build their items dynamically, the onContextMenu handler is your hook to do so. When the menu is about to be presented, this is called for all elements under the mouse from front to back in the view order until some view responds. When handling this, you should simply build your context menu items and set your contextMenuItems property to the array of items.

JavaScript

```
myTextArea.onContextMenu
```

Example

```
<onContextMenu>
var items = new Array();
items[0] = new MenuItem();
items[0].title = "This is the title";
items[0].enabled = false;
items[0].checked = true;
items[0].onSelect = "alert( 'you chose it!' );";

items[1] = new MenuItem();
items[1].title = "This is the second title";
items[1].onSelect = "beep();";

myTextArea.contextMenuItems = items;
</onContextMenu>
```

onDragDrop

the script called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string is dragged from another application (e.g. the Finder) and dropped on the object.

In the "onDragDrop" action objects can access `system.event.data` to see what was dropped. This is an array of strings the first element of which tells you what kind of thing was dropped: `filenames`, `urls` or `string`. The remaining elements of the array are the items that were dropped.

JavaScript

`myObjectName``.onDragDrop`

Example

```
<textarea data="Drop Stuff Here">
  <name>dropper</name>
  <onDragDrop>
        if (system.event.data[0] == "filenames")
        {
              dropper.data = runCommand("cat " +
                             system.event.data[1]);
        }
  </onDragDrop>
</textarea>

<textarea data="Drop Stuff Here">
  <onDragDrop>dragCode.js</onDragDrop>
</textarea>

dropper.onDragDrop = "handleDragDrop();";
```

onDragEnter

the script that gets called when an item is dragged into the object

Description

The `onDragEnter` attribute of the `textarea` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object. This happens before the item is dropped (indeed it may not be dropped as the user can change their mind).

This is useful for triggering a visual change of the object to indicate to the user that the dragged object will be accepted or rejected if it is dropped. Information about the item being dragged is contained in `system.event.data` (see `onDragDrop` for details).

JavaScript

myObjectName.onDragEnter

Example

```
<textarea data="Drop Stuff Here">
  <name>dropper</name>
  <onDragEnter>
        highlightDropTarget(dropper);
  </onDragEnter>
</textarea>

well.onDragEnter = "highlightDropTarget(well);";
```

onDragExit

the script that gets called when an item is dragged out of the object

Description

The `onDragExit` attribute of the `textarea` block is a wrapper for JavaScript code that will execute when the user has dragged an item from another application into the object and then out again.

This is useful for undoing things that were done in `onDragEnter`.

JavaScript

myObjectName.onDragExit

Example

```
<textarea data="Drop Stuff Here">
  <name>dropper</name>
  <onDragExit>
        unhighlightDropTarget(dropper);
  </onDragExit>
</textarea>

dropper.onDragExit = "unhighlightDropTarget(dropper);";
```

onGainFocus

called when a textarea gets the keyboard focus

Description

This is called when a text area acquires the keyboard focus via a call to textarea.focus(),

or the user clicks in the text area. You might use this to draw a focus adornment to indicate focus. Only editable text areas get the keyboard focus, and as such this action will only be called for an editable text area.

JavaScript

NewObjectName.onGainFocus

Example

```
<textarea data="Type Stuff Here">
  <name>typomatic</name>
  <onGainFocus>
       print("I am the focus!");
  </onGainFocus>
</textarea>

typomatic.onGainFocus = "print( 'focus gained!' );";
```

onKeyDown

the code that is activated when a key is pressed

Description

The code to be run when a key is pressed and the mouse cursor is within the bounds of the textarea object is specified with the `onKeyDown` attribute. Note that it is generally best to attach key code to the `onKeyPress` action for textareas as that works the way users expect.

JavaScript

NewObjectName.onKeyDown

Example

```
<textarea data="Type Stuff Here">
  <name>typomatic</name>
  <onKeyDown>
       print(system.event.key);
  </onKeyDown>
</textarea>

typomatic.onKeyDown = "keypressed = true";
```

onKeyPress

the script called when a key is pressed and the textarea has focus

Description

The `onKeyPress` attribute of a `textarea` block is a wrapper for JavaScript code that will execute when the user presses a key that will affect the object (unless steps are taken, see below).

This is useful for performing validation of text entry. Normally any key pressed is processed by the system and the appropriate change made to the textarea (adding a character, deleting a word, etc), you can override this behavior by calling the textarea method `rejectKeyPress()` which causes the key press to be ignored (it is always available in `system.event.key`).

JavaScript

myObjectName.onKeyPress

Example

```
<textarea>
  <name>ta1</name>
  <onKeyPress>
      // Convert input to uppercase
      var key = system.event.key;
      if (key.charCodeAt(0) &gt;= "A".charCodeAt(0) &&
          key.charCodeAt(0) &lt;= "z".charCodeAt(0))
      {
        // Tell the text area to ignore this keyPress
        ta1.rejectKeyPress();


        // Append an upper case copy of the key pressed
        ta1.replaceSelection(key.toUpperCase());
      }
  </onKeyPress>
</textarea>

<textarea data="Example Text" name="ta1">
  <onKeyPress>textareaCode.js</onKeyPress>
</textarea>

ta1.onKeyPress= "doProcessKeys(ta1);";
```

onKeyUp

the code that is activated when a key is released

Description

The code to be run when a key is pressed and the mouse cursor is within the bounds of the textarea object is specified with the `onKeyUp` attribute (note that `onKeyPress` is generally a better way to handle keystrokes in a textarea).

JavaScript

`NewObjectName``.onKeyUp`

Example

```
<textarea data="Type Stuff Here">
  <name>typomatic</name>
  <onKeyUp>
       print(system.event.key);
  </onKeyUp>
</textarea>

typomatic.onKeyUp = "keypressed = true";
```

onLoseFocus

called when a textarea loses focus

Description

This is called when a text area that was previous focused via focus() has lost its focus. You might use this to clear any focus adornment you might draw around the text area to indicate focus. Only editable text areas get the keyboard focus, and as such this action will only be called for an editable text area.

JavaScript

`NewObjectName``.onLoseFocus`

Example

```
<textarea data="Type Stuff Here">
  <name>typomatic</name>
  <onLoseFocus>
       print("I lost focus!");
  </onLoseFocus>
</textarea>

typomatic.onLoseFocus = "print( 'focus lost!' );";
```

onMouseDown

the script called when the mouse button is down inside the object

Description

The `onMouseDown` attribute of a `textarea` block is a wrapper for JavaScript code that will execute when the user presses the mouse button down within the object.

Since the mouse is used to move the insertion point, select text, etc care should be taken when specifying mouse actions on textarea objects.

JavaScript

myObjectName.onMouseDown

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <onMouseDown>
        ta1.color = "#FF0000";
  </onMouseDown>
</textarea>

<textarea data="Example Text" name="ta1">
  <onKeyPress>textareaCode.js</onKeyPress>
</textarea>

ta1.onMouseDown = "doHighlight(ta1);";
```

onMouseEnter

the script that gets called when the mouse rolls into the object

Description

The `onMouseEnter` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has moved the cursor within the object.

Since the mouse is used to move the insertion point, select text, etc care should be taken when specifying mouse actions on textarea objects.

JavaScript

myObjectName.onMouseEnter

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <onMouseEnter>
       ta1.color = "#EEEEEE";
  </onMouseEnter>
</textarea>

ta1.onMouseEnter = "handleMouseEnter(ta1);";
```

onMouseExit

the script that gets called when the mouse rolls out of an object

Description

The `onMouseExit` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has moved the cursor from within the object to outside the object.

Since the mouse is used to move the insertion point, select text, etc care should be taken when specifying mouse actions on textarea objects.

JavaScript

myObjectName.onMouseExit

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <onMouseExit>
       ta1.color = "#FFFFFF";
  </onMouseExit>
</textarea>

ta1.onMouseExit = "handleMouseExit(ta1);";
```

onMouseUp

the script that gets called on mouse up in an object

Description

The `onMouseUp` attribute of the `text` block is a wrapper for JavaScript code that will execute when the user has released the mouse after having it down within the object.

Since the mouse is used to move the insertion point, select text, etc care should be taken when specifying mouse actions on textarea objects.

Please note that `onMouseUp` will trigger even if the mouse is not inside the object with the mouse is released.

JavaScript

myObjectName.onMouseUp

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <onMouseUp>
        ta1.color = "#FFFFFF";
  </onMouseUp>
</textarea>

ta1.onMouseUp = 'handleOnMouseUp(ta1);';
```

onMultiClick

A multiple click just occurred.

Description

You can easily trap double-clicks (or triple-clicks, etc.) using the onMultiClick handler. This handler can be set on image, text, text area, and window objects. Whenever your onMultiClick handler is called, you can inspect system.event.clickCount to see what the value is. It will always be 2 (for a double-click) or greater.

It is also possible to inspect this system.event.clickCount in an onMouseUp handler as well in lieu of using onMultiClick. However, the advantage to using onMultiClick is that it does not interfere with window dragging the way that onMouseUp does, i.e. a mouse up handler on an textarea will prevent a window from being dragged if you click that textarea. If your textarea only needs to respond to a multi-clicks, you can use onMultiClick and the Widget will still be able to be dragged as usual.

```
<onMultiClick>
   if ( system.event.clickCount == 2 )
      alert( "Double Click!" );
</onMultiClick>
```

Availability

Available in version 2.0 or later.

opacity

how translucently the text displays

Description

The `opacity` attribute allows you to specify a value from 0 to 255 which controls the alpha value with which the text is rendered. An opacity of 0 is completely transparent (invisible) and has such side effects as preventing the object from reacting to mouse events. A value of 255 will render the text 100% opaque.

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <opacity>128</opacity>
</textarea>

ta1.opacity = 33;
```

secure

sets the textarea to display bullets instead of text

Description

This property is used to mimick a password field, where the user cannot see the actual text being typed, but rather just a series of bullet characters (small circles). This should generally be used with 1 row of text.

Example

```
<textarea data = "hello!">
  <secure>true</secure>
</textarea>
```

Availability

Available in version 2.1 or later.

scrollbar

controls the display of a scrollbar on the textarea

Description

By default a textarea will display a vertical scrollbar. Use this attribute to turn it off.

Example

```
<textarea data="Example Text">
  <scrollbar>false</scrollbar>
</textarea>

ta1.scrollbar = true;
```

size

font size for the textarea block

Description

The point size for the textarea object.

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <size>22</size>
</textarea>

ta1.size = 33;
```

spellcheck

controls continuous spellchecking in the textarea

Description

By default a textarea will highlight spelling errors as the user types, this can be turned off using this attribute.

Example

```
<textarea data="Example Text">
  <spellcheck>false</spellcheck>
</textarea>

ta1.spellcheck = true;
```

style

font style for the textarea block

Description

The style in which to render the text. Style can be any combination of:

```
italic, bold, narrow, expanded, condensed, smallcap, poster,
compressed, fixed
```

For example:

```
textAreaObject.style = "bold;italic";
```

requests a bold, italic variation of the font named in the `font` attribute.

Note that the font must have the requested variation or else the style is ignored. Most fonts support only two or three variations.

Example

```
<textarea data="Example Text">
  <name>ta1</name>
  <style>bold</style>
</textarea>

ta1.style = 'italic';
```

thumbColor

the thumb color of the scrollbar

Description

The `thumbColor` property is used to control the tint of the scroll bar thumb if your text area has a scroll bar specified. The default thumb in the standard scroll bar is a medium gray. The color you specified is applied via colorization.

To clear the current color completely, you can set it to null in Javascript.

Example

```
<scrollbar>
  <thumbColor>#333366</thumbColor>
</scrollbar>

myScrollbar.thumbColor = "#333366";
myScrollbar.thumbColor = null;
```

Availability

Available in version 3.0 or later. Currently only available on Windows.

tooltip

the tooltip for a textarea object

Description

The `tooltip` attribute defines the text displayed in a popup tooltip window when the mouse cursor rests over a `textarea`.

JavaScript

`object``.tooltip`

Example

```
<textarea data="Example Text">
  <tooltip>Example tooltip</tooltip>
</textarea>
```

visible

Controls the visibility of a textarea object

Description

You can set the visible property of a textarea object to show or hide it by setting it to true or false, respectively. This allows you to hide objects without affecting their opacity, or having to save off the current opacity to restore it later. The default visibility for any object if not specified is true.

JavaScript

`myObjectName``.visible`

Example

```
<textarea data="Example Text">
  <visible>false</visible>
</textarea>

myTextArea.visible = true;
```

Availability

Available in version 3.0 or later.

vAlign

Controls the vertical alignment of a text area.

Description

The vAlign property of a text area defines how it is positioned vertically relative to its vOffset. For example, a text area with a `bottom` alignment will be drawn so that its bottom edge appears at the `vOffset` (see below). If this tag is not specified, the default value is `top`.

Valid values are: `top, bottom` or `center`.

JavaScript

myObjectName.vAlign

Example

```
<textarea src="button.png">
  <vAlign>bottom</vAlign>
</textarea>

myTextArea.vAlign = "bottom";
```

Availability

Available in version 2.0 or later.

vOffset

the vertical offset of a textarea object

Description

The `vOffset` attribute of the `textarea` block defines the vertical (top to bottom) offset for the text based on 0,0 being the upper left hand corner of the object's parent view (superview). The greater the value assigned, the farther down the text will be drawn.

JavaScript

object.vOffset

Example

```
<textarea data="Example Text">
  <vOffset>20</vOffset>
</textarea>
```

width

how wide the textarea object is made

Description

The `width` attribute controls the horizontal dimension of the textarea object.

JavaScript

myObjectName.width

Example

```
<textarea data="Example Text">
  <width>30</width>
</textarea>

ta1.width = 30;
```

See Also

columns

window

The window to which this textarea belongs.

Description

You can specify the window a textarea belongs to by specifying its name in the XML or its variable in JavaScript. If you do not specify a window, the textarea is automatically attached to the first window found in the XML description of a Widget.

JavaScript

myObjectName.window

Example

```
<window name="fred" width="100" height="100"/>
<textarea>
  <window>fred</window>
</textarea>

// Or in code
var myWind = new Window();
myTextArea.window = myWind;

// You can also specify it in the constructor
var myTextArea = new Image( myWind );
```

Availability

```
Available in version 2.0 or later.
```

zOrder

the stacking order of a textarea object

Description

The `zOrder` attribute of the `text` block defines the stacking order of the text. Objects with a higher `zOrder` are drawn on top of those with lesser `zOrder`s. Normally the `zOrder` is determined by the order in which objects are defined in the XML file with earlier objects being drawn under later ones but it can also be manipulated using JavaScript at runtime.

JavaScript

myObjectName.zOrder

Example

```
<textarea data="Example Text">
  <zOrder>10</zOrder>
</textarea>

ta1.zOrder = customZOrder++;
```

<timer>

block to define a timer

Attributes

`interval`
`name`
`ticking`
`onTimerFired`

Methods

reset()

Description

Timer objects allow you to perform a task at a periodic interval (e.g every 5 seconds) or can be used to simply fire once at a later time. They are a replacement for the older onTimer triggers in actions. They allow you to create multiple timers all running on different frequencies. They can also be started and stopped without having to delete them.

The intervals are not guaranteed to be exact. Timers run 'cooperatively' meaning that they fire when the Widget is not busy doing other things. You cannot do high-precision time-based actions in a timer.

Availability

Available in version 2.0 or later.

interval

This is simply the interval, in seconds the timer should fire. It can be expressed in floating point, so if you want a timer to fire every half second, just specify 0.5 for this property. Each time the timer fires, it executes the Javascript in the onTimerFired property.

JavaScript

object.interval

Example

```
<timer name="myTimer">
  <interval>1.0</interval>
</window>

myTimer.interval = 1.0;
```

Availability

Available in version 2.0 or later.

name

the reference name of a timer

Description

The name attribute of the preference block defines the name of the global variable to be created in Javascript. Since the name is used for reference in code, it should not contain any spaces or non ASCII characters. Once the name is used to build the object, it should be considered to be invalid.

Example

```
<timer name="my_timer"/>

// then later in Javascript, you can reference by that name:
my_timer.interval = 10;
```

Availability

Available in version 2.0 or later.

ticking

This allows you to turn a timer on and off by setting it to true and false, respectively. If you want to disable a timer for a while, just set ticking to false. Later, set it to true, and it will start firing again. Once restarted, the next time it fires will be 'now' plus the interval. So if you have a one second timer, it will fire one second after you set ticking to true.

JavaScript

object.ticking

Example

```
<timer name="myTimer">
  <ticking>false</ticking>
</timer>

myTimer.ticking = true;
```

Availability

Available in version 2.0 or later.

onTimerFired

This property contains the Javascript you want to execute when the timer is fired.

JavaScript

object.onTimerFired

Example

```
<timer name="myTimer">
  <onTimerFired>alert( 'hello!' );</onTimerFired>
</timer>

myTimer.onTimerFired = "alert( 'fired!' );";
```

Availability

Available in version 2.0 or later.

<widget>

block to define the scope of the Widget

Attributes

```
author
company
copyright
debug
defaultTracking
image
minimumVersion
option
requiredPlatform
version
```

Description

The outermost scope in the XML file is defined by the `widget` block. This groups together all the objects that make up the Widget.

In version 3.0 and later, the widget object is accessible through the global object named, surprisingly, 'widget'. Currently all attributes are read-only.

author

specifies the author's name

You can provide this optional piece of information for your Widget. In version 3.0 this is used to display in our 'first run' security dialog so that people can see who wrote the Widget (along with the 'company' attribute). In the future, this information will be used in other areas of the interface, so it's a good idea to supply author and/or company and copyright attributes.

Availability

Available in version 3.0 or later.

company

specifies the company name

You can provide this optional piece of information for your Widget. In version 3.0 this is used to display in our 'first run' security dialog so that people can see what company is publishing the Widget. In the future, this information will be used in other areas of the interface, so it's a good idea to supply author and/or company and copyright attributes.

Availability

Available in version 3.0 or later.

copyright

specifies the Widget's copyright string

You can provide this optional piece of information for your Widget. In the future, this information will be used in other areas of the interface, so it's a good idea to supply author and/or company and copyright attributes.

Availability

Available in version 3.0 or later.

debug

control if the debug console is shown for this Widget

Description

If you need to enable or suppress debug output, add this block inside the `widget` block.

`<debug>errors</debug>`
Turns Widget debug information on if an error occurs while the Widget is running. Errors can be JavaScript or Widget Engine runtime messages. This is the default.

`<debug>on</debug>`
Turns Widget debug information on when the Widget is opened (useful when developing a Widget).

`<debug>off</debug>`
Keeps Widget debug information off when the Widget is opened even if errors occur. This mode should only be used when a Widget is thoroughly debugged. Note that if a Widget generates 10 errors the debug console is displayed no matter what the setting of this option is.

`<debug>verbose</debug>`
Turns Widget debug information on when the Widget is opened and causes information about object actions and other automatically triggered events to be displayed (useful when developing a Widget).

Notes

In version 2.0.1 or later, you can hold control-option while selecting the Gear menu and a "Debug Mode" option will be presented. Checking that will enable debugging for any Widget launched after that option is turned on. This means you don't necessarily need to set the debug attribute in a Widget any longer.

defaultTracking

set the default tracking style for images

Description

The defaultTracking attribute specifies whether the default cursor tracking style for images. It can be either `opacity` (the default) or `rectangle` which makes images clickable anywhere inside their bounding rectangle instead of only on their non-transparent parts.

Example

```
<widget defaultTracking="rectangle">
  ...
</widget>
```

image

specifies the Widget's icon/image

You can provide this optional piece of information for your Widget. This attribute contains the relative path to a 150x150 image to display to represent your Widget. Currently, this image is only displayed in our standard Security dialog when an unknown Widget is run for the first time, or has been modified since last run. In the future it will be used in other areas of the interface, so it's a good idea to supply an image for your Widget along with author/company/copyright information.

