Konfabulator

Reference Manual

July 22, 2003

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

Thanks to all who have submitted comments and corrections.

The Basics

Konfabulator 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 Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>

    <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>
</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 terrifically 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).

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. These are packaged together in a Mac OS X *bundle* which is a directory that is treated as a single unit by the operating system. A `.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 above). Control-click on one of the default Widgets and choose the **Show Package Contents** option to see this structure in use.

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>
```

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 like is commonly done in HTML, like so:

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

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

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

File paths in Konfabulator are always relative to the location of the XML file so 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.

By far the best and easiest way to get started creating Konfabulator Widgets is to take an existing Widget and start making changes to it. Konfabulator 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.

Konfabulator 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/>
  <hotkey/>
  <image/>
  <preference/>
  <text/>
  <textarea/>
  <window/>
</widget>
```

<about-box>

block to define images for an about box

Attributes

`image`

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

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>
```

<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 in one of two ways. In the `file` attribute for the `action` block, or as the sole item listed between the `action` XML blocks.

Example

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

<action trigger="onLoad">
    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, Konfabulator 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>
```

trigger

the event that triggers the enclosed code

Values

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

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 to have an active and inactive state to your Widget. This action is also triggered when the Widget first starts running.

`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.

`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 to have an active and inactive state to your Widget.

`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).

`onTimer` executes at regular intervals based on what's defined in the `interval` attribute. If no `interval` attribute is defined, Konfabulator will default to a one minute interval. Note that there can only be one `onTimer` trigger per Widget.

`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.

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

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>
```

<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 Command+Shift+F1.

Hotkey objects can also be created and destroyed dynamically via Konfabulator's 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. Command+Tab). If more than one Widget or application uses the same hotkey then all receive a notification when the user presses those keys.

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 HotKey()
delete newObjectName
```

Example

```
<hotkey name="hkey1">
  <key>F1</key>
  <modifier>command+shift</modifier>
  <onKeyUp>focusWidget();</onKeyUp>
</hotkey>
```

key

the name of the function key

Description

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, Help, Home, PageDown,
PageUp, Space, Tab
```

A modifier is always used which is **Command** by default.

JavaScript

myObjectName.key

Example

```
<hotkey name="hkey1">
  <key>F1</key>
</hotkey>

hkey1.key = “F2”;
```

modifier

the modifier keys for the hotkey

Description

The modifier attribute can be any combination of:

```
command, control, option, shift
```

A modifier is always used and is **Command** by default.

JavaScript

myObjectName.key

Example

```
<hotkey name="hkey1">
  <key>F1</key>
  <modifier>command+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>F1</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 that it is generally best to attach key code to the `onKeyUp` action as that is what users expect.

JavaScript

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

Example

```
<hotkey name="hkey1">
  <key>F1</key>
  <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 release 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>F1</key>
  <modifier>command+control</modifier>
  <onKeyUp>focusWidget();</onKeyUp>
</hotkey>
```

<image>

block defining an image and associated default properties

Attributes

```
alignment
height
hOffset
hRegistrationPoint
name
onDragDrop
onDragEnter
onDragExit
onKeyUp
onKeyDown
onMouseDown
onMouseEnter
onMouseExit
onMouseMove
onMouseUp
opacity
rotation
src
useFileIcon
vOffset
vRegistrationPoint
width
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 Konfabulator's 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 the text block 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’;
```

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).

JavaScript

`myObjectName``.height`

Example

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

myButton.height = 30;
```

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 Widget's window. 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>
```

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;
```

onDragDrop

the script called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string are 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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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).

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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`.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

myObjectName.onDragExit

Example

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

well.onDragExit = “unhighlightDropTarget(well);”;
```

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 image is specified with the `onKeyDown` attribute. Note that it is generally best to attach key code to the `onKeyUp` action as that is what users expect.

JavaScript

`NewObjectName``.onKeyDown`

Example

```
<image src="well.png">
  <name>well</name>
  <onKeyDown>
       print(system.event.keyPress);
  </onKeyDown>
</image>

well.onKeyDown = “keypressed = true”;
```

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 image is specified with the `onKeyUp` attribute.

JavaScript

`NewObjectName``.onKeyUp`

Example

```
<image src="well.png">
  <name>well</name>
  <onKeyUp>
       print(system.event.keyPress);
  </onKeyUp>
</image>

well.onKeyUp = “keypressed = true”;
```

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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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 that's hidden unless you're hovering over the Widget.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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 that's hidden unless you're hovering over the Widget.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

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 moves the mouse cursor within the bounds of an object. The current mouse position is available in the `system.event` object.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object 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).

