iControlWeb Scripting Manual

JavaScript support in iControlWeb builds on core JavaScript (ECMAScript). It further implements E4X (ECMAScript for XML), which makes XML a first-class data type in JavaScript. This document outlines the features that are specific to iControlWeb's implementation. The reader is expected to have a basic understanding of JavaScript. If you need to learn or refresh your memory, a good starting point is http://www.mozilla.org/js/scripting/.

Object hierarchy

- **`navigator`**– Navigator
    - **`gsm`**– GSM
    - **`history`**– History
    - **`scriptingHosts[]`**– RemoteScripting
    - **`windows[]`**– Window
        - **`document`**– Document
            - WebBrowser
            - Button
                - **`alarms`**– AlarmSet
            - Player
            - VTRControl
            - Widget
            - AlarmPanel
            - LogViewer
                - **`results`**- LogEntry
            - Zone
                - **`document`**– Document
                  or
                - **`widget`**– Widget
                  or
                - iControl component (**`VTRControl`**, iControl service panel, GSM alarm browser, iNavigator, …)

Please note: the **`browsers[]`**, **`buttons[]`**, **`players[]`**, **`vtrControls[]`**, and **`zones[]`** arrays have been *deprecated* and should not be accessed directly. Access to any component located within a page or widget is through the **`getElementById()`** method (see: Document or Widget).

Scriptable Components

History	The **`History`** object allows you to navigate through the history of iControlWeb pages that the associated window has displayed.
Gateway	The **`Gateway`** object represents an iControl XML Gateway, which is used to communicate with remote gateway-enabled processes using an XML-based protocol.

KaleidoAlto	The `KaleidoAlto` object is used to communicate with a Kaleido Alto or Quad multi-image display device from Miranda.
KaleidoK2	The `KaleidoK2` object is used to communicate with a Kaleido K2 multi-image display device from Miranda.
KaleidoX	The `KaleidoX` object is used to communicate with a Kaleido X multi-image display device from Miranda.
RCP100	The `RCP100` object is used to communicate with an iControl RCP-100 control panel from Miranda.
Router	The `Router` object is used to communicate with a routing switcher configured in iControl Router.
Workbook	The `WorkBook` object is used to represents an Excel WorkBook.
GSMAlarm	The `GSMAlarm` component provides introspection into an alarm published on GSM.
Button	The `Button` object represents a script-enabled button within an iControlWeb page.
TreeTable	The `TreeTable` component is a cross between a tree and a table. You can see it as a table which contains a tree in its first column, with expandable/collapsable nodes. It can be used as a simple tree or a simple table when required.
PopupMenu	Allows the creation of a script-controlled pop-up menu.
MouseEvent	The `MouseEvent` object is a standard W3C DOM level 2 event used as a parameter to event handlers like `onClick()`.
AsiControl	The `AsiControl` object provides you with the ability to communicate with the ASI service.
XMLHttpRequest	The `XMLHttpRequest` object can be used to easily retrieve data via HTTP.

Quick Reference

$Id: iControlWebScriptingManual.html,v 1.55 2008/05/15 20:19:38 mcormier Exp $

Navigator

The `Navigator` object represents the iControlWeb application as a whole. There is only one `Navigator` object in the application, and it is accessible as the global variable `navigator`. It is also the parent of all `Window` objects.

Property/Method	Description
`appCodeName`	Represents the internal code name of the application. Read-only.
`appName`	Read-only property used to get the name of the application, for instance "iControl Web".
`appVersion`	Read-only property used to get the application version string.
`blinkWhenUnacknowledged`	A read/write `boolean` property that defaults to `false`. When set to true, status icons for alarms that were not acknowledged by an operator will blink. Acknowledging the alarm stops the blinking.
`gsm`	References the `GSM` view of iControlWeb. Note that this may actually reference a cluster of GSM servers as a unified view.
`includeJS(script)`	This function is used to load an external JavaScript file into the context of the navigator object. By using this method you can create reusable JavaScript libraries. The parameter is an absolute URL to load the script from, as a `string`.
`gsms[]`	Array of `GSM` objects indexed by the IP address of the corresponding GSM server.
`language`	Read-only property used to get the language of the application.
`mainUserRole`	Read-only `string` property identifying the main role of the user currently logged in to the application. Will be `null` if security hasn't been enabled for a site. Other predefined values include `"no role"` and `"super"`.
`scriptingHosts[]`	Read-only `Array` of the currently known remote scripting hosts, indexed by their `serviceID` `string`. The values are of type `RemoteScripting`.
`securityDomain`	Read-only property identifying the security domain path for the current application. Will be `null` if security hasn't been enabled for a site.
`serviceID`	Read-only `String` property reflecting the Jini serviceID used by the application. Might be `null` if the application is not currently running as a Jini service.
`username`	Read-only property identifying the user currently logged in to the application. Will be `null` if security hasn't been enabled for a site.
`vendor`	Read-only property used to get the vendor of the application.
`operationalMode`	Read-only dynamic property referencing the current *client-side* operational mode(s) of the application.