Availability

Available in version 3.0 or later.

minimumVersion

the minimum version of the Widget Engine that is required to run this Widget

Description

Specifying `minimumVersion` for a Widget causes the Widget Engine to check it against the version of the engine that is currently running. If the current version is less than the version specified, an error message is displayed to the user and the Widget won't run.

Example

```
<widget minimumVersion="1.5">
  ...
</widget>
```

Starting in version 3.0 and later, this attribute also tells the Widget Engine that it can enable new behaviors. We use this as a clue that the Widget has been modified to

work with the minimum version specified and as such, has been adjusted to behave correctly with that version. With this mechanism, we allow older Widgets to continue to run unmodified, but newer/modified Widgets who advertise their support for 3.0 to possibly require modifications to run in the new environment. This basically allows us to change the way things work without breaking existing Widgets.

option

various Widget options

Description

These options affect the behavior of the Widget as a whole.

`<option>allowCustomObjectAttributes</option>`
When this Widget attribute is specified, custom object attributes are allowed, and will not trigger debug errors. This is an advanced feature for people who are comfortable with the JavaScript object model and has the drawback that simple typos in object attribute names (e.g. hOffset) can be very difficult to locate.

`<option>dontRememberWindowPosition</option>`
This option tells the Widget Engine to not save the window position of this Widget when it is closed. Normally, the positions of all Widgets are remembered between invocations of the Widget Engine so the user can lay out their desktop just as they like but some kinds of Widgets work better by positioning themselves programmatically each time they run.

`<option>allowArbitraryXML</option>`
This turns off checking for valid XML tags and attribute names. If you want to embed XML not recognized by the Widget Engine in your Widget you should specify this option to avoid getting errors (note that the additional XML must be well formed). It is rare that this option is required.

requiredPlatform

specifies if this Widget requires a particular platform to run

While it is preferred that Widgets run on all platforms that the Widget Engine supports, at times Widgets might make use of highly specific platform features (such as COM on Windows). To indicate that your Widget requires a particular platform, you can specify this attribute with the values "macintosh" or "windows".

Availability

Available in version 1.8 or later.

version

the version number of your Widget

You can specify the current version of your Widget in this attribute. If you make use of the <about-version> object in your <about-box> object, this is the information that gets displayed.

Availability

Available in version 1.5 or later.

<window>

block that defines the main window of the Widget

Attributes

```
alignment
contextMenuItems
height
hOffset
level
name
onContextMenu
onFirstDisplay
onGainFocus
onLoseFocus
onMultiClick
opacity
shadow
title
visible
vOffset
width
```

Description

The `window` block describes the size and position of a Widget window. This window is always transparent and only the images and text objects you put in it are visible to the user.

Multiple Windows

Starting in version 2.0, you can have multiple windows in your Widgets. You attach the objects (images, text, etc.) to your window by specifying the variable of a window as the window attribute of the object. This way, each object knows where they live.

If you give your window a name (either in the XML or in a JavaScript constructor), the Widget Engine will track the windows position and save it to preferences automatically for you. Unnamed windows will not have their preferences saved (they are usually a small transient window such as a bezel so there's no point).

Each window maintains its own window level. However, if the user sets the Widget's level in the Preferences dialog to a value, all windows will be set to that value. If you don't wish a window to take on such a value, reset it to what you like in an onPreferencesChanged handler.

alignment

how the window is aligned relative to its screen position

Description

This specifies the alignment of the Widget's window relative to its `hOffset` and `vOffset`. Possible values are `left`, `right` or `center`. A window that is `left` aligned will have its top left corner positioned at the specified offsets; a right aligned window will have its top right corner positioned at the offsets; a `center` aligned window will be positioned so that its top center falls on the offsets.

The default window alignment is `center`.

JavaScript

`object``.alignment`

Example

```
<window title="My Widget">
  <alignment>left</alignment>
</window>

myWindow.alignment = "left";
```

contextMenuItems

Specifies an array of context menu items.

Description

You can add items to the standard context menu that appears when the user right-clicks the mouse button on your Widget by adding contextMenuItems to your window. This tag is actually valid for text, textArea, and image objects as well. You can also dynamically build your context items by specifying some JavaScript to execute on your onContextMenu tag (see onContextMenu for more information).

You specify your items by including an array of menuItem objects. See the section on menuItem for more information about them.

JavaScript

`myObjectName``.contextMenuItems`

Example

```
<window>
...
<contextMenuItems>
   <menuItem title="Test" onSelect="beep();"/>
   <menuItem title="Another Test">
      <onSelect>alert( 'hello' );</onSelect>
   </menuItem>
</contextMenuItems>
...
</window>
```

See the onContextMenu section for an example of building a context menu in JavaScript.

Availability

Available in version 2.0 or later.

height

how tall the window can be

Description

This specifies the height of the Widget's window in pixels.

JavaScript

object.height

Example

```
<window title="My Widget">
  <height>200</height>
</window>

myWindow.height = 250;
```

hOffset

what the initial horizontal placement of the window is

Description

The `hOffset` attribute of the `window` block defines the horizontal (left to right) offset for the window based on 0,0 being the upper left hand corner of the screen. The greater the value assigned, the farther across the screen the window will appear.

If you do not specify a value for `hOffset` in the XML then it will appear as `-1` (minus one) in the `onLoad` action of the Widget. This allows you to programmatically determine the initial window position (for example, place it in the lower right corner no matter the screen resolution).

JavaScript

object.hOffset

Example

```
<window title="My Widget">
  <hOffset>200</hOffset>
</window>

myWindow.hOffset = 250;
```

level

where the window sits in relation to others

Description

This attribute can have one of the following values: `konspose`, `desktop`, `below`, `normal`, `topmost` or `floating`. It specifies how the window behaves with respect to other windows on the desktop, whether it appears below others or floats above everything (`konspose` means it only appears in Konsposé mode). The default is `normal`.

JavaScript

object.level

Example

```
<window title="My Widget">
  <level>below</level>
</window>

myWindow.level = 'normal';
```

name

name of the window

Description

Name used to identify the window in JavaScript.

JavaScript

object.name

Example

```
<window title="My Widget">
  <name>myWindow</name>
</window>

print(myWindow.name);
```

onContextMenu

A context menu is about to appear. Time to add your items.

Description

The simplest way to specify context menu items that get added to the standard context menu for a Widget is to use the contextMenuItems tag in the XML. However, for those Widgets that need to build their items dynamically, the onContextMenu handler is your hook to do so. When the menu is about to be presented, this is called for all elements under the mouse from front to back in the view order until some view responds. When handling this, you should simply build your context menu items and set your contextMenuItems property to the array of items. The window always gets a chance to add items, regardless of if a view in front of it handled this message itself.

JavaScript

myWindow.onContextMenu

Example

```
<onContextMenu>
var items = new Array();
items[0] = new MenuItem();
items[0].title = "This is the title";
items[0].enabled = false;
items[0].checked = true;
items[0].onSelect = "alert( 'you chose it!' );";

items[1] = new MenuItem();
items[1].title = "This is the second title";
items[1].onSelect = "beep();";

myWindow.contextMenuItems = items;
</onContextMenu>
```

Availability

Available in version 2.0 or later.

onFirstDisplay

Specifies an action to carry out the very first time a window is displayed.

Description

The very first time a window is ever shown in a Widget (i.e. we see that it has no saved preferences for position, etc.), any onFirstDisplay handler on the window is called. This allows a Widget to decide where it should appear the very first time the Widget is launched.

Example

```
<onFirstDisplay>
    setInitialPosition();
</onFirstDisplay>
```

Remember, this is only sent the first time the window appears. Once the prefs are saved for that window, you will never receive this message again.

Availability

Available in version 2.0 or later.

onGainFocus

Handle window-specific activation.

Description

This handler allows you to be informed when your window becomes active. You can use this time to put up adornments, etc. to show that you are active if you wish. In versions prior to 2.0 you could only do this at the Widget level. With the advent of multiple windows in 2.0, you should generally move to use window-specific handlers instead.

Example

```
<onGainFocus>
   showPlayButton();
</onGainFocus>
```

Availability

Available in version 2.0 or later.

onLoseFocus

Handle window-specific deactivation.

Description

This handler allows you to be informed when your window becomes inactive. You can use this time to remove any adornments, etc. you might have been showing while active. In versions prior to 2.0 you could only do this at the Widget level. With the advent of multiple windows in 2.0, you should generally move to use window-specific handlers instead.

Example

```
<onLoseFocus>
   hidePlayButton();
</onLoseFocus>
```

Availability

Available in version 2.0 or later.

onMultiClick

A multiple click just occurred.

Description

You can easily trap double-clicks (or triple-clicks, etc.) using the onMultiClick handler. This handler can also be set on image, text, and text area objects. Whenever your onMultiClick handler is called, you can inspect system.event.clickCount to see what the value is. It will always be 2 (for a double-click) or greater.

It is also possible to inspect this system.event.clickCount in an onMouseUp handler as well in lieu of using onMultiClick.

```
<onMultiClick>
   if ( system.event.clickCount == 2 )
      alert( "Double Click!" );
</onMultiClick>
```

Availability

Available in version 2.0 or later.

opacity

how translucently the window displays

Description

A value from 0 to 255 which affects the opacity of the entire window and its contents.

JavaScript

object.opacity

Example

```
<window title="My Widget">
  <opacity>180</opacity>
</window>

myWindow.opacity = 255;
```

shadow

if the window casts an Aqua generated shadow

Description

Controls whether the Widget has an Aqua shadow.

This attribute is not applicable on Windows.

Note: as of Mac OS X 10.2, there's an additional gray border that gets placed around the window when you turn on this feature. Keep this in mind when designing your Widget.

JavaScript

object.shadow

Example

```
<window title="My Widget">
  <shadow>false</shadow>
</window>

myWindow.shadow = true;
```

Windows Note

Window shadows are not currently supported on the Windows platform. If you decide to render your artwork with your own shadows, be sure to set the shadow property of your window to 0. If not, when and if Windows does support shadows you'll end up with a double-shadow.

title

name of the window for display

Description

This is currently used as the name for the Widget in the context menu.

JavaScript

object.title

Example

```
<window>
  <title>My Widget</title>
</window>

myWindow.title = "My New Widget";
```

visible

if the window is visible to the user

Description

Useful for hiding an initial building of a dynamic Widget by setting it to `false` in the XML definition of the window and then setting it to `true` at end of the `onLoad` trigger once the Widget has been constructed.

Note that if the `visible` attribute is false your Widget won't appear on the screen and you won't be able to interact with it.

JavaScript

object.visible

Example

```
<window title="My Widget">
  <visible>false</visible>
</window>

myWindow.visible = true;
```

vOffset

what the initial vertical placement of the window is

Description

The `vOffset` attribute of the `window` block defines the vertical (top to bottom) offset for the window based on 0,0 being the upper left hand corner of the screen. The greater the value assigned, the farther down the window will appear.

If you do not specify a value for `vOffset` in the XML then it will appear as `-1` (minus one) in the onLoad action of the Widget. This allows you to programmatically determine the initial window position (for example, place it in the lower right corner no matter the screen resolution).

JavaScript

object.vOffset

Example

```
<window title="My Widget">
  <vOffset>200</vOffset>
</window>
```

width

how wide the window can be

Description

This specifies the width of the Widget's window in pixels.

JavaScript

object.width

Example

```
<window title="My Widget">
  <width>200</width>
</window>

myWindow.width = 250;
```

JavaScript Reference

This section describes the extensions to JavaScript that are provided by the Widget Engine. If JavaScript is new to you, consider obtaining a guide to the language to help with its syntax and structure. The Yahoo! Widget Engine implements a JavaScript engine (Mozilla Spidermonkey) that conforms to the JavaScript 1.5 standard (ECMA-262, revision 3).

The Widget Engine's extensions fall into several categories:

- global functions
- system functions
- system attributes
- system objects
- Widget Engine object methods

Global Functions

These can be used anywhere in your JavaScript.

alert()

display an alert dialog

Synopsis

`alert(`*`string, [button one, button two, button three]`*`)`

Attributes

`string`	The text contents of the alert that that will be displayed.
`button one`	The text presented on the first (or only) button shown on the alert. This argument is optional.
`button two`	The text presented on the second button of the alert. This argument is optional.
`button three`	The text presented on the third button shown on the alert. This argument is optional.

Returns

Once the alert dialog is presented to the user, the dialog returns 1, 2, or 3 based on which button was pressed.

Description

Used to give the user an immediate message in a standard alert dialog, or to ask them to pick from up to three options. The return value can be `1`, `2`, or `3` to indicate which of three optional buttons were pressed.

Example

```
alert("The time is now " + Date());

answer = alert("Do you wish to continue?", "Yes", "No");

if (answer == 2)
  closeWidget();
```

appleScript()

execute an AppleScript

Synopsis

```
appleScript(appleScriptCode[, timeout])
```

Parameters

appleScriptCode
A string that contains a complete AppleScript code snippet that you want to have executed. If the string consists only of a valid filename then the code is loaded from that file.

timeout
The optional number of seconds to wait for the AppleScript to complete. For compatibility reasons the default timeout is 2 seconds.

Description

Using this function, your Widget can control an element of the System or an application via an AppleScript call.

The AppleScript must be formatted as a non-breaking line, using new-line characters to connote a physical break. We suggest pre-formatting and validating your AppleScript in Apple's Script Editor application before using it in a Widget.

The **iTunes Remote Widget** makes extensive use of the `appleScript()` call.

Example

```
// Note the embedded new-lines that are required
// in AppleScripts.
appleScript('tell application "Internet Explorer"\nOpenURL ("' +
newURL + '")\nend tell\n');
```

beep()

play the alert sound

Synopsis

```
beep()
```

Description

This function will cause the user's Mac to beep. This can be useful if you need to get their attention, would like to notify them of a completed task, or for debugging your Widget's script.

Example

```
if (done)
    beep();
```

bytesToUIString()

Turns a number of bytes into a UI-friendly string

Synopsis

```
string = bytesToUIString(integer)
```

Description

Often times it is desirable to turn a given number of bytes into a string such as "1K" or "34.2M". This function does exactly that.

Example

```
print( "There is " + bytesToUIString( numBytes ) + " memory
available" );
```

Availability

Available in version 2.0 or later.

chooseColor()

Puts up a standard color picker dialog box and allows the user to choose a color

Synopsis

```
string = chooseColor( [string] );
```

Description

You can use this function to display the standard color picker for the platform and allow the user to select a color. You can optionally pass the initial color that is selected as a parameter. This function will return the color as a hex string (e.g. "#FF0000") or null if the user cancelled the dialog.

Example

```
print( chooseColor( "#EEEEEE" ) );
```

Availability

Available in version 2.0 or later.

chooseFile()

Puts up a standard file dialog box and allows the user to choose a file

Synopsis

```
file = chooseFile([string | array]);
```

Description

You can use this function to display the standard open dialog for the platform and allow the user to select a file. You can also optionally pass a single extension or an array of extensions into this function to limit what kinds of files the user can choose. If the dialog is cancelled by the user, null is returned.

Example

```
print( chooseFile() ); // select anything
print( chooseFile( ".png" ) ); // just PNG files
print( chooseFile( new Array( ".png", ".jpg" ) ) )
```

Availability

Available in version 2.0 or later.

chooseFolder()

Puts up a standard file dialog box and allows the user to choose a folder

Synopsis

```
file = chooseFolder();
```

Description

You can use this function to display the standard open dialog for the platform and allow the user to select a folder. If the dialog is cancelled by the user, null is returned.

Example

```
print( chooseFolder() );
```

Availability

Available in version 2.0 or later.

convertPathToHFS()

converts a UNIX style path to an HFS one

Synopsis

```
convertPathToHFS(myPath[, localize])
```

Description

Converts a UNIX style path (with '/'s) to a Mac HFS style path (with a volume name and ':'s). If the optional second boolean parameter is `true` then the returned path is localized in the current system language. Note that the file referenced by the path must exist for conversion to succeed if the localized path is requested.

Example

```
convertPathToHFS('/Users/joe/foo.txt');
```

yields:

```
Macintosh HD:Users:joe:foo.txt
```

On a German system:

```
convertPathToHFS('~/Movies', true)
```

yields:

```
Macintosh HD:Benutzer:joe:Filme
```

Windows Notes

This function returns an empty string on Windows.

convertPathToPlatform()

converts a JavaScript style path to a platform specific one

Synopsis

```
convertPathToPlatform(myPath[, forDisplay])
```

Description

Converts a JavaScript style file path ("/foo/bar/baz") to a platform style path (e.g. on Windows, "\\foo\\bar\\baz"). Note that by default the path is escaped (has any backslashes doubled), ready for use with `runCommand()`. If you want a path suitable for display to a user, specify `true` for the optional second parameter.

On the Macintosh this function does nothing (paths are already in the correct format).

Example

```
convertPathToPlatform('c:/temp/foo.txt');
```

On Windows yields:

```
c:\\temp\\foo.txt
```

And

```
convertPathToPlatform('c:/temp/foo.txt', true);
```

On Windows yields:

```
c:\temp\foo.txt
```

closeWidget()

closes the Widget

Synopsis

```
closeWidget()
```

Description

Shuts down the currently running Widget as if the user had selected **Close Widget** from the context menu.

Example

```
answer = alert("Do you wish to continue?", "Yes", "No");

if (answer == 2)
  closeWidget();
```

escape()

encode a string to safely be used as a URL

Synopsis

`escape(`*`string`*`)`

Attributes

`String` A string containing text that is intended for use as a URL.

Returns

A string that contains the argument but with characters unsuitable for URLs converted

to their escaped counterparts.

Description

This is useful if you're collecting information from a user preference that you would like to pass via a URL. It saves having to validate the strings yourself before passing them off to the URL handler.

Example

```
// The single quote, spaces and ampersand will be
// replaced with URL escape characters
mySearch = "Konfabulator's FAQ & JavaScript Reference";
openURL("google.com/search?q=" + escape(mySearch);
```

See Also

`unescape()`

focusWidget()

brings the Widget to the foreground

Synopsis

`focusWidget()`

Description

Brings the Widget to the foreground on the user's desktop. Useful when responding to a hotkey.

Note that if a Widget comes to the foreground when not requested the user will become annoyed and will probably trash the Widget.

Example

```
<hotkey name="hkey1">
  <key>F2</key>
  <modifier>command+control</modifier>
  <onKeyUp>focusWidget();</onKeyUp>
</hotkey>
```

form()

preference-like form generation for acquiring user input via a dialog

Synopsis

`form(`*`fieldArray, [dialogTitle], [confirmButtonLabel],`*

[cancelButtonLabel])

Description

`form()` takes up to four arguments, the first is an Array of FormField objects (which have the same arguments as preference objects which are defined in the XML). The array of form fields is used to define a dialog which is displayed to the user. When the user presses the "confirm" button the `form()` function returns an array of strings representing the values entered in the form (if the "dismiss" button is pressed, null is returned). The remaining arguments are, in order, a title for the dialog, the label for the "confirm" button and the label for the "dismiss" button. The last 3 arguments are optional.

Example

```
var formfields = Array();

formfields[0] = new FormField();
formfields[0].name = 'name1';
formfields[0].type = 'text';
formfields[0].title = 'Text Pref Title';
formfields[0].defaultValue = 20;
formfields[0].description = 'This is a description of a text field.';

formfields[1] = new FormField();
formfields[1].title = 'Basic Field';

formfields[3] = new FormField();
formfields[3].name = 'name4';
formfields[3].title = Checkbox Pref Title';
formfields[3].type = 'checkbox';
formfields[3].defaultValue = 1;
formfields[3].description = 'This is a description of a checkbox
field.';

formResults = form(formfields, 'my title', 'Save It And Continue');

if (formResults != null) {
    print("formResults = " + formResults);
} else {
    print("form was cancelled");
}
```

include()

include the contents of another JavaScript file

Synopsis

include(*string*)

Description

Include the contents of the specified file at the current point in the script. This is done in a manner internally equivalent to the old style method:

```
eval(runCommand("cat onload.js"));
```

The only difference being that `include()` arranges for any error messages to have correct filenames and line numbers.