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

`myObjectName``.onMouseUp`

Example

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

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

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;
```

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 uses a path to it on your hard drive relative to the XML file of the Widget it's referenced from.

JavaScript

myObjectName.src

Example

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

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>
```

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 Widget's window. The greater the value assigned, the farther down the image will be drawn.

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 based on the `vOffset` you gave it.

JavaScript

`myObjectName``.vRegistrationPoint`

Example

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

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).

JavaScript

myObjectName`.width`

Example

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

myButton.width = 20;
```

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++;
```

<preference>

block defining a preference setting and associated properties

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 tells the preference 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 UI.

It's optional, but highly recommend to explain the preference and it's 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</foo>
</preference>
```

hidden

is the preference is 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 either `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 users 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>
```

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 UI.

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 UI object is used to display the data choices.

Type can be `text`, `checkbox`, `popup`, `selector` or `slider`. When `checkbox` is used, the value returned is either 0 or 1. When `popup` is used, the values are defined as `<option>` attributes.

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 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;
```

<text>

block defining a text object and associated default properties

Attributes

```
alignment
color
data
font
height
hOffset
name
onDragDrop
onDragEnter
onDragExit
onKeyUp
onKeyDown
onMouseDown
onMouseEnter
onMouseExit
onMouseMove
onMouseUp
opacity
size
style
vOffset
width
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 Konfabulator's 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

Set 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>
```

data

the text that the text object draws

Description

The text to be displayed. Note that any newlines 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.

Example

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

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 Widget's window. 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;
```

onDragDrop

the script called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string are 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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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).

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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`.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

`myObjectName``.onDragExit`

Example

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

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

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 text object is specified with the `onKeyDown` attribute. Note that it is generally best to attach key code to the `onKeyUp` action as that is what users expect.

JavaScript

`NewObjectName``.onKeyDown`

Example

```
<text data="Type Stuff Here">
  <name>typo</name>
  <onKeyDown>
       print(system.event.keyPress);
  </onKeyDown>
</text>

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

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 text object is specified with the `onKeyUp` attribute.

JavaScript

`NewObjectName``.onKeyUp`

Example

```
<text data="Type Stuff Here">
  <name>typo</name>
  <onKeyUp>
       print(system.event.keyPress);
  </onKeyUp>
</text>

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

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.

If you would like to reference an external JavaScript file, it must be wrapped between the blocks (note you cannot use a `file=` attribute with an object action).

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.

If you would like to reference an external JavaScript file, it must be wrapped between the blocks (note you cannot use a `file=` attribute with an object action).

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.

If you would like to reference an external JavaScript file, it must be wrapped between the blocks (note you cannot use a `file=` attribute with an object action).

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 moves 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.

If you would like to reference an external JavaScript file, it must be wrapped between the blocks (note you cannot use a `file=` attribute with an object 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).

If you would like to reference an external JavaScript file, it must be wrapped between the blocks (note you cannot use a `file=` attribute with an object action).

JavaScript

myObjectName.onMouseUp

Example

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

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

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";
```

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

brief description

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.

Example

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

myLabel.style = ‘italic’;
```

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 Widget's window. 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;
```

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
color
columns
bgColor
bgOpacity
data
editable
font
height
hOffset
lines
name
onDragDrop
onDragEnter
onDragExit
onKeyUp
onKeyDown
onKeyPress
onMouseDown
onMouseEnter
onMouseExit
onMouseUp
opacity
scrollbar
size
spellcheck
style
vOffset
width
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 Konfabulator's 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 wont 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>
```

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.

Example

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

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 Widget's window. 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;
```

onDragDrop

the script called when something is dropped on the object

Description

The `onDragDrop` trigger fires when a file, URL or string are 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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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).

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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`.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

myObjectName.onDragExit

Example

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

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

onKeyPress

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

Description

The onMouseDown 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`).

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

myObjectName.onMouseDown

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);”;
```

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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

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.

If you want to put the code in an external JavaScript file, its name should be specified in place of the JavaScript code (note you cannot use a `file=` attribute with an object action).

JavaScript

myObjectName.onMouseUp

Example

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

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

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;
```

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’;
```

vOffset

the vertical offset of a textarea 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 Widget’s window. 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.

See also the `columns` attribute.

JavaScript

myObjectName.width

Example

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

ta1.width = 30;
```

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++;
```

<widget>

block to define the scope of the Widget

Attributes

debug
minimumVersion
option
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.

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 Konfabulator 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).

minimumVersion

the minimum version of Konfabulator that is required to run this Widget

Description

Specifying `minimumVersion` for a Widget causes Konfabulator 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 wont run.