`unavailableTextAlarmLabel`	A `string` that is used as the value for text alarms that become unavailable for some reason (GSM lost, device removed, etc.). The default value is `(unavailable)`.
`NO_MODE`	The value of `operationalMode` when no mode is set.
`OFFLINE`	A possible value for `operationalMode`.
`MAINTENANCE`	A possible value for `operationalMode`.
`windows[]`	Array of Window objects. As of version 2.5, the array contains only a single element at index 0.
`connectCrosspoint()`	This method switches a single router cross point. It takes four parameters: the router ID (String), the level, the source and destination (as integers). The source and destination are 1-based integers.
`exec()`	This method allows launching external applications. It takes a `string` parameter representing the executable to launch (which may be in the system path), then an optional `Array` of `string` arguments. It returns an object representing the process, on which you can call two methods: `waitFor()` to block until the application is closed, `destroy()` to forcibly close terminate the application, and `exitValue()` which obtains the exit code of the application once it has exited. You can ignore the return value of `exec()` if all you want is to launch the application and be done with it.
`getServices(regex)`	This method returns a list service longIds that match the supplied regular expresion.
`getService(longId)`	This method returns a service proxy which matches the supplied longID.
`setOperationalMode()`	This method sets or unsets the specified *client-side* operational mode (s) of the application. It takes two parameters of types `int` and `boolean`. The first parameter specifies the mode, and the second specifies whether to set or unset (true or false). Note 1: `NO_MODE` cannot be used. Note 2: The page may need to be refreshed in order to display the statuses accordingly.
`resetAllLatches()`	Resets all alarm latches for the application.
`getAudioClip()`	This method takes a single parameter representing the fully-qualified URL of a sound (WAV, AIFF, AU) file. The returned object can be controlled using its three methods: `play()`, `loop()` and `stop()`.
`getRouter()`	Returns a `Router` object for a specific router. It takes one argument: the router ID.

Top Quick Reference

GSM

The GSM object may represent one or many remote GSM services and provides a unified view of the alarms managed by these GSM services.

Property/Method	Description
activateAlarm()	This method attaches a function or method to a GSMAlarm that gets invoked every time the status or text of the alarm changes. It takes two arguments. The first is the URI of the alarm, as a string. The second is either a reference to a function or to an object that has an onAlarm() method. In either case, the function or method should accept a single argument which will be a GSMAlarm object reflecting the latest status of the alarm. The first call to the provided function or method will provide the initial status of the alarm, even when it hasn't changed recently. The notifications will stop automatically soon after there are no more references to the second argument elsewhere in the code.
getAlarm()	The getAlarm() method is used to query the current status of a GSM alarm from the server. It takes a single string parameter that represents the URI (unique identifier) of the alarm in question. The return value is of type GSMAlarm. Note that it provides only a snapshot of the alarm, and will not update in real time. See activateAlarm() if you need dynamic updates and notifications.
logEvent()	This method takes two parameters: a GSMAlarm that the event relates to, and a message string to attach to the event. Given those informations, it will log an event that can later be seen in the GSM log viewer.
getTreeTableModel()	This method takes a single parameter, representing the path of the GSM folder to use as the tree root (a JavaScript Array of strings). The resulting model will be backed by any and all known GSM services.
eval()	The eval() method takes a single String parameter representing JavaScript code to be evaluated in the context of the remote GSM. It returns the value returned by the evaluated script element. An example would be: `eval("navigator.appVersion;");`

Top Quick reference

History

The `History` object allows you to navigate through the history of iControlWeb pages that the associated window has displayed. iControlWeb stores a history of visited URLs in a list, which the `History` object references.

Property/Method	Description
`current`	Refers to the current URL in the history list (read-only).
`length`	Returns the number of entries in the history list (read-only).
`next`	Refers to the next URL in the history list (read-only).
`previous`	Refers to the previous URL in the history list (read-only).
`back()`	Loads the URL for to the previous page in the history list.
`forward()`	Loads the URL for to the next page in the history list.
`go()`	Loads a URL from the history list. This method takes a single integer parameter, which indicates which page to load in reference to the current page. Negative values move back in the history, while positive values move forward. `go(-1)` is therefore equivalent to `back()`, and `go(1)` is equivalent to `forward()`.

Top Quick Reference

RemoteScripting

`RemoteScripting` objects represent remote scripting hosts, typically other instances of iControlWeb.