Example

```
include("onload.js");
```

isApplicationRunning()

returns true if a given application is running

Synopsis

```
isApplicationRunning(string)
```

Description

You can use this function to decide if an application is currently running. This is often useful before you do something like invoke AppleScript on the Mac or COM on Windows. Pass the name of the application you are interested in, not a full path.

Example

```
// On Mac
if ( isApplicationRunning( "iTunes" ) )

// OnWindows
if ( isApplicationRunning( "itunes.exe" ) )
```

Platform Notes

Please note that this is an area where you have to use the exact name for the platform. As shown above, on Windows, you might use "itunes.exe" whereas on Macintosh you'd use just "itunes" for the application name parameter.

Availability

Available in version 2.0 or later.

konfabulatorVersion()

returns the current version of the Widget Engine as a string

Synopsis

```
konfabulatorVersion()
```

Description

You can use this function for informational purposes, or perhaps to control how your code behaves on different versions of the Widget Engine.

Example

```
print( "This version is " + konfabulatorVersion() );
```

log()

display a string in the debug window with a timestamp

Synopsis

`log(`*`string`*`)`

Description

Often used for debugging. Note you will need to specify:

```
<debug>on</debug>
```

in the Widget's XML to see the output.

Example

```
log("idx = " + idx);
```

openURL()

open the specified URL in the default web browser

Synopsis

`openURL(`*`validURL`*`)`

Description

Using this function to launch a URL will cause the URL to be launched using the appropriate application set in the user's Internet System Preferences. This function will return true if the argument is a well formed URL, otherwise false is returned. Note that even a well formatted URL may point to a non-existent resource so the Widget Engine

would return true while your browser may still complain.

Example

```
openURL("http://widgets.yahoo.com");
openURL("ftp://myname:pa55w0rd@ftp.mysite.com");
```

See Also

`escape()`, `unescape()`, `URL.fetch()`

play()

play a sound file

Synopsis

`play(`*`pathToSound[, truncate]`*`)`

Description

Supported formats are MP3, AIFF, AU, WAV and SND. The call returns immediately and the sound is played asynchronously. `pathToSound` must point to a valid sound file either somewhere on the user's hard drive, or inside the Widget's bundle. The optional second boolean parameter specifies whether the new sound should truncate (stop) any currently playing sounds.

Examples

```
play("sounds/sample.mp3");

play("sounds/bark.aiff", true);
```

popupMenu()

displays a popup menu at a specified location

Synopsis

popupMenu(menuItems, x, y);

Description

This function allows you to display a popup menu at a specified location. You pass an array of menuItem objects in the first parameter, much like you would for a context menu. The x and y coordinates are passed in window coordinates.

You should only call this function while handling a mouse down event.

Example

```
// put up a popup menu where the mouse is
<onMouseDown>
  var items = new Array;

  items[0] = new MenuItem;
  items[0].title = "This is item 1";
  items[0].enabled = true;
  items[1] = new MenuItem;
  items[1].title = "this is item 2";
  items[1].enabled = true;

  popupMenu( items, system.event.hOffset,
      system.event.vOffset );
</onMouseDown>
```

Availability

Available in version 2.1 or later.

print()

print a string in the debug window

Synopsis

`print(`*`string`*`)`

Description

Often used for debugging. Note you will need to specify:

```
<debug>on</debug>
```

in the Widget's XML to see the output.

Example

```
print("idx = " + idx);
```

prompt()

text entry field for user input

Synopsis

`prompt(`*`<promptText>`*`, `*`[defaultValue], [dialogTitle],`*
`[confirmButtonLabel], [cancelButtonLabel]``)`

`promptText`

Prompt to be displayed to the user.

defaultValue
Value to populate the text field with (and the value that will be returned if the user does not change anything).

dialogTitle
Title that will be used for the dialog.

confirmButtonLabel
Label for the button that confirms the user's changes to the dialog.

cancelButtonLabel
Label used for the button that cancels the dialog.

Description

Used to get a string of text back from the user. This is a subset of the functionality found in `form()`, and is provided for ease of coding. Note that `null` is returned if the user cancels this dialog.

Example

```
result = prompt("Name:", "Your Name","Name Dialog", "OK",
"Cancel");

if (!result)
    result = "no name";
```

random()

return a random number

Synopsis

`random([lower_limit, upper_limit])`

Description

Generate a random number, optionally within given limits. Note that the lower limit may be included in the returned values while the upper never is.

Example

```
// This will return a random number between 0 and 64K
number = random();
// This will return a random number between 0 and 100
percentage = random(100);
// This will return a random number between 27 and 72
number = random(27,72);
```

reloadWidget()

causes the Widget to reload itself

Synopsis

```
reloadWidget()
```

Description

Sometimes it's desirable to have a Widget restart. Calling this function gives the same result as if the user had held down **Command** while choosing the Widget in the gear menu.

See Also

`closeWidget(), focusWidget()`

resolvePath()

normalize a file system file path

Synopsis

`resolvePath(`*`pathToFile`*`)`

Description

This function can make the following changes in the provided path:

- Expand an initial tilde expression (e.g. `~/Pictures`) to the correct directory (e.g. `/Users/joe`)
- Reduce empty components and references to the current directory (that is, the sequences "`//`" and "`/./`") to single path separators.
- In absolute paths only, resolve references to the parent directory (that is, the component "`..`") to the real parent directory if possible, which consults the file system to resolve each potential symbolic link.
- In relative paths, because symbolic links can't be resolved, references to the parent directory are left in place.
- Remove an initial component of `/private` from the path if the result still indicates an existing file or directory (checked by consulting the file system).
- If the path is an HFS+ alias, the file name that is the target of the alias is returned (note that this only works for the final path element, aliases embedded in paths will not be resolved and may have to be handled specially if expected).
- If the given path is "`.`", it is expanded to the fully qualified path of the current directory.

Example

```
realPath = resolvePath(myPath);
```

resumeUpdates()

allows Widgets to visually update dynamically

Synopsis

```
resumeUpdates()
```

Description

JavaScript code can affect the layout of all the objects in the Widgets window. If the Widget is complex it can be quite inefficient (and possibly unattractive) to have these changes appear individually. By bracketing areas of code that rearrange the visible parts of the Widget with suppressUpdates() and resumeUpdates() the Widget author can control what the user sees.

See Also

`suppressUpdates(), updateNow()`

runCommand()

executes a shell command and returns the result

Synopsis

`runCommand(`*`string`*`)`

Description

This function allows any command in the UNIX layer of the operating system to be executed and the results saved in a string variable. Note that only commands the user has privilege for can be run.

If the last character of the result is a newline it is removed.

Example

```
str = runCommand("ls -l /");
print(str);
```

runCommandInBg()

executes a shell command in the background

Synopsis

```
runCommandInBg(string, tag)
```

Description

This takes a UNIX command and a *tag*, runs the command in the background (i.e. does not wait for it to complete) and when it does complete causes a global action called `onRunCommandInBgComplete` to be triggered and sets the value of a variable called *tag* to the results of the command (the value of `system.event.data` is set to the name of the *tag*). The order in which commands finish may be unrelated to the order which they were started.

Note that the value of `system.event.data` changes whenever a background command finishes and that this can happen in the middle of an action if you have multiple commands in the background at one time. You should save the value at the beginning of the `onRunCommandInBgComplete` action to avoid unexpected results. Also note that the *tag* specifies the **name** of the variable which will receive the data, not the variable itself.

Example

```
<action trigger="onLoad">
  var yahooData;
  runCommandInBg("curl www.yahoo.com", "yahooData");
</action>

<action trigger="onRunCommandInBgComplete">
  print("onRunCommandInBgComplete for tag: " +
         system.event.data);
  print("Yahoo's home page is " +
         yahooData.length + " bytes");
</action>
```

saveAs

Display standard SaveAs dialog box

Synopsis

```
string = saveAs()
```

Description

At times it might be useful to ask the user where to save a file. This function allows you to display the standard dialog box to allow the user to choose a destination folder. The path to the folder will be returned. If the user cancelled the dialog box, null is returned.

Example

```
destination = saveAs();
if ( destination != null )
    saveFileTo( destination );
```

Availability

Available in version 2.0 or later.

savePreferences()

saves the Widget's preferences

Synopsis

```
savePreferences()
```

Description

Normally a Widget's preferences are automatically saved whenever the user edits them using the Widget Preferences panel or when the Widget exits. If a Widget is manipulating preference values in JavaScript it can ensure they are saved to disk in a timely manner by calling this function.

showWidgetPreferences()

opens the Widget's preference panel

Synopsis

```
showWidgetPreferences()
```

Description

This opens the Widget Preferences panel just as if the user had selected **Widget Preferences** from the context menu. It is often used to provide a preferences button on the face of the Widget or to get initial preferences the first time a Widget runs.

sleep()

suspend script execution

Synopsis

```
sleep(number)
```

Description

Suspends execution of the Widget's code for the specified number of milliseconds (one thousandth of a second).

Example

```
// pause script for one second
sleep(1000);
```

speak()

speak text

Synopsis

`speak(`*`string`*`)`

Synopsis

```
speak(theText)
```

Description

This function speaks the given text in the default voice of the computer (which can be set using the Speech panel in the System Preferences).

Example

```
speak("Now there's something you don't see everyday.");
speak("Unless you're me.");
```

suppressUpdates()

makes Widgets wait to visually update

Synopsis

```
suppressUpdates()
```

Description

Suppresses screen updating until a corresponding call to `resumeUpdates()`. Alternatively, updates can be performed manually using `updateNow()`. Suppressing updates can improve performance or hide messy interim states from the Widget user.

See Also

```
resumeUpdates(), updateNow()
```

tellWidget()

Sends a message to another Widget.

Synopsis

tellWidget(nameOrPath, message);

Description

You can use tellWidget to do inter-Widget messaging. In order for this to work successfully, the Widget you are sending the message to must have an onTellWidget handler. The message is passed in system.event.data. It's completely up to the Widget author to decide what is an acceptable message. In its simplest form, you could send Javascript over and eval() it. That is not very safe however, because you have no idea what the JavaScript in question might do. So Widget authors might want to consider a special set of terms that they support via messaging like this. For example, a webcam might support 'reload' as an action it supports.

```
<action trigger="onTellWidget">
   if ( system.event.data == "reload" )
      reloadCamPicture();
</action>
```

In our PIM Overview Widget, we settled on the following structure:

```
msg = action ":" params
params = (param) (";" param)*
param = name "=" value
```

"action", "name" and "value" are merely strings. Value could be placed in quotes, perhaps.

You will note that you can send a message to either the name of the Widget (as long as it is either running, or lives in the user's Widgets folder), or the path to the Widget. Also, at present, this is a one-way message. Later versions will allow a response to be sent back.

This is implemented in AppleScript on Mac and COM on Windows. This means you could write scripts in AppleScript on the Mac, or on Windows, you could use JScript or VB, etc. to send messages to a Widget.

Availability

Available in version 2.0 or later.

Platform Notes

Currently, Windows will launch the Widget if it can find it. Mac will only launch the Widget if it's not running and you give it a full path. In the future there will instead be a boolean parameter to control this more exactly.

Security Notes

You should always double-check the input message and never merely eval() the message.

unescape()

unencode string that contains URL escapes

Synopsis

`unescape(`*`string`*`)`

Description

This is the inverse of `escape()`.

Example

```
encURL = escape(url);

url = unescape(encURL);
```

updateNow()

force a Widget's visual update

Synopsis

`updateNow()`

Description

By using `suppressUpdates()` and calling updateNow() as needed the Widget author can completely control how their Widget is displayed. Note that if your code fails to call `updateNow()` when updates are suppressed then the screen may not reflect the true state of the Widget.

Example

```
updateNow();
```

See Also

`resumeUpdates(), suppressUpdates()`

yahooCheckLogin()

verify whether a Widget is currently logged in

Synopsis

```
boolean yahooCheckLogin()
```

Description

This function is used to see whether the user is currently logged into their Yahoo! account. If the function returns true, they are, and if it returns false, well guess what: they're not. You can use this to predicate whether or not you can use Yahoo! APIs which require a logged in user successfully.

Example

```
var loggedIn = yahooCheckLogin();
```

See Also

`yahooLogin(), yahooLogout()`

Availability

Available in version 3.0 or later.

yahooLogin()

ensure a user is logged in, authenticating if necessary

Synopsis

```
boolean yahooLogin()
```

Description

This function is used to log into a user's Yahoo! account if you are using Web APIs that require a logged in user. If yahooCheckLogin() returns false, you'd normally call this function. It will present the standard Yahoo! Widget Engine login dialog to prompt the user for their user name and password.

This call generally works asynchronously. If the user is already logged in, yahooLogin() will simply return true and you are done. If yahooLogin() returns false, then the user needs to authenticate. This will happen automatically while your Widget is free to do other things. When the user finally authenticates, you will receive notification via the onYahooLoginChanged action. In there you can call yahooCheckLogin() to see if the user is now logged in.

In general you should always wait for the onYahooLoginChanged event if you are not logged in when your Widget starts up before trying to do anything with APIs that require a logged-in user.

Example

```
var loggedIn = yahooCheckLogin();

if ( !loggedIn )
  yahooLogin();

// go about our business while the user authenticates.

// In your XML:
<action trigger="onYahooLoginChanged">
  if ( yahooCheckLogin() )
       RefreshInformation();
  else
       LoggedOut();
</action>
```

Note that in our onYahooLoginChanged action we also have to deal with the case where we've logged out. When this happens, you might need to clear your display and present a way for the user to log in again. You might get logged out at unexpected times, so you must be prepared to deal with this.

Availability

Available in version 3.0 or later.

See Also

`yahooCheckLogin(), yahooLogout()`

Availability

Available in version 3.0 or later.

yahooLogout()

log out of a user's Yahoo! account

Synopsis

`yahooLogout()`

Description

This function requests that the Widget Engine log out of a user's Yahoo! account. This will return immediately while the request is pending. On completion, all Widgets will receive an onYahooLoginChanged action and yahooCheckLogin will return false. Widgets must be prepared to deal with the situation where the user has logged out. This might happen if the previously valid credentials have timed out, so always be prepared to deal with a logout. Also, keep in mind this call affects all Widgets who require the user's Yahoo! credentials to use Yahoo! APIs. It does not just affect the current Widget.

Example

```
yahooLogout();

// go about our business while the logout occurs.

// In your XML:
<action trigger="onYahooLoginChanged">
  if ( yahooCheckLogin() )
       RefreshInformation();
  else
       LoggedOut();
</action>
```

Availability

Available in version 3.0 or later.

See Also

`yahooCheckLogin(), yahooLogout()`

Availability

Available in version 3.0 or later.

These give JavaScript code access to various system settings and hardware information.

COM

functions to call COM interfaces on Windows

Functions

```
createObject
connectObject
disconnectObject
```

Description

The COM object is an interface to enable your Widgets to call an a COM interface in the system. For example, you could use it to talk to iTunes (if you didn't use our built-in support), MSN Messenger, Outlook, etc.

You can connect to any COM object using COM.createObject(progID | CLSID).

Example

```
var it = COM.createObject( "iTunes.Application" );

track = it.CurrentTrack;

print( track.Album );
print( track.Artist );
```

Here's another sample which prints some info from MSN Messenger:

```
messenger = COM.createObject( "Messenger.UIAutomation" );

contacts = messenger.MyContacts;
num = contacts.Count;
for ( i = 0; i < num; i++ ) {
   contact = contacts.Item( i );
   print( " " + contact.FriendlyName + " " + contact.Status );
}
```

This code works only when logged into MSN Messenger. Things like the Status of a contact can be found out via the web, MSDN, etc.

It is also possible to hook up an 'event sink'. This allows you to have an application inform you when things change (like buddy status, etc.) as opposed to having to poll for that information. COM.connectObject tells the Widget Engine that you want to be informed of an objects events. COM.disconnectObject tells us that you no longer wish

to be informed of those events. You should always disconnect a sink when you are done with it.

And finally, a note on object references. You should try to be sure to clear your references out to null whenever you know you are through with an interface. This is because COM requires a certain amount of refcounting to work, and Javascript's garbage collection mentality can confuse it a bit. For those reasons, it's good to always clear your references to interfaces (COM objects) you've gotten when you are done with them. It's not strictly required, but you'll feel better about yourself.

Availability

COM support is available in version 2.0 and later.

COM.connectObject

connect an event sink to listen to an object's events

Synopsis

```
COM.connectObject( object, prefix )
```

Description

This function allows you to connect to an object created or otherwise received from using the COM interface to listen to events. Many objects in the COM world have an event interface which you can listen to for events. For example, you can connect to the main iTunes application object and it will tell you when the player starts and stops, as well as when the application is about to quit.

The object you pass in must have been either created via COM.createObject or gotten via a call to a COM object that you created (e.g if you receive a track from iTunes, you can use that iTunes track, provided it has an event interface).

For the second parameter, you pass a prefix for a function that will be called when an event occurs. For example, the iTunes COM interface will send out "OnPlayerPlayEvent" when the player starts a track, passing it the track that it started to play. The prefix helps us find the function to call. When the event occurs, it will try to call a function called <prefix>OnPlayerPlayEvent(). The following example shows us listening to that event.

Example

```
var iTunesObj = COM.createObject( "iTunes.Application" );

COM.connectObject( iTunesObj, "iTunes_" );

function iTunes_OnPlayerPlayEvent( track )
{
   print( "Started to play " + track.Name );
}
```

Note that our function is called iTunes_OnPlayerPlayEvent which is the combination of the prefix we specified, and the name of the COM method which gets invoked. Also note that we were passed a parameter which is another COM object representing the track. From there we are able to reference its Name property.

When you are done with an object and listening to its events, you should take care to call disconnectObject.

There can be only one established event sink per object at a time.

Notes

This function will throw an exception if it cannot successfully connect. It is recommended to call this inside a try/catch handler.

Availability

Available in version 2.0 or later.

COM.createObject

creates a COM object via ProgID or CLSID

Synopsis

```
COM.createObject( progID | CLSID )
```

Description

This is the main place to start your romp through the COM forest. The trick here is to find out what interfaces you can call. Unfortunately, there's no simple way to figure this out short of looking in a COM browser and using trial and error. Some information is available via the web (a quick Yahoo search will yield stuff if you search for things like "automation" "COM" and your favorite application). You can also use regedit to look for ProgIDs. But really, the COM browser provided by Visual Studio is probably the best way (and cheapest if you already have Visual Studio).

Example

```
var iTunesObj = COM.createObject( "iTunes.Application" );
```

Availability

Available in version 2.0 or later.

COM.disconnectObject

disconnect an event sink previously established with connectObject

Synopsis

```
COM.disconnectObject( object )
```

Description

After you are done with an event sink created with a call to connectObject you should call this function to break the connection. It is in some cases important that you do this before releasing the main COM object due to the way some applications are written.

Example

```
COM.disconnectObject( iTunesObj );
```

Availability

Available in version 2.0 or later.

filesystem

get information from and interact with the file system

Synopsis

```
filesystem
```

Description

The `filesystem` object provides access to the underlying files and directories of the system on which the Widget is running. See below for details of the individual functions and attributes.

Attributes

```
volumes
```

Functions

copy()

```
emptyRecycleBin()/emptyTrash()
getDirectoryContents()
getDisplayName()
getFileInfo()
getRecycleBinInfo()/getTrashInfo()
isDirectory()
itemExists()
move()
moveToRecycleBin()/moveToTrash()
open()
openRecycleBin()/openTrash()
readFile()
reveal()
writeFile()
```

Availability

The filesystem object is available in version 1.8 or later.

filesystem.copy()

Copies a file or files to a location

Synopsis

```
filesystem.copy( path, destination );
```

Description

This function allows you to copy a file or files to a specified destination. The source can be a single path, or an array of paths.