Example

```
<widget minimumVersion="1.5">
  ...
</widget>
```

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. `hOfset`) can be very difficult to locate.

```
<option>dontRememberWindowPosition</option>
```

This option tells Konfabulator to not save the window position of this Widget when it is closed. Normally, the positions of all Widgets are remembered between invocations of Konfabulator so the user can layout 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 Konfabulator XML tags and attribute names. If you want to embed XML not recognized by Konfabulator 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.

<window>

block that defines the main window of the Widget

Attributes

height
hOffset
level
name
opacity
shadow
title
visible
vOffset
width

Description

The `window` block describes the size and position of the Widget's main window. This window is always transparent and only the images and text objects you put in it are visible to the user.

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: `below`, `normal` or `topMost`. It specifies how the window behaves with respect to other windows on the desktop, whether it appears below others or floats above everything. 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);
```

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.

Note: as of Max 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;
```

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 wont appear on the screen and you wont 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;
```

Konfabulator JavaScript Reference

This section describes the extensions to JavaScript that are provided by Konfabulator. If JavaScript is new to you, consider obtaining a guide to the language to help with its syntax and structure. Konfabulator implements a JavaScript engine that conforms to version the JavaScript 1.5 standard (ECMA-262, revision 3).

Konfabulator's extensions fall into several categories:

- global functions
- system functions
- system attributes
- system objects
- Konfabulator object methods

Global Functions

These can be used anywhere in Konfabulator 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 end user an immediate message in an 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 ScriptEditor 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();
```

convertPathToHFS()

converts a UNIX style path to an HFS one

Synopsis

`convertPathToHFS(`*`myPath`*`)`

Description

Converts a UNIX style path (with '/'s) to a Mac HFS style path (with a volume name and ':'s).

Example

```
convertPathToHFS('/Users/perry/foo.txt');
```

yields:

```
Paragon:Users:perry: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>F1</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].maxLength = 250;
formfields[0].minLength = 10;
formfields[0].validator = 'TextPrefValidator()';
formfields[0].validationMesg = 'Please enter a valid
text Pref';
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");
}
```

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 Konfabulator would return true while your browser may still complain.

Example

```
openURL("http://www.konfabulator.com");
openURL("ftp://myname:pa55w0rd@ftp.mysite.com");
```

See Also

escape(), unescape(), URL.fetch()

play()

play a sound file

Synopsis

`play(`*`pathToSound`*`)`

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.

Example

```
play("sounds/sample.mp3");
```

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 when 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(`*`[limit1, limit2]`*`)`

Description

Generate a random number, optionally within given limits.

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/perry`)
- 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 a 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*).

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.

Example

```
<action trigger="onLoad">
  var yahooData;
  runCommandInBg("curl www.yahoo.com", yahooData);
</action>

<action trigger="onRunCommandInBgComplete">
  print("Yahoo's home page is " +
        yahooData.length + " bytes");
  print("Yahoo's home page is " +
        eval(system.event.data + '.length') + " bytes");
</action>
```

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 an improve performance or hide messy interim states from the Widget user.

See Also

`resumeUpdates(), updateNow()`

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()`

System Attributes

These give Konfabulator JavaScript code access to various system settings and hardware information.

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.airport

built in support for accessing AirPort information

Attributes

`available`	true if AirPort is installed.
`info`	a summary of airport status.
`network`	the name of the current network.
`noise`	the connection's noise level.
`powered`	true if AirPort is powered on.
`signal`	the connection's signal level.

Description

The settings and status of an installed AirPort card are available through the `system.airport` object.

The AirPort Widget makes extensive use of this object.

Example

```
if (system.airport.available && system.airport.powered)
  alert("Current network is " + system.airport.network);
```

system.airport.available

determine if an AirPort (or other compatible wireless card) is installed

Synopsis

```
system.airport.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";
```

See Also

system.airport.signal

system.airport.info

AirPort status summary

Synopsis

```
system.airport.info
```

Description

A brief, human readable description of AirPort status.

Example

```
alert(system.airport.info);
```

system.airport.network

return name of current AirPort network

Synopsis

```
system.airport.network
```

Description

This attribute contains the name of the current AirPort network, if any.

Example

```
alert("AirPort network " + system.airport.network + " in
use");
```

system.airport.noise

the noise level of the current AirPort connection

Synopsis

```
system.airport.noise
```

Description

This attribute contains a numeric value which indicates the level of noise on the current airport connection.

Example

```
if (system.airport.noise > 20)
  status.src = “noisy.png”;
```

system.airport.powered

boolean that indicates if the AirPort card is on or off

Synopsis

`system.airport.powered`

Description

This boolean variable indicates whether the AirPort is currently turned on or off. Use this to decide whether to access the other AirPort status attributes.