If copying one file, the destination need not exist. In this case, it is assumed the destination is a new file name for the file. If the destination does exist, it is assumed it specifies a directory in which to copy the new file.

If copying multiple files, the destination must be a directory that exists.

This function returns true if successful, false otherwise.

Example

```
// to copy a file to a folder
filesystem.copy( "myfile.txt", "/Users/evoas" );

// to duplicate a file
filesystem.copy( "myfile.txt", "myfile_copy.txt" );
```

Availability

Available in version 2.0 or later.

filesystem.emptyRecycleBin()
filesystem.emptyTrash()

Empties the system trash

Synopsis

```
filesystem.emptyRecycleBin()
filesystem.emptyTrash()
```

Description

This function merely empties the trash/recycle bin. If the user has their settings set to see the warning dialog, etc. it will come up automatically.

The function has two names to reflect the different terms used on the two platforms, Windows and Mac OS X.

Example

fileystem.emptyTrash();

Availability

Available in version 2.0 or later.

filesystem.getDirectoryContents()

get the names of the files in a directory

Synopsis

```
array = filesystem.getDirectoryContents(directory, recurse)
```

Description

Retrieves the names of the files in the specified directory optionally recursing (descending) into each subdirectory.

Example

```
fileList = filesystem.getDirectoryContents(path, false);
```

Behavior Notes

As of version 2.0, this function now behaves the same on both Mac and Windows. It will now always return an array of names that are rooted at the directory you pass and never a full path. Previously, Windows would return full paths if you passed in a full path.

Availability

Available in version 1.8 or later.

filesystem.getDisplayName()

Returns the user-friendly name of a file

Synopsis

```
filesystem.getDisplayName( path )
```

Description

This function returns the display name for a file path. The display name is essentially what you'd see in the Finder or Explorer. For example, if a file's extension is supposed to be hidden, this function will remove it. Essentially you are guaranteed to print the same name for a file path that you'd see in the OS.

Example

```
print( filesystem.getDisplayName( "C:\" ) );
```

Availability

Available in version 2.0 or later.

filesystem.getFileInfo()

Returns information about a file or directory

Synopsis

filesytem.getFileInfo(path)

Description

This function returns a small object which describes the file or directory passed to it. The object has the following attributes:

```
size
isDirectory
isHidden
lastModified
```

While there is an isDirectory function, this information comes with that tidbit as well to save the number of filesystem operations necessary to traverse a tree of files.

Example

```
info = filesystem.getFileInfo( "myfile.txt" );

print( "myfile is " + bytesToUIString( info.size ) + " in size" );
```

Availability

Available in version 2.0 or later.

filesystem.getRecycleBinInfo()
filesystem.getTrashInfo()

get information about files that have been deleted but not yet purged

Synopsis

```
filesystem.getRecycleBinInfo()
filesystem.getTrashInfo()
```

Description

Retrieves the number and total size of files that are in the user's trash or recycle bin. An object with two members is returned, `numItems` and `size`.

The function has two names to reflect the different terms used on the two platforms, Windows and Mac OS X.

Example

```
mytrash = filesystem.getRecycleBinInfo();
mesg = myTrash.numItems + " items (" + myTrash.size +
                " bytes)";
```

Availability

Available in version 1.8 or later.

filesystem.isDirectory()

determine if a given path is a directory

Synopsis

`filesystem.isDirectory(`*`path`*`)`

Description

Returns `true` if the given `path` is a directory, `false` otherwise.

Example

```
isDir = filesystem.isDirectory(path);
```

Availability

Available in version 1.8 or later.

filesystem.itemExists()

determine if a given path exists

Synopsis

`filesystem.itemExists(`*`path`*`)`

Description

Returns `true` if the given `path` exists (is a file or a directory), `false` otherwise.

Example

```
exists = filesystem.itemExists(path);
```

Availability

Available in version 1.8 or later.

fileystem.move()

Moves a file or files to a location

Synopsis

```
filesystem.move( path, destination );
```

Description

This function allows you to move a file or files to a specified destination. The source can be a single path, or an array of paths. The destination must be a directory that exists. This function returns true if successful, false otherwise.

Note that at present you cannot use this function to rename a file. In fact, there is no rename facility in the current release.

Example

```
filesystem.move( "myfile.txt", "/Users/evoas" );
```

Availability

Available in version 2.0 or later.

filesystem.moveToRecycleBin()
filesystem.moveToTrash()

delete items by moving them to the trash or recycle bin

Synopsis

```
filesystem.moveToRecycleBin(files)
filesystem.moveToTrash(files)
```

Description

Sends the specified file or files (provide an array of Strings to delete multiple files at a time).

The function has two names to reflect the different terms used on the two platforms, Windows and Mac OS X.

Example

```
filesystem.moveToTrash(myTmpFile);
```

Availability

Available in version 1.8 or later.

filesystem.open()

Opens a file based on its file type/extension

Synopsis

```
filesystem.open( path )
```

Description

You can use this function to open an arbitrary file with the correct application. For example, passing a Widget file path into this function will open the Widget in the Yahoo! Widget Engine (i.e. it will run the Widget, as expected). Passing a folder in will open it in Finder/Explorer.

Example

```
filesystem.open( "PIM Overview.widget" );
```

Availability

Available in version 2.0 or later.

filesystem.openRecycleBin()
filesystem.openTrash()

Opens the folder that contains the items in the trash/recycle bin.

Synopsis

```
filesystem.openRecycleBin()
filesystem.openTrash()
```

Description

This function is the equivalent of double-clicking on the Trash or Recycle Bin icons. It opens up a window showing the contents of the Trash/Recycle Bin.

The function has two names to reflect the different terms used on the two platforms, Windows and Mac OS X.

Example

```
filesytem.openTrash();
```

Availability

Available in version 2.0 or later.

filesystem.readFile()

read a text file into a string or array

Synopsis

```
filesystem.readFile( path [,asLines] )
```

Description

This function is used to read in a text file into either a string or array variable. If the optional second parameter is true, the file is read and broken into lines and an array of those lines is returned. Else just one long string of the contents is returned.

The Widget Engine can read most of the typical text file formats, but works best with either UTF-16 or UTF-8 encodings.

Example

```
var data = filesystem.readFile( "myfile.txt" );

var lines = filesystem.readFile( "myfile.txt", true );
```

Availability

Available in verison 2.0 or later.

filesystem.reveal()

make the system file browser display an item in context

Synopsis

filesystem.reveal(*path*)

Description

Causes the system file browser (**Explorer** on Windows, the **Finder** on Mac OS X) to display the directory containing the specified item. This is useful for revealing file system items to the user.

Example

```
filesystem.reveal(myPath);
```

Availability

Available in version 1.8 or later.

filesystem.volumes

Array of currently mounted volumes

Descripton

The volumes property of the filesystem object contains an array of all the currently mounted volumes. Each entry in the array is a small object with the following attributes:

```
path
freeBytes
totalBytes
```

You can use these in concert with functions like bytesToUIString or filesystem.getDisplayName.

Example

```
vols = filesystem.volumes;
for ( a in vols )
{
   print( "Volume " +
      filesystem.getDisplayName( vols[a].path ) +
      " (path " + vols[a].path + ") has a capacity of " +
      bytesToUIString( vols[a].totalBytes ) + " and " +
      bytesToUIString( vols[a].freeBytes ) + " free." );
}
```

Availability

Available in version 2.0 or later.

filesystem.writeFile()

Writes a string or array to a text file

Synopsis

```
filesystem.writeFile( path, string | array )
```

Description

This function writes out a file given either a string or an array of strings. If given a string, the data is written out as-is. If passed an array, the data is written out separated by return characters "\n". Currently, this function always writes files as UTF-8.

Example

```
filesystem.writeFile( "myfile.txt", myData );
```

Availability

Available in verison 2.0 or later.

screen

Information about the display

Attributes

```
availHeight
availLeft
availTop
availWidth
colorDepth
height
pixelDepth
resolution
width
```

Synopsis

```
screen
```

Description

The `screen` object has various attributes which describe the metrics of the current screen (the display the main window of a Widget is mostly on). See below for details of the individual attributes.

Example

```
for (a in screen)
  print("screen." + a + ": " + eval("screen." + a));
```

screen.availHeight

the current screen's available height

Synopsis

```
screen.availHeight
```

Description

The number of pixels available vertically on the screen most of the Widget's window occupies. This value omits space taken by things like the system menubar and the Dock.

Example

```
myWindow.vOffset = screen.availHeight - 30;
```

screen.availLeft

the leftmost available position on the screen

Synopsis

```
screen.availLeft
```

Description

The first available position at the left of the screen most of the Widget's window occupies that is not occupied by a system feature such as the Dock.

Example

```
myWindow.hOffset = screen.availLeft + 30;
```

screen.availTop

the topmost available position on the screen

Synopsis

```
screen.availTop
```

Description

The first available position at the top of the screen most of the Widget's window occupies that is not occupied by a system feature such as the menubar.

Example

```
myWindow.vOffset = screen.availTop + 10;
```

screen.availWidth

the current screen's available width

Synopsis

```
screen.availWidth
```

Description

The number of pixels available vertically on the screen most of the Widget's window occupies. This value omits space taken by system features like the Dock.

Example

```
myWindow.width = screen.availWidth / 4;
```

screen.colorDepth

the current screen's color depth

Synopsis

```
screen.colorDepth
```

Description

The number of bits per pixel available on the screen most of the Widget's window occupies.

Example

```
alert("Bits per pixel: " + screen.colorDepth);
```

screen.height

the current screen's height

Synopsis

```
screen.height
```

Description

The number of pixels available vertically on the screen most of the Widget's window occupies. Normally `screen.availHeight` provides a more useful measure of the screen's height.

Example

```
myWindow.vOffset = screen.availHeight - 30;
```

screen.pixelDepth

the current screen's color depth

Synopsis

```
screen.pixelDepth
```

Description

The number of bits per pixel available on the screen most of the Widget's window occupies. This is a synonym for `screen.colorDepth` and is provided for compatibility.

Example

```
alert("Bits per pixel: " + screen.pixelDepth);
```

screen.resolution

the current screen's resolution

Synopsis

```
screen.resolution
```

Description

The raster resolution in dots per inch (dpi) of the screen most of the Widget's window occupies.

Example

```
alert("Screen resolution: " + screen.resolution);
```

screen.width

the current screen's width

Synopsis

```
screen.width
```

Description

The number of pixels available horizontally on the screen most of the Widget's window occupies. Normally `screen.availWidth` provides a more useful measure of the screen's width.

Example

```
myWindow.hOffset = screen.width - 80;
```

system

Information about the machine or environment

Attributes

```
airport/wireless
appearance
battery
clipboard
cpu
event
languages
memory
mute
platform
volume
widgetDataFolder
userDocumentsFolder
userDesktopFolder
userPicturesFolder
userMoviesFolder
userMusicFolder
userWidgetsFolder
applicationsFolder
temporaryFolder
trashFolder
```

Description

The system object is your interface to things about the machine you are running on or some aspect of the environment. For example, you can get information about the state of the battery or wireless connection if present.

system.airport
system.wireless

built in support for accessing WiFi/AirPort information

Attributes

`available`	true if WiFi/AirPort is installed.
`info`	a summary of WiFi/AirPort status.
`network`	the name of the current network.
`noise`	the connection's noise level.
`powered`	true if WiFi/AirPort is powered on.
`signal`	the connection's signal level.

Description

The settings and status of an installed WiFi/AirPort card are available through the `system.airport` or `system.wireless` object.

The WiFi/AirPort Widget makes extensive use of this object.

Example

```
if (system.airport.available && system.airport.powered)
  alert("Current network is " + system.airport.network);

if(system.wireless.available && system.wireless.powered)
  alert("Current network is "+ system.wireless.network);
```

system.airport.available
system.wireless.available

determine if an WiFi/AirPort (or other compatible wireless card) is installed

Synopsis

```
system.airport.available
system.wireless.available
```

Description

The `available` property returns a boolean true/false value that corresponds to the availability of the wireless device capable of connecting to a network.

Example

```
if (! system.airport.available)
    signal_status.src = "NoCard.png";
else
    signal_status.src = "Signal.png";

if (! system.wireless.available)
    signal_status.src = "NoCard.png";
else
    signal_status.src = "Signal.png";
```

See Also

`system.airport.signal, system.wireless.signal`

system.airport.info
system.wireless.info

WiFi/AirPort status summary

Synopsis

```
system.airport.info
system.wireless.info
```

Description

A brief, human readable description of WiFi/AirPort status.

Example

```
alert(system.airport.info);
alert(system.wireless.info);
```

system.airport.network
system.wireless.network

return name of current WiFi/AirPort network

Synopsis

```
system.airport.network
system.wireless.network
```

Description

This attribute contains the name of the current WiFi/AirPort network, if any.

Example

```
alert("AirPort network " + system.airport.network + " in use");

alert("WiFi network " + system.wireless.network + " in use");
```

system.airport.noise
system.wireless.noise

the noise level of the current WiFi/AirPort connection

Synopsis

```
system.airport.noise
system.wireless.noise
```

Description

This attribute contains a numeric value which indicates the level of noise on the current WiFi/AirPort connection.

This value is not generally reliable.

Example

```
if (system.airport.noise > 20)
  status.src = "noisy.png";

if (system.wireless.noise > 20)
  status.src = "noisy.png";
```

Windows Note

On Windows, this attribute is always zero.

system.airport.powered
system.wireless.powered

boolean that indicates if the WiFi/AirPort card is on or off

Synopsis

```
system.airport.powered
system.wireless.powered
```

Description

This boolean variable indicates whether the WiFi/AirPort is currently turned on or off. Use this to decide whether to access the other WiFi/AirPort status attributes.

Example

```
if (system.airport.available && system.airport.powered)
  alert("Current network is " + system.airport.network);

if(system.wireless.available && system.wireless.powered)
  alert("Current network is "+ system.wireless.network);
```

system.airport.signal
system.wireless.signal

return the signal strength of the current WiFi/AirPort connection

Synopsis

```
system.airport.signal
```

system.wireless.signal

Description

The signal property of the WiFi/AirPort object returns a number value that corresponds to the signal strength of the wireless network the device is connected to.

It should be noted that in this release of the Widget Engine, the range is 0-75 and is not a linear mapping to Apple's signal strength.

Example

```
theStrength = system.airport.signal;
if ( theStrength =&lt; 33 )
   signalBars.src = "halfFull.png"

theStrength = system.wireless.signal;
if ( theStrength =&lt; 50 )
   signalBars.src = "halfFull.png"
```

system.appearance

the current system appearance

Synopsis

system.appearance

Description

The current appearance of the system. As of Mac OS X 10.3 this will only be Blue or Graphite.

If your Widget uses images that you would like to be specific to the current Mac OS X Appearance, simply use this variable to get the running Appearance and adjust your image source file appropriately.

It should be noted that these get returned with initial caps, so make sure you test for the words "Blue" and "Graphite", not "blue" and "graphite".

Note: in this release of the Widget Engine we do not support notifying Widgets of an Appearance change.

On Windows, the value Blue is always returned.

Example

```
if (system.appearance == "Graphite")
  header.src = "graphiteHeader.png";
else
  header.src = "aquaHeader.png";
```

system.battery

built in support for accessing battery and UPS information

Synopsis

```
system.battery
```

Description

The battery number is an array that's 0 based. Single battery laptops will always be `battery[0]`, however when run on a machine with dual batteries, the expected primary bay registers as battery 1, and the optional battery bay registers as battery 0. The number of batteries installed in the current system is available in `system.batteryCount`.

Note: in this release of the Widget Engine only one battery is supported on Windows (however, information about it is an aggregate of all batteries in the system).

system.battery[*n*].currentCapacity

the charge in the battery

Synopsis

```
system.battery[batteryNumber].currentCapacity
```

Description

Current percentage charge of the battery.

system.battery[*n*].isCharging

charging state

Synopsis

```
system.battery[batteryNumber].isCharging
```

Description

True if battery is being charged (i.e. it is at less that 100% capacity and the system is plugged into AC power).

system.battery[*n*].isPresent

is battery installed

Synopsis

```
system.battery[batteryNumber].isPresent
```

Description

True if battery is physically present.

system.battery[*n*].maximumCapacity

the maximum charge of the battery

Synopsis

```
system.battery[batteryNumber].maximumCapacity
```

Description

Maximum capacity of the battery (since capacity is represented as a percentage, this is always `100`).

system.battery[*n*].name

the name of the battery

Synopsis

```
system.battery[batteryNumber].name
```

Description

The human readable name of the battery.

system.battery[*n*].powerSourceState

the current source of power

Synopsis

```
system.battery[batteryNumber].powerSourceState
```

Description

Returns "AC Power" or "Battery Power" based on whether the system is plugged in or not.

system.battery[*n*].timeToEmpty

minutes until battery is discharged

Synopsis

```
system.battery[batteryNumber].timeToEmpty
```

Description

This value is in minutes. A value of -1 means the system is still determining how fast the battery is draining (also known as the "calculating" phase).

system.battery[*n*].timeToFullCharge

minutes until battery is fully charged

Synopsis

```
system.battery[batteryNumber].timeToFullCharge
```

Description

This value is in minutes. A value of -1 means the system is still determining how fast the battery is charging (also known as the "calculating" phase).

Note, `currentCapacity` is generally a more reliable determination of how charged the battery is.

Example

```
alert(system.battery[0].timeToFullCharge + ' minutes to full
charge');
```

system.battery[*n*].transportType

battery communication channel

Synopsis

```
system.battery[batteryNumber].transportType
```

Description

"Internal" or method of UPS communication

system.batteryCount

the number of batteries installed

Synopsis

```
system.batteryCount
```

Description

The number of batteries installed in the current system is available in `system.batteryCount`. Normally this is 1 but some laptops support more so any Widget that intends to work with batteries should take this into account.

Example

```
for (b = 0; b < system.batteryCount; b++)
  totalTime += system.battery[b].timeToEmpty;
```

Windows Notes

Currently the value of this attribute is always `1` (though the power available from all batteries is reported).

system.clipboard

accesses the current system clipboard

Synopsis

```
system.clipboard
```

Description

`system.clipboard` contains the text (if any) on the system clipboard. Setting this attribute will load the system clipboard with that data, removing anything there previously.

Example

```
myText = system.clipboard;
myNewText = "--<(" + myText + ")>--";

system.clipboard = myNewText;
```

system.cpu

contains information about the current CPU load

Synopsis

```
system.cpu
```

Description

`system.cpu` is an object with several members that summarize the level of activity of the system CPU (members are detailed below).

Note that the underlying mechanism that gathers this data has a resolution of 1 second so that is as quickly as this information can change. In other words, polling `system.cpu` more than once per second is not useful.

Example

```
for (a in system.cpu)
  print("system.cpu." + a + ": " + eval("system.cpu." + a));
```

system.cpu.activity

returns information about the current CPU activity

Synopsis

```
system.cpu.activity
```

Description

`system.cpu.activity` contains the current percentage load of the CPU. If the machine is very busy it will be near 100. It is the sum of the other `system.cpu` members `user`, `sys` and `nice`. It represents the load of the machine as a whole, no matter how many processors it has.

Example

```
load = system.cpu.activity;
```

system.cpu.idle

returns information about idle CPU cycles

Synopsis

```
system.cpu.idle
```

Description

`system.cpu.idle` provides a measure of how much of the CPU is available for more work. It is a percentage.

Example

```
idle_percent = system.cpu.idle;
```

system.cpu.nice

returns information about raised priority CPU cycles

Synopsis

`system.cpu.nice`

Description

`system.cpu.nice` is a measure of how much of the CPU is occupied with tasks whose priority has been raised (normal processes are reported as `system.cpu.user`).

Example

```
priorityTasks = system.cpu.nice;
```

Windows Notes

The value of this attribute is always zero.

system.cpu.numProcessors

returns the number of processors in the system

Synopsis

`system.cpu.numProcessors`

Description

`system.cpu.numProcessors` indicates how many processors there are in the current system.

Example

```
if (system.cpu.numProcessors == 2)
  system = "Dualie";
```

system.cpu.sys

returns information about system CPU cycles

Synopsis

```
system.cpu.sys
```

Description

`system.cpu.sys` contains the percentage of the CPU occupied with system tasks (as opposed to user tasks).

Example

```
systemTime = system.cpu.sys;
```

system.cpu.user

returns information about user CPU cycles

Synopsis

```
system.cpu.user
```

Description

`system.cpu.user` is a measure of how much of the CPU is occupied with normal tasks (as opposed to system tasks).

Example

```
userTasks = system.cpu.user;
```

Example

```
userTasks = system.cpu.user;
```

system.event

information about the last event received

Synopsi

```
system.event
```

Description

`system.event` contains a variety of information about the last event the Widget received (typically as the result of a user action such as a mouse click). See below for details.

Example

```
for (a in system.event)
  print("system.event." + a + ": " + eval("system.event." + a));
```

system.event.hOffset, system.event.vOffset

mouse position in window coordinates

Synopsis

`system.event.hOffset, system.event.vOffset`

Description

`system.event.hOffset` and `system.event.vOffset` contain the position of the mouse in coordinates relative to the Widget's window.

The values are accessible in any JavaScript context so it's possible for the values to be outside the Widget's window (they are still relative to it though).

Example

```
print("Mouse: " + system.event.hOffset + ", " +
   system.event.hOffset);
```

system.event.key

the key that triggered the current key event

Synopsis

`system.event.key`

Description

`system.event.key` contains the key that was pressed.

Example

```
print("Key: " + system.event.key);
```

See Also

`system.event.keyString`

system.event.keyString

the name of the key that triggered the current key event

Synopsis

```
system.event.keyString
```

Description

`system.event.keyString` contains the name of the key that was pressed, i.e. the name of special keys, e.g. "PageUp" or the hex value of normal keys.

Example

```
print("Key Name: " + system.event.keyString);
```

See Also

`system.event.key`

system.event.modifiers

the state of the modifier keys for the current key event

Synopsis

```
system.event.modifiers
```

Description

`system.event.modifiers` contains the modifiers when a key event is being processed (in an `onKeyDown` or `onKeyUp`). It can be a combination of:

```
shift, capslock, control, option, numlock, help, fkey
```

For example:

```
shift+control
```

Example

```
print("Modifiers: " + system.event.modifiers);
```

system.event.screenX, system.event.screenY

mouse position in screen coordinates

Synopsis

```
system.event.screenX, system.event.screenY
```

Description

`system.event.screenX` and `system.event.screenY` contain the position of the mouse in screen coordinates.

Example

```
print("Mouse: " + system.event.screenX + ", " +
   system.event.screenY);
```

system.event.scrollDelta

the delta the mouse wheel moved

Synopsis

```
system.event.scrollDelta
```

Description

`system.event.scrollDelta` contains the number of lines you should scroll, positive or negative. This is only valid during an onMouseWheel handler.

Example

```
frame.scrollY += (system.event.scrollDelta * 10);
```

system.event.timestamp

the time at which the event occurred

Synopsis

```
system.event.timestamp
```

Description

`system.event.timestamp` contains a Date object which records the time an event occurred (as opposed to the time it is processed by JavaScript). This can be used for a variety of things, for example, determining the length of time a key was held down.

Example

```
print("Event happened at: " + system.event.timestamp);
```

Windows Notes

Available on Windows as of version 2.1.1.

system.event.x, system.event.y

mouse position in object coordinates

Synopsis

```
system.event.x, system.event.y
```

Description

`system.event.x` and `system.event.y` contain the position of the mouse in coordinates relative to the current object (the one whose action was triggered).

Example

```
print("Mouse: " + system.event.x + ", " +
   system.event.y);
```

system.languages

returns the current set of languages preferred by the user

Synopsis

```
system.languages
```

Description

`system.languages` contains the list of languages the user has specified in the **International** System Preference panel. Element 0 is their primary language, 1 their second choice, and so forth.

You can only read this setting, it cannot be changed except by using the System Preferences panel.

Example

```
print("system.languages: " + system.languages);

system.languages: en,de,ja,fr,nl,it,es,zh_TW
```

system.memory

information about the physical/virtual memory of a machine

Attributes

```
availPhysical
availVirtual
load
totalPhysical
totalVirtual
```

Description

You can inspect the amount of memory on a machine via this system object. Please note that at present, the virtual memory numbers are somewhat suspicious on both platforms.

system.memory.availPhysical

amount of available physical memory

Description

Returns the number of bytes of available physical memory. Use bytesToUIString to turn this into something more user-friendly if you wish.

Example

```
print( "available RAM: " +
   bytesToUIString( system.memory.availPhyiscal ) );
```

Availability

Available in version 2.0 or later.

system.memory.availVirtual

amount of available virtual memory

Description

Returns the number of bytes of available virtual memory. Use bytesToUIString to turn this into something more user-friendly if you wish.

Example

```
print( "avail virtual memory: " +
   bytesToUIString( system.memory.availVirtual ) );
```

Availability

Available in version 2.0 or later.

Notes

This number is not quite accurate at present, particularly on Windows.

system.memory.load

percentage of used memory

Description

Returns a number from 0 to 100 indicating the current amount of physical RAM that is in use.

Example

```
print( "current system load: " +
   system.memory.load + "%" );
```

Availability

Available in version 2.0 or later.

system.memory.totalPhysical

amount of physical RAM installed

Description

Returns the number of bytes of installed physical memory. Use bytesToUIString to turn this into something more user-friendly if you wish.

Example

```
print( "Installed RAM: " +
   bytesToUIString( system.memory.totalPhysical ) );
```

Availability

Available in version 2.0 or later.

system.memory.totalVirtual

amount of total virtual memory

Description

Returns the number of bytes of total virtual memory. Use bytesToUIString to turn this

into something more user-friendly if you wish.

Example

```
print( "total virtual memory: " +
   bytesToUIString( system.memory.totalVirtual ) );
```

Availability

Available in version 2.0 or later.

Notes

This number is not quite accurate at present, particularly on Windows.

system.mute

get or set the mute state of your system volume

Synopsis

```
system.mute
```

Description

This variable reflects whether the machine's sound is muted. Setting it to `true` mutes the system sound.

Examples

```
// Find out if the machine is muted or not
if ( system.mute )
    print("What?  I can't hear you!");
else
    print("I can hear sounds from my Mac!");

// Turn off system sound
system.mute = true;
```

system.platform

contains the type of system the Widget is running on

Synopsis

```
system.platform
```

Description

This variable contains the current platform a Widget is executing on.

Example

```
print("platform: " + system.platform);
```

On Mac OS X results in:

```
platform: macintosh
```

On Windows results in:

```
platform: windows
```

system.userDocumentsFolder
system.userDesktopFolder
system.userPicturesFolder
system.userMoviesFolder
system.userMusicFolder
system.userWidgetsFolder
system.applicationsFolder
system.temporaryFolder
system.trashFolder

system variables that contains the names of various user folders

Synopsis

```
system.userDocumentsFolder
system.userDesktopFolder
system.userPicturesFolder
system.userMoviesFolder
system.userMusicFolder
system.userWidgetsFolder
system.applicationsFolder
system.temporaryFolder
system.trashFolder
```

Description

These variables contain the paths of various user-centric and system folders. The correct locations can be determined in a platform independent manner using these variables.

Example

```
print("userMusicFolder: " + system.userMusicFolder);
```

On Mac OS X results in:

```
userMusicFolder: /Users/joe/Music
```

On Windows results in:

```
userMusicFolder:
    c:/Documents and Settings/joe/My Documents/My Music
```

system.volume

get or set the system audio volume

Synopsis

```
system.volume
```

Description

This variable reflects the current audio volume. Setting it to a number between 0 and 16 to changes the system volume level. Setting the volume to 0 (zero) is effectively the same as setting `system.mute` to `true`.

Example

```
// Set the audio volume to 50%
system.volume = 8;
```

system.widgetDataFolder

name of folder where Widget can safely store data

Synopsis

```
system.widgetDataFolder
```

Description

This variable contains the name of a folder on the user's hard disk where persistent data (or even data that needs to be cached for a short length of time) can safely be saved by the Widget. Historically, Widgets have tried to save data inside their own bundles but this has various drawbacks chiefly that the location may not be writable. Each Widget gets a separate folder which is created if it does not already exist.

Example

```
saveFileName = system.widgetDataFolder + "/data";
```

Platform Notes

On the Mac, the location of this folder is "~/Library/Application Support/Konfabulator/Widgets". On the PC, it is located in "C:\Documents and Settings\<user>\Local Settings\Application Data\Yahoo\Widget Engine\Widget Data"

Application Attributes and Functions

These give JavaScript code access to certain applications allowing remote control and retrieval of data.

Currently the only supported application is **iTunes** (available from http://www.apple.com/itunes for both Windows and Mac OS X).

iTunes

get information from and interact with iTunes

Synopsis

```
iTunes
```

Description

The `iTunes` object allows remote control and display of **iTunes** track and artist information. See below for details of the individual functions and attributes.

Availability

The iTunes object is available in version 1.8 or later.

iTunes.backTrack()

tell iTunes to move to the previous track

Synopsis

```
iTunes.backTrack()
```

Description

Tell iTunes to move to the previous track.

Example

```
iTunes.backTrack();
```

See Also

```
iTunes.nextTrack()
```

iTunes.fastForward()

tell iTunes to fast forward within the current track

Synopsis

```
iTunes.fastForward()
```

Description

Tell iTunes to skip forward within the current track.

Example

```
iTunes.fastForward();
```

See Also

```
iTunes.rewind()
```

iTunes.nextTrack()

tell iTunes to move to the next track

Synopsis

```
iTunes.nextTrack()
```

Description

Tell iTunes to move to the next track.

Example

```
iTunes.nextTrack();
```

See Also

```
iTunes.backTrack()
```

iTunes.pause()

tell iTunes to pause playback

Synopsis

```
iTunes.pause()
```

Description

Tell iTunes to pause playback.

Example

```
iTunes.pause();
```

See Also

`iTunes.resume()`

iTunes.play()

tell iTunes to start playing the current track

Synopsis

`iTunes.play()`

Description

Tell iTunes to play the current track.

Example

```
iTunes.play();
```

See Also

`iTunes.pause()`

iTunes.playPause()

tell iTunes to toggle between playing and pause

Synopsis

`iTunes.playPause()`

Description

Tell iTunes to play if it's currently paused, or pause if it's currently playing.

Example

```
iTunes.playPause();
```

iTunes.playerPosition

returns the current position within the current track

Synopsis

```
iTunes.playerPosition
```

Description

This attribute returns the current position (in seconds) within the currently playing track. Setting it moves the playback position to the specified number of seconds into the track.

Example

```
iTunes.playerPosition;
```

iTunes.playerStatus

returns a string describing the current state of iTunes

Synopsis

```
iTunes.playerStatus
```

Description

This attribute returns one of the following strings: stopped, paused, playing, fast forwarding, rewinding or unknown.

Example

```
currentState = iTunes.playerStatus;
```

iTunes.random
iTunes.shuffle

reflects the shuffle state of iTunes

Synopsis

```
iTunes.random
```

Description

This attribute reflects the shuffle state of iTunes. If the current playlist is set to shuffle, it is true, false otherwise. Setting the attribute changes iTunes' shuffle state.

Example

```
iTunes.random = 1;
```

```
shuffleState = iTunes.shuffle;
```

iTunes.repeatMode

reflects the current repeat mode of iTunes

Synopsis

```
iTunes.repeatMode
```

Description

This attribute returns one of the following strings: `off`, `one` or `all` indicating the current repeat mode. The repeat mode can be set by setting the attributes to one of those strings.

Example

```
mode = iTunes.repeatMode;

iTunes.repeatMode = 'off';
```

iTunes.resume()

tell iTunes to resume playback

Synopsis

```
iTunes.resume()
```

Description

Tell iTunes to resume playback after being paused.

Example

```
iTunes.resume();
```

See Also

```
iTunes.pause()
```

iTunes.rewind()

tell iTunes skip backwards

Synopsis

```
iTunes.rewind()
```

Description

Tell iTunes to skip backwards in the current track.

Example

```
iTunes.rewind();
```

See Also

`iTunes.fastForward()`

iTunes.running

returns whether iTunes is currently running

Synopsis

```
iTunes.running
```

Description

Use this attribute to determine if iTunes is currently running.

Example

```
iTunes.running;
```

iTunes.stop()

tell iTunes to stop playing

Synopsis

```
iTunes.stop()
```

Description

Tell iTunes to stop playing.

Example

```
iTunes.stop();
```

See Also

`iTunes.play()`

iTunes.streamURL

returns the URL of the currently playing stream

Synopsis

```
iTunes.streamURL
```

Description

If iTunes is currently playing an audio stream, this attribute will contain the URL of the stream.

Example

```
url = iTunes.streamURL;
```

iTunes.trackAlbum

returns the name of the current album

Synopsis

```
iTunes.trackAlbum
```

Description

This attribute contains the name of the current album (if known). If a stream is playing, the name of the stream will appear here.

Example

```
currAlbum = iTunes.trackAlbum;
```

iTunes.trackArtist

returns the artist of the currently playing track

Synopsis

```
iTunes.trackArtist
```

Description

This attribute contains the name of the artist of the current album (if known). If a stream is playing, this information is not available.

Example

```
iTunes.trackArtist;
```

iTunes.trackLength

returns the length of the current track

Synopsis

```
iTunes.trackLength
```

Description

This attribute contains the length of the currently playing track. If a stream is playing, this information is not available.

Example

```
len = iTunes.trackLength;
```

iTunes.trackRating

reflects the rating of the current track

Synopsis

```
iTunes.trackRating
```

Description

This attribute contains the rating of the currently playing track. Setting the attribute changes the current track's rating in iTunes. If a stream is playing, this information is not available.

Example

```
rating = iTunes.trackRating;
```

iTunes.trackTitle

returns the title of the current track

Synopsis

```
iTunes.trackTitle
```

Description

This attribute contains the title of the currently playing track. If a stream is playing, this information may not be available.

Example

```
title = iTunes.trackTitle;
```

iTunes.trackType

returns the type of the current track

Synopsis

```
iTunes.trackType
```

Description

This attribute contains the type of the currently playing track. It can include one of the following: `audio file`, `audio cd track`, `audio stream`, `audio device`, `shared library` or `unknown`

Example

```
tt = iTunes.trackType;
```

iTunes.version

returns the version of iTunes

Synopsis

```
iTunes.version
```

Description

This attribute contains the version of the copy of iTunes that is being controlled.

Example

```
log("iTunes Version: " + iTunes.version);
```

iTunes.volume

reflects the volume iTunes plays at

Synopsis

```
iTunes.volume
```

Description

This attribute reflects the volume iTunes is playing at. It can vary between `0` and `100`. Assign a value to the attribute to change the volume.

Example

```
iTunes.volume = 60;
```

Widget Engine Object Properties and functions

This section contains Javascript functions and properties that aren't covered in the XML-centric section of this document.

Frame Properties and Functions

Frame.addSubview()

adds a view to a frame as a subview

Synopsis

```
void Frame.addSubview( object );
```

Description

This function adds an object to a frame. Currently Image, Text, TextArea, Frame, and ScrollBar objects can be added to a frame object as a child.

Example

```
myFrame.addSubview( myImage );
```

Availability

Available in version 3.0 or later.

Frame.home()

scrolls a frame to the upper left

Synopsis

```
void Frame.home();
```

Description

This function basically sets the scrollX and scrollY properties to 0, 0.

Example

```
myFrame.home();
```

Availability

Available in version 3.0 or later.

Frame.hScrollBar

the horizontal scroll bar for a frame

Synopsis

```
Frame.hScrollBar
```

Description

You can set or query the horizontal scroll bar of a frame with this property. Attaching a scroll bar will do all the automatic setup for communicating between the frame and the scroll bar.

Example

```
myFrame.hScrollBar
```

Availability

Available in version 3.0 or later.

Frame.end()

scrolls a frame to the bottom left

Synopsis

```
void Frame.end();
```

Description

This function scrolls a frame to the bottom of its contents.

Example

```
myFrame.end();
```

Availability

Available in version 3.0 or later.

Frame.lineDown()

scrolls a frame one line down

Synopsis

```
void Frame.lineDown();
```

Description

This function scrolls a frame one line down by the amount specified by the frame's

vLineSize property.

Example

```
myFrame.lineDown();
```

Availability

Available in version 3.0 or later.

Frame.lineLeft()

scrolls a frame one line left

Synopsis

```
void Frame.lineLeft();
```

Description

This function scrolls a frame one line left by the amount specified by the frame's hLineSize property.

Example

```
myFrame.lineLeft();
```

Availability

Available in version 3.0 or later.

Frame.lineRight()

scrolls a frame one line right

Synopsis

```
void Frame.lineRight();
```

Description

This function scrolls a frame one line right by the amount specified by the frame's hLineSize property.

Example

```
myFrame.lineRight();
```

Availability

Available in version 3.0 or later.

Frame.lineUp()

scrolls a frame one line up

Synopsis

```
void Frame.lineUp();
```

Description

This function scrolls a frame one line up by the amount specified by the frame's vLineSize property.

Example

```
myFrame.lineUp();
```

Availability

Available in version 3.0 or later.

Frame.pageDown()

scrolls a frame one page down

Synopsis

```
void Frame.pageDown();
```

Description

This function scrolls a frame one page down by the height of the frame minus one line height as specified by vLineHeight.

Example

```
myFrame.pageDown();
```

Availability

Available in version 3.0 or later.

Frame.pageLeft()

scrolls a frame one page left

Synopsis

```
void Frame.pageLeft();
```

Description

This function scrolls a frame one page left by the width of the frame minus one line

height as specified by hLineHeight.

Example

```
myFrame.pageLeft();
```

Availability

Available in version 3.0 or later.

Frame.pageRight()

scrolls a frame one page right

Synopsis

```
void Frame.pageRight();
```

Description

This function scrolls a frame one page right by the width of the frame minus one line height as specified by hLineHeight.

Example

```
myFrame.pageRight();
```

Availability

Available in version 3.0 or later.

Frame.pageUp()

scrolls a frame one page up

Synopsis

```
void Frame.pageUp();
```

Description

This function scrolls a frame one page up by the height of the frame minus one line height as specified by vLineHeight.

Example

```
myFrame.pageUp();
```

Availability

Available in version 3.0 or later.

Frame.removeFromSuperview()

detaches an object from its parent view

Synopsis

```
void Frame.removeFromSuperview()
```

Description

Use this method to remove an object from a window. You might do this because you are done with it and are reloading new information. Once detached, you can merely clear your reference to it by setting it to null if you are done with it, or put it into another window or frame if you like.

When your Widget's minimumVersion is set to 3.0, you must call this to remove an object from a window. Deleting the reference will not work.

Example

```
myObject.removeFromSuperview();
```

Availability

Available in version 3.0 or later.

Frame.subviews

array of subviews in this frame

Synopsis

```
array Frame.subviews (read-only)
```

Description

This property contains all the views contained in this frame, as a Javascript array. If the frame has no subviews, this property will be set to null.

Example

```
var x = myFrame.subviews[0];
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Frame.superview

the parent view of this view

Synopsis

```
frame|root Frame.superview (read-only)
```

Description

This property contains the parent view of this view. The parent can either be a Frame object or a Root object. If the view has no parent, this property will be set to null.

Example

```
var parent = myFrame.superview;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Frame.vScrollBar

the vertical scroll bar for a frame

Synopsis

```
Frame.vScrollBar
```

Description

You can set or query the vertical scroll bar with this property. Attaching a scroll bar will do all the automatic setup for communicating between the frame and the scroll bar.

Example

```
myFrame.vScrollBar
```

Availability

Available in version 3.0 or later.