Example

```
if (system.airport.available && system.airport.powered)
  alert("Current network is " + system.airport.network);
```

system.airport.signal

return the signal strength of the current AirPort connection

Synopsis

`system.airport.signal`

Description

The `signal` property of the 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 Konfabulator, 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"
```

See Also

system.airport.available, system.airport.powered

system.appearance

the current system appearance

Synopsis

```
system.appearance
```

Description

The current appearance of the system. As of Mac OS X 10.2 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 Konfabulator we do not support notifying Widgets of an Appearance change.

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`.

system.battery[*n*].currentCapacity

the charge in the battery

Synopsis

`system.battery[`*`batteryNumber`*`].currentCapacity`

Description

Current mAh of the battery.

system.battery[*n*].isCharging

charging state

Synopsis

`system.battery[`*`batteryNumber`*`].isCharging`

Description

True if battery is being charged.

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 mAh of the battery.

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 if the device 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).

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;
```

system.clipboard

accesses the current system clipboard

Synopsis

`system.clipboard`

Description

`system.clipboard` contains the text (if any) on the Mac OS clipboard. Setting this attribute will load the Mac OS clipboard with that data, removing anything there previously.

Example

```
myText = system.clipboard;
myNewText = "--<(" + myText + ")>--";

system.clipboard = myNewText;
```

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.

See also: `system.event.keyString`

Example

```
print("Key: " + system.event.key);
```

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.

See also: `system.event.key`

Example

```
print("Key Name: " + system.event.keyString);
```

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.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 who's action was triggered).

Example

```
print("Mouse: " + system.event.screenX + ", " +
   system.event.screenY);
```

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.mute()

get or set the mute state of your system volume

Synopsis

`system.mute(`*`onOff`*`)`

Description

Called without an argument returns if the machine is muted or not. Called with a boolean, will set the mute state of the machine.

Example

```
// 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!");
```

system.volume()

get and set the system audio volume

Synopsis

`system.volume(`*`[number]`*`)`

Description

Used without an argument returns the current volume. You can also pass it a number between 0 and 16 to change the system volume level. Setting the volume to 0 (zero) is the same as calling `system.mute(true)`.

Example

```
// Set the machine to half volume
system.volume(8);
```

Konfabulator Object Methods

Image Methods

Image.fade()

fade in or out an image

Synopsis

`Image.fade(start, end, speed)`

Description

The fade() command will cause an image to fade from a starting opacity to a finishing opacity at a specified speed (1 is fastest, 100 is slowest).

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, speed)`

Description

The image's origin (`hOffset`, `vOffset`) is moved to the new coordinates specified by `newX` and `newY` at the specified `speed` (1 is fastest, 100 is slowest). 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.slide()

slide an image in a specified direction and speed

Synopsis

```
Image.slide(direction, amountOfImageToConceal>, speed)
```

Description

The slide() command is an animation effect used to simulate the Drawer in Mac OS X. It is used when you want to slide an image in a particular direction, and have it disappear into itself rather than just move. Speed of 1 is fastest, 100 is slowest.

Example

```
myImage.slide("up,left", 50, 3);
```

Text Methods

Text.fade()

fade in or out a text object

Synopsis

```
Text.fade(start, end, speed)
```

Description

The fade() command will cause a text object to fade from a starting opacity to a finishing opacity at a specified speed (1 is fastest, 100 is slowest).

Example

```
newOpacity = 0;
myText.fade(myText.opacity, newOpacity, 1);
```

Text.slide()

slide a text object in a specified direction and speed

Synopsis

`Text.slide(`*`direction, amountOfTextToConceal>, speed`*`)`

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. Speed of 1 is fastest, 100 is slowest.

Example

```
myText.slide("up,left", 50, 3);
```

TextArea Methods

TextArea.focus()

make the current textarea object the focus of keypresses

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.

Example

```
myTextArea.focus();
```

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 keypress will be 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.replaceSelection()

replace the current selection in a textarea with a string

Synopsis

`Text.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

`Text.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:

```
ta1.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:

```
ta1.select(10, 10);
```

To set the insertion point after any text already in the textarea you would use:

```
ta1.select(-1, -1);
```

Example

```
ta1.select(5, 15);
```

URL Methods

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

`fetch()` Retrieve the specified URL as a string.

Example

```
var url = new URL();
location = "http://www.yahoo.com";
contents = url.fetch(location);
```

URL.fetch()

return URL data as string

Synopsis

`URL.fetch(validURL)`

Description

Retrieves data from the remote location specified. This is done synchronously so the Widget will pause until the data is retrieved.

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` is fine because different people's Widgets will be started at different times).

Example

```
var url = new URL();
location = "http://www.yahoo.com";
contents = url.fetch(location);
```

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.

Example

```
if (myWindow.shadow)
  myWindow.recalcShadow();
```

Index

O

P

R

S

T

U

W

X