Image.fade()

fade in or out an image

Synopsis

```
Image.fade(start, end, duration)
```

Description

The `fade()` command will cause an image to fade from a starting opacity to a finishing opacity. `duration` specifies the time (in tenths of a second) you want the animation to last for.

Example

```
newOpacity = 0;
myImage.fade(myImage.opacity, newOpacity, 1);
```

Image.moveTo()

move an image from point a to point b via animation

Synopsis

```
Image.moveTo(newX, newY, duration)
```

Description

The image's origin (`hOffset`, `vOffset`) is moved to the new coordinates specified by `newX` and `newY`. `duration` specifies the time (in tenths of a second) you want the animation to last for. The move of the object is animated (which is what makes this different from just changing `hOffset` and `vOffset`).

Example

```
myImage.moveTo(50, 50, 3);
```

Image.reload()

reload an image from disk

Synopsis

```
Image.reload()
```

Description

Use this method to reload an image from disk. This is especially useful if your Widget

makes use of a graphic that is being constantly updated by an external process as it defeats the normal caching behavior of the Image object.

Example

```
myImage.reload();
```

Image.removeFromSuperview()

detaches an image object from its parent view

Synopsis

```
Image.removeFromSuperview()
```

Description

Use this method to remove an image object from a window. You might do this because you are done with it and are reloading new information. Once detached, you can merely clear your reference to it by setting it to null if you are done with it, or put it into another window or frame if you like.

When your Widget's minimumVersion is set to 3.0, you must call this to remove an object from a window. Deleting the reference will not work.

Example

```
myImage.removeFromSuperview();
```

Availability

Available in version 3.0 or later.

Image.slide()

slide an image in a specified direction and duration

Synopsis

```
Image.slide(direction, amountOfImageToConceal, duration)
```

Description

The slide() command is an animation effect used to hide and reveal parts of the user interface. It is used when you want to slide an image in a particular direction, and have it disappear into itself rather than just move. `duration` specifies the time (in tenths of a second) you want the animation to last for.

Example

```
myImage.slide("up,left", 50, 3);
```

Image.superview

the parent view of this view

Synopsis

```
frame|root Image.superview (read-only)
```

Description

This property contains the parent view of this view. The parent can either be a Frame object or a Root object. If the view has no parent, this property will be set to null.

Example

```
var parent = myImage.superview;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Root Properties and Functions

The root is the container for all views in a window. When an object such as an image has it's window property set, this is the view it is added to internally. It is here for completeness of the view hierarchy, and has only two properties and one function. The root can be accessed as a property of the window it belongs to. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Root.addSubview()

adds a view to the root of a window

Synopsis

```
void Root.addSubview( object );
```

Description

This function adds an object to a root view for a window. Currently Image, Text, TextArea, Frame, and ScrollBar objects can be added as a child.

You can attach a view to the top level of a window by using this method, or by simply setting the `window` property of an object. Both approaches yield the exact same result.

Example

```
myWindow.root.addSubview( myImage );
```

Availability

Available in version 3.0 or later.

Root.subviews

array of subviews

Synopsis

```
array Root.subviews (read-only)
```

Description

This property contains all the views contained in this view, as a Javascript array. If the root has no subviews, this property will be set to null.

Example

```
var x = myWindow.root.subviews[0];
```

Availability

Available in version 3.0 or later.

Root.superview

the parent view of this view

Synopsis

```
null Root.superview (read-only)
```

Description

This property contains the parent view of this view. This property always returns null. It is here for completeness of the hierarchy.

Example

```
var shouldBeNull = myWindow.root.superview;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

ScrollBar.removeFromSuperview()

detaches an object from its parent view

Synopsis

```
ScrollBar.removeFromSuperview()
```

Description

Use this method to remove an scroll bar object from a window. You might do this because you are done with it and are reloading new information. Once detached, you can merely clear your reference to it by setting it to null if you are done with it, or put it into another window or frame if you like.

When your Widget's minimumVersion is set to 3.0, you must call this to remove an object from a window. Deleting the reference will not work.

Example

```
myScrollbar.removeFromSuperview();
```

Availability

Available in version 3.0 or later.

ScrollBar.setRange()

sets the min and max of a scroll bar

Synopsis

```
ScrollBar.setRange(int min, int max)
```

Description

Use this method to define the range of values that can be expressed by a scroll bar. You cannot modify the min and max properties directly, so you must use this function to set those properties. The value will be pinned to this range if the value falls outside of the new range when this function is called.

You will normally not need to deal with this function if you are attaching a scrollbar to a frame object. In that case, the range is set automatically for you.

Example

```
myScrollbar.setRange(0, 100);
```

Availability

Available in version 3.0 or later.

ScrollBar.setThumbInfo()

replace the images that comprise the thumb of a scroll bar

Synopsis

```
ScrollBar.setThumbInfo(int offset, string|array images )
```

Description

While the standard scroll bar object allows you to customize the thumb color, this might not meet every Widget's need. To facilitate more customization, you can pass either one or three image paths to this function as a string or array, respectively, and the thumb will use those images to make the thumb. If you pass one image, this implies a fixed size thumb and your scroll bar will not be proportional. This may be appropriate if you are trying to make a slider object instead. If you pass three, the first and third image paths specify the caps to use for top/bottom. The second image is a stretchable center. Therefore, three part scroll bar thumbs are always proportional.

The offset parameter controls how far into the scroll bar the thumb should be positioned. If you specify 3 for example, your scroll bar images will appear 3 pixels to the right of the left edge of the scroll bar view. For horizontal scroll bars, this is the number down to nudge the thumb.

Please note that at present, horizontal scroll bar images have to be created with the same orientation as vertical ones. So you need to design your scroll bars horizontally, but rotate them 90 degrees clockwise before chopping them up and saving out the pieces.

Example

```
myScrollbar.setThumbInfo( 1, new Array( "images/topCap.png",
       "images/middle.png", "images/bottomCap.png" ) );

myScrollbar.setThumbInfo( 0, "images/fixedthumb.png" ) );
```

Availability

Available in version 3.0 or later.

ScrollBar.setTrackInfo()

replace the images that comprise the track of a scroll bar

Synopsis

```
ScrollBar.setThumbInfo(int offset, int topLimit,
```

```
int bottomLimit, string|array images )
```

Description

While the standard scroll bar object allows you to customize the thumb color, this might not meet every Widget's need. To facilitate more customization, you can pass either one or three image paths to this function (as a string or array, respectively), and the track will use those images to draw itself. If you pass three, the second image is a stretchable center.

The offset parameter controls how far into the scroll bar the track should be positioned (perhaps you have a one-line track for a slider look). If you specify 3 for example, your images will appear 3 pixels in from the left edge of the scroll bar view. For horizontal scroll bars, this is the number down to nudge the track images.

The topLimit and bottomLimit parameters are used to set the limits of how far the thumb can travel. They are basically 'bumper' limits. If you want to ensure the thumb can not travel anywhere less than 5 pixels from the top of your scroll bar, specify 5 for the topLimit.

Please note that at present, horizontal scroll bar images have to be created with the same orientation as vertical ones. So you need to design your scroll bars horizontally, but rotate them 90 degrees clockwise before chopping them up and saving out the pieces.

Example

```
myScrollbar.setTrackInfo( 0, 2, 2, new Array( "images/topCap.png",
        "images/middle.png", "images/bottomCap.png" ) );
```

Availability

Available in version 3.0 or later.

ScrollBar.superview

the parent view of this view

Synopsis

```
frame|root ScrollBar.superview (read-only)
```

Description

This property contains the parent view of this view. The parent can either be a Frame object or a Root object. If the view has no parent, this property will be set to null.

Example

```
var parent = myScrollBar.superview;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Text Methods

Text.fade()

fade in or out a text object

Synopsis

`Text.fade(`*`start, end, duration`*`)`

Description

The `fade()` command will cause a text object to fade from a starting opacity to a finishing opacity. `duration` specifies the time (in tenths of a second) you want the animation to last for.

Example

```
newOpacity = 0;
myText.fade(myText.opacity, newOpacity, 6);
```

Text.moveTo()

move text from point a to point b via animation

Synopsis

`Text.moveTo(`*`newX, newY, duration`*`)`

Description

The text's origin (`hOffset`, `vOffset`) is moved to the new coordinates specified by `newX` and `newY`. `duration` specifies the time (in tenths of a second) you want the animation to last for. The move of the object is animated (which is what makes this different from just changing `hOffset` and `vOffset`).

Example

```
myText.moveTo(50, 50, 3);
```

Text.removeFromSuperview()

detaches a text object from its parent view

Synopsis

```
Text.removeFromSuperview()
```

Description

Use this method to remove a text object from a window. You might do this because you are done with it and are reloading new information. Once detached, you can merely clear your reference to it by setting it to null if you are done with it, or put it into another window or frame if you like.

When your Widget's minimumVersion is set to 3.0, you must call this to remove an object from a window. Deleting the reference will not work.

Example

```
myText.removeFromSuperview();
```

Availability

Available in version 3.0 or later.

Text.slide()

slide a text object in a specified direction and duration

Synopsis

`Text.slide(`*`direction, amountOfTextToConceal, duration`*`)`

Description

The `slide()` function is used when you want to slide a text object in a particular direction, and have it disappear into itself rather than just move. `duration` specifies the time (in tenths of a second) you want the animation to last for.

Example

```
myText.slide("up,left", 50, 8);
```

Text.superview

the parent view of this view

Synopsis

```
frame|root Text.superview (read-only)
```

Description

This property contains the parent view of this view. The parent can either be a Frame object or a Root object. If the view has no parent, this property will be set to null.

Example

```
var parent = myText.superview;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

TextArea Methods

TextArea.focus()

make the current textarea object the focus of key presses

Synopsis

```
TextArea.focus()
```

Description

The `focus()` function will make the given textarea be the one to which typed keys are sent. It is most useful when there are several textareas on a Widget and you want to move the insertion point from one to another. The textarea must be `editable` for this to be effective.

When focus is acquired, the `onGainFocus` action is called for the text area in version 3.0 or later.

Example

```
mytextarea.focus();
```

TextArea.loseFocus()

relinquishes keyboard focus if the text area currently is the focus

Synopsis

```
TextArea.loseFocus()
```

Description

The `loseFocus()` function releases the keyboard focus from the text area if the text area is the current focus (via a call to `focus()`). This function is useful for clearing

the focus after the user perhaps enters a value in a text area used as a search field. There is no need to call this when the window the text area loses the focus, as it will automatically lose focus in that case.

When focus is lost, the `onLoseFocus` action is called for the text area in version 3.0 or later.

Example

```
mytextarea.loseFocus();
```

Availability

Available in version 3.0 or later.

TextArea.rejectKeyPress()

control whether keys are accepted by a textarea

Synopsis

```
TextArea.rejectKeyPress()
```

Description

The `rejectKeyPress()` function is used in the `<onKeyPress>` action to control whether the current key press will affect the textarea.

Example

```
<onKeyPress>
<!--
  // Convert all typed characters to uppercase
  var key = system.event.key;

  if (key.charCodeAt(0) >= "A".charCodeAt(0) &&
      key.charCodeAt(0) <= "z".charCodeAt(0))
  {
    // Tell the text area to ignore this keyPress as
    // we are replacing it with our own
    ta1.rejectKeyPress();

    // Append an upper case copy of the key pressed
    // (the insertion point is a 0 length selection)
    ta1.replaceSelection(key.toUpperCase());
  }
// -->
</onKeyPress>
```

TextArea.removeFromSuperview()

detaches a text area object from its parent view

Synopsis

```
Image.removeFromSuperview()
```

Description

Use this method to remove a text area object from a window. You might do this because you are done with it and are reloading new information. Once detached, you can merely clear your reference to it by setting it to null if you are done with it, or put it into another window or frame if you like.

When your Widget's minimumVersion is set to 3.0, you must call this to remove an object from a window. Deleting the reference will not work.

Example

```
myTextArea.removeFromSuperview();
```

Availability

Available in version 3.0 or later.

TextArea.replaceSelection()

replace the current selection in a textarea with a string

Synopsis

`TextArea.replaceSelection(`*`string`*`)`

Description

The `replaceSelection()` function replaces the current selection in the textarea with the given string. Note that the "cursor" or "insertion point" is actually a selection of zero length so, if nothing is selected in the textarea, using `replaceSelection()` has the effect of inserting the given string at the current cursor position.

Example

```
replacement = "new text";
mytextarea.replaceSelection(replacement);
```

TextArea.select()

select text in the textarea

Synopsis

```
TextArea.select(start, end)
```

Description

The select function changes the selection in the text area. Characters from start to end are selected. As a special case, the position –1 means "the end of the text", thus:

```
mytextarea.select(0, -1);
```

selects all the text.

To set the position of the "cursor" or "insertion point" specify a selection of zero length, for example:

```
mytextarea.select(10, 10);
```

To set the insertion point after any text already in the textarea you would use:

```
mytextarea.select(-1, -1);
```

When the insertion point is set, the contents of the text area are scrolled so it is visible to the user.

Example

```
mytextarea.select(5, 15);
```

TextArea.superview

the parent view of this view

Synopsis

```
frame|root TextArea.superview (read-only)
```

Description

This property contains the parent view of this view. The parent can either be a Frame object or a Root object. If the view has no parent, this property will be set to null.

Example

```
var parent = myTextArea.superview;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Timer.reset()

Restarts a timer's countdown

Synopsis

```
Timer.reset()
```

Description

The `reset()` function will cause a timer to start its countdown over. For example, if you had a timer that was on a one-minute interval, and 30 seconds after it started to run you called `reset()`, it would start its one minute countdown over again. So instead of firing in 30 seconds, it would start over and fire in one minute.

A good example of when this is useful is if you were trying to implement some sort of idle timer. Let's say you wanted to do something in your Widget if the user hasn't interacted with it in 15 seconds. If the user clicked your Widget, you could start a timer that will fire in 15 seconds. If the user clicks again, you can merely reset the timer, starting the 15 second countdown over. Eventually, after not clicking for 15 seconds, the timer will fire.

Example

```
myTimer.reset();
```

URL Object

Description

The URL object encapsulates the state needed to manage a connection to a remote resource. URLs are never defined in the XML section of a Widget.

Method

`addPostFile()`	Add a file for a multipart POST request.
`cancel()`	Cancel an outstanding fetchAsync request.
`fetch()`	Retrieve the data at the specified URL as a string.
`fetchAsync()`	Retrieve the data at the specified URL asynchronously.
`getResponseHeaders()`	Retrieve the headers from an HTTP response.
`setRequestHeader()`	Set a header for an HTTP request.

Attributes

`autoRedirect`	A boolean indicating whether the URL object should follow redirects automatically (default is true).
`location`	A string representing the URL.
`outputFile`	If set, `URL.fetch()` will place retrieved data in a file with this name.
`postData`	If set, `URL.fetch()` will perform a POST to the specified location instead of the default GET using this string as the data to be posted.
`response`	The HTTP response code indicating the result of the most recent `URL.fetch()`.
`responseData`	The actual response data, regardless of response code.
`result`	The result of the most recent `URL.fetch()` or `fetchAsync()`.

Example

```
var url = new URL();
url.location = "http://www.yahoo.com";
contents = url.fetch();
```

URL.addPostFile()

Adds a file for a multipart POST request

Synopsis

```
URL.addPostFile(path)
```

Description

This function adds a file path to a list of files to be sent along with a POST request. The path is not tested for existence until the POST is actually sent. When files have been added, your request is automatically set to be a POST request.

Example

```
var myURL = new URL;
myURL.addPostFile( "myfile.png", "/A/Local/File/Path.png" );
myURL.location = "http://mysite.com";
myURL.fetch();
```

Availability

Available in version 2.1 or later.

URL.autoRedirect

indicates whether to automatically follow redirects

Synopsis

```
URL.autoRedirect
```

Description

This attribute allows you to control whether a URL object will follow redirects automatically. The default is true. Setting this to false will allow you to get the 302 redirect response and process it as you wish.

Example

```
var myURL = new URL;
myURL.autoRedirect = false;
myURL.location = "http://mysite.com";
myURL.fetch();
```

Availability

Available in version 2.1 or later.

URL.cancel()

Cancels an asynchronous request sent via fetchAsync()

Synopsis

```
URL.cancel()
```

Description

This function cancels an outstanding request sent via fetchAsync(). It has no effect if there is no async request pending. If called, the request is dropped and your function which would normally receive the result of the request is not called.

Example

```
myUrl.cancel();
```

Availability

Available in version 2.1 or later.

Example

```
var myURL = new URL;
myURL.location = "http://mysite.com";
myURL.fetchAsync( myCallback );
...
myURL.cancel();
```

Availability

Available in version 2.1 or later.

URL.clear()

clears the current settings of a URL object

Synopsis

URL.clear()

Description

After using a URL object, if you wish to reuse it to send another request, you can call the clear() method to ensure any prior post data (files, etc) are gone from the object. If you call this function on a URL object that is currently running an async request, the request is cancelled before the object is cleared.

Example

```
myURL = new URL;
myURL.location = "http://widgets.yahoo.com/"
result = myURL.fetch();
// reuse the object
myURL.clear();
myURL.location = "http://www.yahoo.com/"
result = myURL.fetch();
```

Availability

Available in version 2.1 or later.

URL.fetch()

return URL data as string

Synopsis

`URL.fetch(`*`[location]`*`)`

Description

Retrieves data from the remote location specified or from the web address specified in the URL's `location` attribute. If a *location* is specified this also sets the value of the `location` attribute of the URL. The data is either returned as a string (the default) or into a file if the `outputFile` attribute has been set. This is done synchronously so the Widget will pause until the data is retrieved.

If an error occurs and `fetch()` is returning a string, then it will return the string "`Could not load URL`" (or the string "`Could not load URL with POST`" if the attribute `postData` is set). The `response` attribute will contain the code indicating the type of error.

Note: if you are retrieving an RSS feed (or any web resource) you should make sure you do not fetch it too often. Any frequency shorter than 30 *minutes* should be very carefully considered. Your Widget may be used by thousands of people and the web site supplying the data may not appreciate the automated traffic. Also make sure that you do not implement a scheme that causes all instances of a Widget to try and fetch data at the same time (e.g. every hour on the hour) as this can also cause problems for sites (using an `onTimer` action is fine because different people's Widgets will be started at different times).

Example

```
var url = new URL();
webAddress = "http://www.yahoo.com";
contents = url.fetch(webAddress);
```

Notes

In version 2.1 or later, you can also get the result via the result attribute.

URL.fetchAsync()

GET or POST something asynchronously

Synopsis

`URL.fetchAsync(`*`function`*`)`

Description

This works similarly to fetch() except that it will perform the request asynchronously, leaving your Widget to go about its business while the request completes. When the request is finished, it calls the function you pass into the function. Your function receives the url object that started the request, which you can query to get the result and/or the response of the request.

Use of this function will greatly improve the responsiveness of your Widget, allowing the user to drag and otherwise interact with it while the request is running.

Example

```
var url = new URL();
url.location = "http://www.yahoo.com";
url.fetchAsync(url_done);

function url_done( url )
{
   print( "fetch complete" );
   print( "response: " + url.response );
   print( "result: " + url.result );
}
```

Availability

Available in version 2.1 or later.

URL.getResponseHeaders()

returns headers from an HTTP response

Synopsis

`URL.getResponseHeaders( name )`

Description

This function allows you to get at the headers that accompany an HTTP response. The

most useful purpose of this is to get "Set-Cookie" headers for use at a later time. Version 2.1 and later disables automatic cookie handling for security reasons, so if your Widget needs to use cookies to work, you will need to use this function to get them out of a response. You can then pass the cookies back to the server in a later call to fetch() by setting them with setRequestHeader.

This function returns an array of the headers that match the name you pass in. In version 3.0 or later, you can pass "*" as the name and you will receive an array of the complete headers, including the name (passing a name yields the value of the headers only).

Example

```
var url = new URL();
url.location = "http://www.my_site.com";|
url.fetch();
var cookies = URL.getResponseHeaders( "Set-Cookie" );
```

Availability

Available in version 2.1 or later.

URL.location

the web address of the URL

Synopsis

```
URL.location
```

Description

Specifies the web address the URL will fetch data from.

Example

```
var url = new URL();
url.location = "http://www.yahoo.com";
contents = url.fetch();
```

URL.outputFile

a file to store the fetched data in

Synopsis

```
URL.outputFile
```

Description

Specifies an optional file into which fetched data will be stored. If you are retrieving

textual data (e.g. a HTML file) it is usually easier to just use the return value of URL.fetch() but if you are retrieving binary data (e.g. an image file) then the retrieved data must be stored in a file as the process of converting it to a string will render it invalid.

Example

```
var url = new URL();
url.outputFile = system.widgetDataFolder + "/mytempfile";
url.location = "http://www.example.com/graphic.jpg";
url.fetch();
myIng.src = url.outputFile;
```

URL.postData

data to be POSTed to a web server

Synopsis

`URL.postData`

Description

Setting `postData` will cause a URL object to POST to its location rather than performing a GET operation. To post nothing, set `postData` to an empty string. To make the URL object GET again, set `postData` to `null`.

The format of this data should be url encoded, i.e. each parameter is passed as name=value and parameters are separated by a '&' symbol. Use encode() when your data contains spaces, '&', '=' or non-ASCII characters.

Example

```
var url = new URL();
var text = encode( "a lot of &&& bad text" );
url.postData = "x=123&y=456&q=" + text;
contents =
     url.fetch("http://www.example.com/myscript.php");
```

URL.response

the HTTP response code for the last fetch

Synopsis

`URL.response`

Description

The `response` attribute indicates the HTTP response code received as a result of the last `fetch` call. Codes greater than or equal to 400 indicate there was a problem completing the request.

Note that a response code is only available if a web server was actually contacted and a request made. If the server is not available or an invalid URL is supplied for the `location` then the `response` attribute will be `0` (zero). A successful web page retrieval is usually indicated by a response code of `200`. By default, the Widget Engine does redirection automatically, so you will never see a response code of 302 unless you set the autoRedirect property to false.

Example

```
var url = new URL();
url.location = "http://www.yahoo.com";
contents = url.fetch();
log("Response was: " + url.response);
```

URL.responseData

the result of the last request

Synopsis

URL.responseData

Description

The responseData attribute is used to get the actual text response from the server regardless of the status code sent back. This differs from the URL.result attribute in that you will always get the real response text back and never any status string. If the connection failed, this attribute is empty.

With this attribute, you can get the actual 404 page that is returned if you get a 404 error from the server.

Synopsis

```
var url = new URL();
url.location = "http://www.mysite.com/a_url_that_doesnt_exist";
url.fetch();

// will print "Could not load URL"
print( url.result );

// will print the actual response from the server
print( url.responseData );
```

Availability

Available in version 2.1.1 or later.

URL.result

the result of the last request, or an error string

Synopsis

```
URL.result
```

Description

The `result` attribute indicates the result received from the last request made via fetch() or fetchAsync(). This will contain the actual text of the result (e.g. a web page), or the error strings "Could not load URL" or "Could not load URL with POST". If you need the actual response even when the status code from the server is not 200, use responseData in version 2.1.1 or later.

Example

```
var url = new URL();
url.location = "http://www.yahoo.com";
url.fetch();
print( url.result );
```

Availability

Available in version 2.1 or later.

URL.setRequestHeader()

sets a header on an HTTP request

Synopsis

```
URL.setRequestHeader( name, value )
```

Description

This function is used to set headers to accompany an HTTP request. The most common use of this is to set cookies for a request. Version 2.1 disables automatic cookie support, so this function is necessary in order to continue to use cookies. With the getResponseHeaders function, this function can be used to deal with cookies in your Widget. After receiving cookies in a prior response (see getResponseHeaders), you can use this function to set the cookie or cookies in a future request. The name parameter is the name of the header, the value is the actual contents of the header.

Example

```
var url = new URL();
url.location = "http://www.my_site.com";
url.setRequestHeader( "Cookie", myCookie );
url.fetch();
```

Available

Available in version 2.1 or later.

Window Properties and Functions

Window.focus()

brings a window to the front

Synopsis

```
Window.focus()
```

Description

If for some reason you need to bring a window foward, you can use the focus() method on a window. On Windows, this API might not always bring the window completely forward, particularly if the Widget is not active to begin with. But if you are interacting with the Widget and require an inspector or other secondary window to come forward, this API will do just that.

Example

```
myWindow.focus();
```

Availability

Available in version 3.0 or later.

Window.locked

disables the ability for a user to drag a window

Synopsis

```
Window.locked
```

Description

You can set or inspect a window's locked property to control whether the user can drag a window or not. This setting is largely controlled by a Widget's Window preferences in the preferences dialog. This property is not expressed in the XML interface.

Example

```
myWindow.locked = true;
```

Window.moveTo()

move a window around the screen

Synopsis

```
Window.moveTo(newX, newY, duration)
```

Description

The window's origin (`hOffset`, `vOffset`) is moved to the new coordinates specified by `newX` and `newY`. `duration` specifies the time (in tenths of a second) you want the animation to last for.. The move of the object is animated (which is what makes this different from just changing `hOffset` and `vOffset`).

Example

```
myWindow.moveTo(150, 150, 2);
```

Window.recalcShadow()

recalculate the Widget's Aqua shadow

Synopsis

```
Window.recalcShadow()
```

Description

If a Widget that has an Aqua shadow changes its shape (for example, by hiding or showing images) it should call the `recalcShadow()` method of its main window before returning control to the user so that the shadow is correctly displayed.

Note: this has no effect on Windows.

Example

```
if (myWindow.shadow)
  myWindow.recalcShadow();
```

Windows Note

Window shadows are not currently supported on Windows. If you decide to render your artwork with your own shadows, be sure to set the shadow property of your window to 0. If not, when and if Windows does support shadows you'll end up with a double-shadow.

Window.root

the root view of the window

Synopsis

```
Root Window.root (read-only)
```

Description

This property contains the root view of the window. This value will never be null and contains all the views at the top level of the window. To add a view to a window, you can either call `root.addSubView()` or set an object's `window` property. In both cases the view will be a child view of the root.

Example

```
var root = myWindow.root;
```

Availability

Available in version 3.0 or later. The Widget's minimum version must be set to at least version 3.0 for this property to exist.

Animation Objects

Objects and functions to aid in doing animations

Version 2.1 and later of the Widget Engine contains animation support that allows you to do animations asynchronously as well as synchronously. It also allows you to do custom animations written in JavaScript. You can fade, move, or rotate objects all at the same time.

A new object called animator controls the animation. You tell tell the animator to start an animation and run it asynchronously, allowing your Widget to do other things in the meantime. You can also take an animation (or multiple animations) and run them all synchronously, meaning the call will block until all the animations are complete.

These facilities take all of the hard work out of doing pretty interesting animations. They also provide 'ease' functions for you to use to get the standard animation technique of easing, where an object's speed can ramp up or down at the start or end of the animation to give a better feeling of realism to the movement.

Each animation type has a 'done' function that can be called to let you know when the animation is complete. This function is only called when running an animation asynchronously. You can use this done function to chain animations together, starting a new animation when an older one is ending.

animator

the master animation object

The animator object is the core of the animation system in version 2.1 or later. It is what you use to start animations. You can also call methods on it to help you deal with 'ease' transitions.

animator.ease()

blend a number between two numbers for an 'ease' effect

Synopsis

```
animator.ease( start, end, percent, easeType )
```

Description

This function is used to help you create an 'ease' effect in your animations. All of the built-in move animations that have been in Widget Engine 2.0 and later have had this effect. Essentially you can make an object speed up as it moves away or slow down as it stops to give it a more realistic feeling of movement.

To use this function, you pass the starting and ending number, along with the percentage complete as a fraction (i.e. if you are half complete, pass 0.5). The ease type is

specified with one of the constants attached to the animator object: kEaseNone, kEaseIn, kEaseOut, kEaseInOut. See the explanation of those constants for what they mean.

Example

```
var n = animator.ease( 0. 100, .7, animator.kEaseOut );

// at this point, n is some place between 0 and 100
// depending on the ease out curve. It is not linear.
```

Availability

Available in version 2.1 or later.

animator.kEaseIn
animator.kEaseOut
animator.kEaseInOut
animator.kEaseNone

constants to dictate the type of easing to use

Description

These constants are used when creating different animation objects as well as using the animator.ease() function. If you are familiar with easing, the engine currently uses a sinusoidal ease function.

Ease In means that the object will start to move slowly and then speed up as it moves.

Ease Out means the object will start quickly and slow down as it comes to rest.

Ease In/Out means the object will start slowly, reach full speed, then start to slow down as it approaches the end of its journey.

Ease None means no easing is in effect. The speed is constant from beginning to end.

Availability

Available in version 2.1 or later.

animator.milliseconds

the current animation timebase

Description

For custom animations, it is useful to get the current animation timebase to mark the start time (or just know when 'now' is). This property of the animator object allows you to determine the current time.

Example

```
myAnimation.startTime = animator.milliseconds;
```

Availability

Available in version 2.1 or later.

animator.runUntilDone()

runs an animation or animations to completion

Synopsis

```
animator.runUntilDone( object | array )
```

Description

This function is used to run an animation or animations until they are all complete. This function will not return until all of the animations specified are considering to be done. For this reason, this function should be used only when you are running short, finite animations. An infinite animation such as a 'pulsing button' effect would mean this call would never exit, so care must be taken to ensure this does not occur.

Example

```
// crossfade
var a = new FadeAnimation( myImage1, 0, 350,
                           animator.kEaseOut );
var b = new FadeAnimation( myImage2, 255, 350,
                           animator.kEaseOut );
animator.runUntilDone( new Array( a, b ) );
// at this point both animations are complete.
```

Availability

Available in version 2.1 or later.

animator.start()

starts an asynchronous animation or animations

Synopsis

```
animator.start( object | array )
```

Description

This function is used to run an animation or animations asynchronously. This function returns immediately — it does not wait for the animations to complete. The animations will not actually even start until your JavaScript code is exited and control returns back

to the Widget's main event loop. This means you can start multiple animations and they will actually start at the exact same time. You can call start for each one, or pass them all as an array into start, it doesn't matter.

Example

```
// crossfade asynchronously
var a = new FadeAnimation( myImage1, 0, 350,
                           animator.kEaseOut );
var b = new FadeAnimation( myImage2, 255, 350,
                           animator.kEaseOut );
animator.start( new Array( a, b ) );
// at this point nothing has started yet. When we leave our
Javascript code, the animations will start up at the exact same
time.
```

Availability

Available in version 2.1 or later.

animation.kill()

base class method to terminate a running animation

Synopsis

animation.kill()

Description

This function is essentially a 'base class' function for all the following animation objects. That is, it can be called on any of the animation objects below. It is used for stopping asynchronous animations that might be running. For example, if you have an animation that rotates an object indefinitely while in a certain mode, you will need to stop that animation when you exit the mode. To do that just use this function.

Example

```
var a = new CustomAnimation( 1, SpinMeRightRoundBaby );
animator.start( a );

// some time later, maybe after the user clicks a button
if ( a != undefined )
  a.kill();
```

Availability

Available in version 2.1 or later.

CustomAnimation()

a custom animation routine written in JavaScript

Synopsis

```
new CustomAnimation( interval, updateFunc [,doneFunc] );
```

Description

This is the most flexible animation object available to you, but you do need to do all the work. In general, you can do fairly interesting things by merely using combinations of the fade, move, and rotate animation objects provided below.

The first parameter is the interval your animation should start running at, in milliseconds. You can change this interval in your update function. This allows you to have an animation that changes speed, etc. A good example of this is something along the lines of an animated GIF, in that each frame can have its own duration. When your update function is called the 'this' object is the animation itself, so you can alter the interval as such:

```
function MyUpdate()
{
  this.interval = 5000; // switch to 5 seconds
  return true;
}
```

The next parameter is your update function. This is where you do the work of the animation. You might move an object, change it's opacity, or do truly interesting things like adjust an image's HSL settings. A custom animation runs until your update Function returns false. So a perpetual animation would always return true, as did the code snippet above. You could always kill an animation that was perpetual by calling the kill() method on the animation:

```
myAnimation.kill();
```

The last, optional parameter is the done function. This is called when your animation is done. If you have a finite animation, it will be called right after your update function returns false. Alternatively, if you wish you might just do the work in your update function right before you return false.

Along with the interval, your custom animation has another property accessible to it, startTime. This is set automatically when your animation is added to the queue (in the case of using start) or when runUntilDone is called. You can query this value inside your update function to determine how much time has elapsed. The example below shows this in use.

Example

```
var x = new CustomAnimation( 1, UpdateMe );
// some custom properties for my animation
x.duration = 350;
x.startOpacity = myObject.opacity;
x.endOpacity = 0;

function UpdateMe()
{
  var now = animator.milliseconds;
  var t = limit( now - this.startTime, 0,
                 this.duration );
  var percent = t / this.duration;

  // set the new opacity of our object based on
  // easing.
  myObject.opacity = animator.ease( this.startOpacity,
                 this.endOpacity, percent,
                 animator.kEaseOut );

  // If the duration is up, let's get out of here
  if ( animator.milliseconds >=
        (this.startTime + this.duration )
  {
        // make sure we reached the end
        myObject.opacity = this.endOpacity;
        return false; // we're done
  }
  return true; // keep going
}
```

Availability

Available in version 2.1 or later.

FadeAnimation()

an animation object to adjust the opacity of an object

Synopsis

```
new FadeAnimation( object, toOpacity, duration,
                   easeType [, doneFunc]);
```

Description

This animation object can be used to adjust the opacity of an image, frame, text, textarea, or window object. This can be used to fade an object in or out. You pass the opacity you ultimately want to reach in the toOpacity parameter. The duration is specified in milliseconds. You can specify the type of easing in the easeType parameter.

Once you've created this animation object, you can pass it to animator.start() or animator.runUntilDone().

If you pass a function for the doneFunc parameter, and you started your animation with animator.start(), when the animation is complete, the function you passed will be called.

Example

```
var a = new FadeAnimation( myObject, 0, 350,
             animator.kEaseOut, FadeDone );
animator.start( a );

function FadeDone()
{
  // the fade above has finished
  print( "fade complete" );
}
```

Availability

Available in verison 2.1 or later.

MoveAnimation()

an animation object to adjust the position of an object

Synopsis

```
new MoveAnimation( object, toX, toY, duration,
                   easeType [, doneFunc]);
```

Description

This animation object can be used to adjust the position of an image, frame, text, textarea, or window object. This can be used to move an object on screen. It works by adjusting the hOffset and vOffset attributes of the object you pas in. You pass the hOffset and vOffset you ultimately want the object to be. The duration is specified in milliseconds. You can specify the type of easing in the easeType parameter.

Once you've created this animation object, you can pass it to animator.start() or animator.runUntilDone().

If you pass a function for the doneFunc parameter, and you started your animation with animator.start(), when the animation is complete, the function you passed will be called.

Example

```
var a = new MoveAnimation( myObject, 100, 100, 350,
             animator.kEaseOut, MoveDone );
animator.start( a );

function MoveDone()
{
  // the move above has finished
  print( "move complete" );
}
```

Availability

Available in verison 2.1 or later.

RotateAnimation()

an animation object to adjust the rotation of an object

Synopsis

```
new RotateAnimation( image, toAngle, duration,
                     easeType [, doneFunc]);
```

Description

This animation object can be used to adjust the rotation of an image object. It does not affect text, textarea, or window objects. It works by adjusting the rotation attribute of the image you pass in. You pass the angle you ultimately want the object to be when the animation is finished. The duration is specified in milliseconds. You can specify the type of easing in the easeType parameter.

Once you've created this animation object, you can pass it to animator.start() or animator.runUntilDone().

If you pass a function for the doneFunc parameter, and you started your animation with animator.start(), when the animation is complete, the function you passed will be called.

Example

```
var a = new RotateAnimation( myObject, 180, 350,
             animator.kEaseOut, RotateDone );
animator.start( a );

function RotateDone()
{
  // the rotate above has finished
  print( "rotate complete" );
}
```

Availability

Available in verison 2.1 or later

XML Services

About XML Services

The Widget Engine provides several mechanisms for dealing with XML. With these services, you can create, parse, and manipulate XML trees. You can also use the built-in implementation of XMLHttpRequest, a pseudo-standard for fetching XML off of web servers.

The parsing and creation of XML documents is done via the global XML object. From there, you can use standard W3C Level 1 DOM APIs to manipulate the XML tree. To make it even easier to extract data from a tree, we provide an XPath 1.0 implementation via the node.evaluate() addition.

DOM API

This section lists the various objects and methods/properties currently supported by the Widget Engine's Level 1 W3C DOM implementation. We currently provide a large subset of the full API. The current parser does not yet deal with DTDs, so it will not do things such as fill in attributes with default values automatically and the like.

The following is a brief overview of the properties and functions we support. For more information, we suggest you visit the w3c.org website.

DOMException

The standard exception class for the DOM.

When an exceptional situation arises, a DOMException is thrown as a Javascript exception. You can inspect the object's `code` attribute to see what happened. Level 1 exception codes are:

```
INDEX_SIZE_ERR                1
DOMSTRING_SIZE_ERR            2
```

```
HIERARCHY_REQUEST_ERR          3
WRONG_DOCUMENT_ERR             4
INVALID_CHARACTER_ERR          5
NO_DATA_ALLOWED_ERR            6
NO_MODIFICATION_ALLOWED_ERR    7
NOT_FOUND_ERR                  8
NOT_SUPPORTED_ERR              9
IN_USE_ATTRIBUTE_ERR           10
```

DOMDocument

represents an entire XML document

Properties

`doctype`	The document type definition for the document.
`documentElement`	The root element of the document.

Functions

```
DOMElement createElement(string tagName);
```

Creates a new element node for the document with the given tag name. You must attach it to the document as appropriate using appendChild.

```
DOMText createTextNode(string data);
```

Creates a new text node for the document with the given content.

```
DOMComment createComment(string data);
```

Creates a new comment node with the given content.

```
DOMCDATASection createCDATASection(string data);
```

Creates a new CDATA section with the given data.

```
DOMProcessingInstruction
    createProcessingInstruction(string target, string data);
```

Creates a new processing instruction with the given target and data.

```
DOMAttribute createAttribute(string name);
```

Creates a new attribute node with the given name.

```
DOMNodeList getElementsByName(string name);
```

Returns a list of all elements in the document with the specified name.

DOMNode

the base class for items in an XML tree

DOMNode is the base class for pretty much everything you'll deal with in the DOM API. You'll never encounter a DOMNode in everyday life, but its interface is something that is common to all node types (text, element, CDATA, etc.) and as such is documented once here rather than over and over for each subclass

Properties

`nodeName`

The name of this node.

`nodeType`

The node type, expressed as an integer.

`parentNode`

The parent of this node (can be null).

`childNodes`

A DOMNodeList of children.

`firstChild`

The first child node of this node.

`lastChild`

The last child node of this node.

`previousSibling`

The previous sibling node of the current node.

`nextSibling`

The next sibling node of the current node.

`attributes`

A DOMNamedNodeMap of this nodes attributes (only valid for Element nodes, null otherwise).

`ownerDocument`

The DOMDocument that this node belongs to.

Functions

```
DOMNode insertBefore(DOMNode newChild, DOMNode refChild);
```

Inserts newChild before refChild in this node's children.

```
DOMNode replaceChild(DOMNode newChild, DOMNode oldChild);
```

Replaces oldChild with newChild.

```
DOMNode removeChild(DOMNode oldChild);
```

Removes oldChild from this node's children and returns it.

```
DOMNode appendChild(DOMNode newChild);
```

Adds the given child node (if this node type allows children).

```
boolean hasChildNodes();
```

Returns true if this node has child nodes.

```
DOMNode cloneNode(boolean deep);
```

Clones this node. If deep is true, clones all descendents as well.

```
<various> evaluate(string xpath-expression);
```

This is an extension defined by the Widget Engine which lets you interface with the engine's XPath support. Using the current node as the context for the XPath expression, you can execute almost any XPath 1.0 expression you can dream up (except for some namespace-specific functions). The result could be a string, number, or a set of nodes. The Widget Engine returns node sets as DOMNodeLists. See the section on XPath for more information.

```
string toXML();
```

Widget Engine DOMNode extension. Converts the subtree starting at this node into XML for output for either writing to a file, or possibly for debugging purposes.

DOMNodeList

a simple list of nodes

In keeping with W3C ways, any list of nodes as expressed through the DOM API is represented as a DOMNodeList, not as a Javascript array.

Properties

```
length
```

The number of items in the list.

Functions

```
DOMNode item(n)
```

Returns the nth item in the list. DOMNodeLists are zero-based.

DOMNamedNodeMap

a map of nodes which is accessible by name or index

When attribute nodes are returned via the attributes property of the DOMElement node, they are returned in a named node map. This map is primarily accessible by name, but you can also traverse it via index like a DOMNodeList. The order of the attributes is not guaranteed and should never be relied upon.

Properties

`length`

The number of items in the list.

Functions

```
DOMNode getNamedItem(string name);
```

Returns the item with the given name, or null if the item is not found.

```
DOMNode setNamedItem(string node);
```

Adds the given node to the map. If a node with the given name exists, it is replaced and the old node is returned. If a node with the give name does not exist, null is returned.

```
DOMNode removeNamedItem(string name);
```

Removes the item with the given name, if it exists.

```
DOMNode item(int n);
```

Returns the nth item in the list. DOMNodeLists are zero-based.

DOMCharacterData

base class for text and comment nodes

This class, like DOMNode, is something that you'll never encounter in real life, but its interface is available for both DOMText and DOMComment nodes. As with DOMNode, the interface is shown here once and not duplicated in both of those classes.

Properties

`data`

The actual character data.

`length`

The length of the character data.

Functions

`string substringData(int offset, int count);`

Returns a substring of the data as a string. It returns `count` characters of the data starting at `offset`.

`void appendData(string data);`

Appends the given text to the node's data.

`void insertData(int offset, string data);`

Inserts the given string at the specified offset.

`void deleteData(int offset, int count);`

Erases `count` characters of data starting at `offset`.

`void replaceData(int offset, int count, string data);`

Replaces the sequence of `count` characters starting at `offset` with `string`.

DOMAttribute

an attribute node for an element

Properties

`name`

The name of the attribute.

`value`

The value of the attribute. Character and entity references are resolved before returning this value.

DOMElement

an element node

Properties

`tagName`

The tag name of the element.

Functions

`string getAttribute(string name);`

Returns the value of the attribute specified, or an empty string if that attribute does not exist.

```
setAttribute(string name, string value);
```

Adds the given attribute and its value to the element, replacing any attribute of the same name that might already exist.

```
removeAttribute(string name);
```

Removes the attribute with the specified name, if present.

```
DOMAttribute getAttributeNode(string name);
```

Returns the attribute node corresponding to the name passed in, or null if the attribute does not exist.

```
DOMAttribute setAttributeNode(DOMAttributes attr);
```

Adds the given attribute to the element, replacing any attribute that might exist with the same name. If the node replaces an existing node, the old node is returned as the result, else null is returned.

```
DOMAttribute removeAttributeNode(DOMAttribute attr);
```

Removes the node specified from the element's attributes and returns it.

```
DOMNodeList getElementsByTagName(string name);
```

Returns a list of all elements with the specified tag name that are a descendant of this node.

```
void normalize();
```

If there are contiguous DOMText nodes in the subtree starting with the current element, this function combines them into a single element.

DOMText

a text element

Functions

```
DOMText splitText(int offset);
```

Splits the given node into two and adds the new node as its new sibling following it in the tree. This node will contain the text up until `offset`. The following node will contain the remainder of the text. The new text node is returned.

DOMComment

a comment node

This node merely has the properties and functions of the DOMCharacterData interface.

DOMCDATASection

a CDATA section

This node merely has the properties and functions of the DOMCharacterData interface.

DOMDocumentType

the document type node

Currently, this node only defines the name property. Entities and notations are not supported by the current version of the Widget Engine.

Properties

`name`

> The name of the document's root object. For a Widget, this would be 'widget'. For HTML it would be 'html'.

DOMNotation

a notation node

Currently unsupported.

DOMEntity

a node representing an entity

Currently unsupported.

DOMEntityReference

a node representing an entity reference

Currently unsupported.

DOMProcessingInstruction

a node representing a processing instruction

Properties

`target`

> The target of the processing instruction.

`data`

The content of the processing instruction. This is from the first non-whitespace character after the target to the character immediately preceding the "?>".

XMLDOM Object

The XMLDOM global object allows you to parse and create XML documents. Note that as per the Level 1 DOM API you cannot create DOMNode entities via new. You must use the XMLDOM object to create a document and the document itself to create elements to attach to the document (i.e. the DOMDocument is the factory for all elements, text items, comments, etc.).

XMLDOM.createDocument()

creates a new, empty DOMDocument

Synopsis

```
doc = XMLDOM.createDocument();
```

Description

This function allows you to create a new DOMDocument element. From there you can use the DOMDocument API to create elements to add to the document, as specified in the W3C Level 1 DOM specification.

Example

```
doc = XMLDOM.createDocument();
root = doc.createElement( "root" );
doc.appendChild( root );

print( doc.toXML() );
```

Availability

Available in version 3.0 or later.

XMLDOM.parse()

parses XML and yields a DOMDocument

Synopsis

```
doc = XMLDOM.parse(xml);
```

Description

The parse function parses the given XML string (gotten either from a web server or using a call such as filesystem.readFile()) and returns a DOMDocument node. The document node is a W3C Level 1 DOMDocument and conforms to the API as specified by the W3C (modulo some omissions such as entity objects).

If the xml fails to parse, an exception is thrown containing the error string. You should always call XML.parse inside a try/catch block to deal with failures.

Example

```
try
{
  doc = XMLDOM.parse( xmlStream );
  root = doc.documentElement;
  ...
}
catch( e )
{
  print( e );
}
```

Availability

Available in version 3.0 or later.

XMLHttpRequest

Description

The XMLHttpRequest object is very much like the URL object that has existed in the Widget Engine since very early on. XMLHttpRequest is, however, a de-facto standard for doing XML over http in web browsers, so its addition here is to provide people with an easier migration path when moving AJAX code over to the Widget Engine, as well as simply trying to adhere to standards so developers find it more approachable.

Method

`abort()`	Cancel an outstanding async request.
`getAllResponseHeaders()`	Returns all response headers.
`getResponseHeader()`	Returns a specific response header.
`open()`	Sets up our request parameters.
`send()`	Sends the request with optional data.
`setRequestHeader()`	Set a header for a request.

Attributes

`onreadystatechange`

Specifes a function to be called when sending an async request as the state of the request changes.

`readyState`	The current state of the request, used inside of the onreadystatechange function.
`responseText`	The full text of the response (e.g. a web page or XML text).
`responseXML`	If the response is text/xml, this property will contain the DOMDocument which represents the XML that was received.
`status`	The HTTP status code (e.g. 200).
`statusText`	The HTTP status text (e.g. "OK").

Example

```
var req = new XMLHttpRequest();
req.open( "GET", "http://www.yahoo.com", false );
req.send();
print( req.responseText );
```

XMLHttpRequest.abort()

aborts an async request

Synopsis

```
XMLHttpRequest.abort()
```

Description

If true was passed for the async parameter of open(), this call can be used to terminate the request if it is still outstanding.

Example

```
request.abort();
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.getAllResponseHeaders()

returns all the headers from a response

Synopsis

```
array XMLHttpRequest.getAllResponseHeaders()
```

Description

After a request is complete, this call can be used to retrieve all the headers returned with the response as an array of strings.

Example

```
var headers = request.getAllResponseHeaders();
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.getResponseHeader()

returns one or more headers from a response by name

Synopsis

```
string|array XMLHttpRequest.getReponseHeader(string)
```

Description

After a request is complete, this call can be used to retrieve one or more headers with the given name. If there is only one header, it will return a single string result. If there are multiple, it will return an array of matches.

Example

```
var cookies = request.getResponseHeader( "Set-Cookie" );
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.onreadystatechange

function to call as an async request is processed

Synopsis

`XMLHttpRequest.onreadystatechange`

Description

If a request is send asynchronously (see `open()`), you must specify a function to be called as the status of the request changes. No parameters are passed to this function. When your function is called, 'this' refers to the request. Generally, you'll only care when the readyState of your request is the value 4 (complete).

Example

```
var request = new XMLHttpRequest();
request.onreadystatechange = myStatusProc;
request.open( "GET", "http://www.yahoo.com", true );
request.send();

// someplace else
function myStatusProc()
{
  if ( this.readyState == 4 ) // complete
  {
     print( this.status );
  }
}
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.open()

sets up a request for sending

Synopsis

```
XMLHttpRequest.open(method,url,async);
```

Description

This sets up a request for sending. You pass the method, the url, and a flag indicating whether you wish to send this request asynchronously. Please note that at present we do not support the traditional username and password parameters. It may be supported in a later release.

Valid values for the method parameter are "GET", "POST", "HEAD", "OPTIONS", "PUT", and "DELETE".

Example

```
var request = new XMLHttpRequest();
request.onreadystatechange = myStatusProc;
request.open( "GET", "http://www.yahoo.com", true );
request.send();
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.readyState

the current state of the request

Synopsis

```
XMLHttpRequest.readyState (read-only)
```

Description

This is used to determine what the current state of the request is. This is typically only used when sending an asynchronous request in your onreadystatechange function.

The values for readyState are:

```
0    uninitialized
1    loading
2    loaded
3    interactive
4    complete
```

The Widget Engine will only set the readyState to 0, 1, or 4 in version 3.0.

Example

```
var request = new XMLHttpRequest();
request.onreadystatechange = myStatusProc;
request.open( "GET", "http://www.yahoo.com", true );
request.send();

// someplace else
function myStatusProc()
{
  if ( this.readyState == 4 ) // complete
  {
     print( this.status );
  }
}
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.responseText

the text returned by the request

Synopsis

```
XMLHttpRequest.responseText (read-only)
```

Description

This property contains the text returned by the web server for the request you sent. Typically this would be a web page or XML.

Example

```
var request = new XMLHttpRequest();
request.open( "GET", "http://www.yahoo.com", false );
request.send();
if ( request.status == 200 )
   print( request.responseText );
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.responseXML

the XML DOM returned by the request

Synopsis

```
XMLHttpRequest.responseXML (read-only)
```

Description

If the response to the request returned data with a content type of "text/xml", this property will contain the DOMDocument node representing the XML document (i.e. it will be automatically parsed and ready for use). If the document cannot be parsed, or the content type is not "text/xml", this property will be set to null.

Example

```
var request = new XMLHttpRequest();
request.open( "GET", "http://www.yahoo.com", false );
request.send();
if ( request.status == 200 )
   print( request.responseXML.toXML() );
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.send()

sends the request to the server

Synopsis

```
XMLHttpRequest.send([body])
```

Description

This function actually does the sending of the data to the server. You can optionally pass data to be passed as the body of the HTTP request into this function.

Example

```
var request = new XMLHttpRequest();
request.open( "POST", "http://www.yahoo.com", false );
request.send( someXML );
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.setRequestHeader()

sets a request header

Synopsis

```
XMLHttpRequest.setRequestHeader(name,value)
```

Description

This function adds a header to a request, potentially replacing any existing header with the same name.

Example

```
var request = new XMLHttpRequest();
request.open( "POST", "http://www.yahoo.com", false );
request.setRequestHeader( "Content-type", "text/xml" );
request.send( xml );
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.status

returns the status of the response

Synopsis

```
XMLHttpRequest.status (read-only)
```

Description

This property represents the HTTP status code returned by the server, e.g. 200, 404, etc.

Example

```
var request = new XMLHttpRequest();
request.open( "POST", "http://www.yahoo.com", false );
request.setRequestHeader( "Content-type", "text/xml" );
request.send( xml );
if ( request.status == 200 ) // success!
  DoSomethingWonderful();
```

Availability

Available in version 3.0 or later.

XMLHttpRequest.statusText

returns the status text of the response

Synopsis

```
XMLHttpRequest.statusText (read-only)
```

Description

This property represents the HTTP status text returned by the server, e.g. "OK", "Not Found", etc. These exactly correspond to the codes returned via status. Normally, you'd use status and not statusText.

Example

```
var request = new XMLHttpRequest();
request.open( "POST", "http://www.yahoo.com", false );
request.setRequestHeader( "Content-type", "text/xml" );
request.send( xml );
if ( request.statusText == "OK" ) // success!
  DoSomethingWonderful();
```

Availability

Available in version 3.0 or later.

XPath Support

brief overview of XPath

Starting in version 3.0, the Widget Engine now supports the XPath 1.0 language for extracting nodes and node information from an XML tree. Chances are you'll almost never use the raw DOM API to extract node information and will instead use XPath. It's far easier and more straightforward. This section explains how it is integrated into the Widget Engine and how you access it. It also demonstrates some examples of how it can be used.

This is only a brief explanation. For full documentation, please consult the w3c.org website on XPath 1.0.

First, let's define an example XML tree:

```
<?xml version="1.0" encoding="utf-8"?>
<my-data>
   <element1 name="fred" size="200">
      This is some text
   </element1>
   <image-list>
       <image size="32"
           src="http://www.yahoo.com/image.png"/>
       <image size="48"
           src="http://www.yahoo.com/image2.png"/>
   </image-list>
</my-data>
```

Now let's try to do some specific things. Normally, XPath will return a list of nodes. The Widget Engine returns those nodes as a DOMNodeList. XPath also starts at a 'context node', i.e. where the XPath search should be relative to. You can also specify a search to start at the top by starting the path with a '/'. Let's say you have the document node in a variable called doc:

```
element = doc.evaluate( "my-data/element1" );
```

The above example fetches all nodes that match the path "my-data/element1" (in this case, one node) and returns it as a node list.

```
images = doc.evaluate( "my-data/image-list/image" );
```

The above statement returns all the nodes that match the given path. This time, we'll get a node list back with two elements (the two child nodes of image-list).

This is XPath at its simplest — just selecting nodes out of a document. Now we'll get a little fancier. Let's say we want to select the image that has a size of 48. We can use a predicate for this. A predicate is a condition applied to the search. It essentially filters the results to those that match the condition.

```
image = doc.evaluate( "my-data/image-list/image[@size='48']" );
```

This says "find me all items that match this path, but only the ones whose 'size' attribute has the value '48'". The @ symbol is a shorthand way of specifying that you are looking for an attribute. The longhand for the predicate would be [attribute::name='48'].

Now, if you wanted to get the src attribute for that image, you'd do this:

```
src = image.item(0).getAttribute( src );
```

But there's an easier way as we'll see in a second. First, let's extract some text. Lets say we want the text inside element1. Here's one way to do this:

```
element = doc.evaluate( "my-data/element1" );
text = element.item(0).firstChild.data;
```

Essentially, we need to know that element is really a node list, so we extract item 0 and then ask it for its first child (since the text is really a subnode of element1 as far as the XML tree goes). Then we get it's data. Well, that's cute, but it's somewhat complicated. Fortunately, XPath has functions you can call to make life easier. We'll use the `string()` function:

```
text = doc.evaluate( "string(my-data/element1)" );
```

That's it. The `string()` function takes the result of the expression passed to it and turns it into a string. For element nodes, it takes all the text subelements under it and concatenates them and returns it. In our case we only had one element and one text node, so we got the exact result we wanted. For attribute nodes, `string()` returns the value of the attribute. Now we can revisit our attempt above to get the src attribute of the image with size 48:

```
src = doc.evaluate( "string(my-data/image-list/image[@size='48']/
attribute::src)" );
```

This time, we used the same basic path as before, but this time we added another path segment to extract the src attribute from the result and then the string function returned the value of that attribute.

There are various things you can search for in the XML using xpath, and many many ways to search. You can find elements with certain parents, you can fetch elements who have a particular subelement, etc. Consult the full XPath 1.0 specification on the w3c.org website for more information.

Windows and Mac OS X Differences

This section's purpose in life is to point out those things that are different between the Mac and PC versions of the Widget Engine in one place.

Unix Commands

First, since the PC is not based on UNIX, as is Mac OS X, there is no guarantee that a Widget that used runCommand successfully on the Mac will meet with the same success on Windows. But in an effort to make as many Widgets as possible work cross-platform, there are a number of commands which are packaged with Yahoo! Widget Engine for Windows:

basename	bc	bunzip2	bzip2	bzip2recover
cal	cat	cksum	cmp	Comm.
compress	cp	curl	cut	Date
dc	dd	df	diff3	Diff
dirname	du	echo	egrep	Env
expand	expr	fgrep	find	Fmt
fold	fsplit	gawk	grep	Gunzip
gzip	head	id	join	Less
lesskey	ln	logname	ls	m4
md5sum	mkdir	mv	od	Open
paste	patch	pr	printenv	Pwd
rm	rmdir	sdiff	sed	Shar
sleep	sort	split	sum	Sync
tail	tar	tee	touch	Tr
uname	unexpand	uniq	unzip	Uudecode
uuencode	wc	which	whoami	Xargs
yes	zcat	zip		

In addition "sh" is also available to run shell scripts.

Command Key

There's no Command key on Windows. When you would think to use it, use Control. So to drag a Widget, use Control-drag for example. This also affects HotKeys, as described

below.

Key Names

When the Delete key is pressed on a Windows keyboard, you will receive "ForwardDelete". If you press Backspace, you will receive "Delete". This is because the naming comes from the original Mac key naming, and we cannot change it. Return and Enter are a bit similar. On most (if not all) Windows keyboard, There is no Return key, it's always Enter. There are two distinct keys on Mac keyboards. We currently return "Return" for the enter key on Windows. Again, it's not something we can change, so please be aware of it. In most cases though, it might be best to always look for "Return" and "Enter".

HotKeys

If you install a hot key that was cmd-control-<key> it will just be control-<key> on Windows (since there's no Command key).

When registering a hot key on Windows, the key is exclusive, which is different from Mac OS. So multiple Widgets cannot register for the same hot key. The second one to try is denied their happy fun key. Unfortunately, we don't have any way for the Widget to know this at present.

F1 typically shouldn't be used, as it is the Help key. F12 is reserved on 2000/XP and latest NT variants for the debugger.

There is no notification on release of the key. Only press, so anyone's onKeyUp handler will never get fired on windows.

Certain hot key sequences are illegal, such as alt-tab and ctrl-alt-delete.

Paths

The native path system of Mac OS X is UNIX-style, or forward slashed paths. Windows has a drive letter and backslashes. The Widget Engine considers its native path style to be forward-slashed paths. This is because of its Mac heritage as well as the JavaScript heritage as well, which is also forward-slashed.

If you use runCommand to call UNIX or Windows functions, you will need to take care to convert paths appropriately. The UNIX commands can actually accept forward- or backward-slashed paths. Windows commands, however, must get backward-slashed paths. To accommodate converting between the two, you should use the convertPathToPlatform function. There is currently no function however to convert from Windows style to UNIX/JavaScript style. This needs to be remedied at some point.

`convertPathToHFS` returns an empty string on Windows.

Perl and PHP

There is no Perl or PHP support in the Widget Engine for Windows. The size of what would need to be included is prohibitive as a standard part of our install, so we recommend those Widgets that demand Perl point users to an appropriate Perl

environment for the PC.

Window Shadows

Window shadows are not currently supported on Windows. If you decide to render your artwork with your own shadows, be sure to set the shadow property of your window to 0. If not, when and if Windows does support shadows you'll end up with a double-shadow.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

U

V

W

X

Y