Property/Method	Description
`eval()`	The `eval()` method takes a single `String` parameter representing JavaScript code to be evaluated in the context of the remote host. It returns the value returned by the evaluated script element. An example would be: `eval("navigator.windows[0].alert('It works!');");`

Top Quick Reference

Window

The Window object represents an iControlWeb window, that is, a window defined by the operating system. As of version 2.5, there is only one window per application, and it is accessible as navigator.windows[0].

Property/Method	Description
parent	References the application that created this window. See the Navigator object for more information.
document	References all the information about the document within this window. See the Document object for more information.
history	The history property of the Window object contains an array of the names and URLs of the pages the window has visited. A specific URL in the history array can be accessed by specifying the indexed location that represents the URL about which you want to retrieve information.
fullscreen	A read/write boolean property read to figure out whether you are currently in full-screen mode, and that you could write to toggle full-screen mode.
totalFullscreen	A read/write boolean property read to figure out whether you are currently in total full-screen mode, and that you could write to toggle total full-screen mode.
alert()	This method displays an alert dialog box when invoked. The value of the string passed as a parameter to the method is displayed in the box.
confirm()	This method displays a confirmation dialog box when invoked. The value of the string passed as a parameter to the method is displayed in the box. This box will contain an OK and a Cancel button. The method returns a boolean value of true if the user clicks OK and false if the user clicks Cancel or closes the dialog box.
loadDocument()	A utility method that creates a Document object and uses it as the content of the window. The method takes a single URL string as a parameter. The URL may be relative or fully specified. This method may throw an exception that you can catch if the page is not found.
prompt()	This method will display a prompt dialog box when invoked. It takes two string parameters; the first one is displayed in the box, and the second one is contained in the text field of the prompt dialog box. The returned value is the text string contained in the text field when the user clicks OK, or null if the user clicks Cancel or closes the dialog box.

`promptPassword()`	This method will display a prompt dialog box with a password field when invoked. It takes one `string` parameter; this parameter is displayed as a message in the dialog. The returned value is the password `string` contained in the text field when the user clicks `OK`, or `null` if the user clicks `Cancel` or closes the dialog box.
`setTimeout()`	A utility method that schedules a task for execution after a given delay. This method takes two parameters: a task and a timeout delay (in milliseconds) before the task is executed. The task can be one of two things: either a JavaScript `Function` or a script string to be evaluated. In either case, `this` in the script refers to the `Window`. One advantage of using a `Function` over a `string` is that the environment where the function was defined remains accessible inside the function (this is technically called a *closure*). The method returns an object that can be saved and used to cancel the task later on using `clearTimeout()`.
`clearTimeout()`	This method can be used to cancel a task that was previously scheduled using `setTimeout()`. It takes a single parameter that must be the object that was returned when the task was initially scheduled.
`setInterval()`	A utility method that schedules a task for execution each time a given time interval has elapsed. This method takes two parameters: a task and an interval delay (in milliseconds) between task executions. The task can be one of two things: either a JavaScript `Function` or a script string to be evaluated. In either case, `this` in the script refers to the `Window`. One advantage of using a `Function` over a `string` is that the environment where the function was defined remains accessible inside the function (this is technically called a *closure*). The method returns an object that can be saved and used to cancel the task later on using `clearInterval()`.
`clearInterval()`	This method can be used to cancel a task that was previously scheduled using `setInterval()`. It takes a single parameter that must be the object that was returned when the task was initially scheduled.
`openServicePanel()`	A utility method that opens a `ServicePanel` object in its own window. This method takes a single `string` parameter representing the MirandaLongID for the device whose panel to show, and returns a handle that can be saved and used to later close the window by passing the handle to `Button.closeServicePanel()` or `Window.closeServicePanel()`.
`closeServicePanel()`	A utility method that closes a `ServicePanel` object previously opened by `Button.openServicePanel()` or `Window.openServicePanel()`. This method takes a single `string` parameter, the handle that was returned by `openServicePanel()`.
`includeJS(script)`	This function is used to load an external JavaScript file into the context of the window. By using this method you can create reusable JavaScript libraries that are not bound to an `.mpf` file. These JavaScript files are typically stored in the `Scripts` directory of the current site, but you can also specify an absolute URL to load the script from.

Top Quick Reference

Document

The `Document` object represents an iControlWeb page (an `.mpf` file).

Property/Method	Description
`parent`	References the zone or window that directly contains the document.
`background`	A read-only string property specifying what background is used for the document.
`parameters[]`	A parsed representation of the query string of the page URL. The value is an `Array` with the keys of the array being the parameter names and the values being the parameter values. The parameter values are already URL-decoded.
`siteName`	References the name of this document's site. This is a read-only `string` property.
`version`	The version of the application where the document was created. This is a read-only `number` property. *(since 3.30)*
`window`	References the top-level window that this document belongs to. See the `Window` object for more information. Might be the same as `parent` if this is the window&apos;s top-level document.
`URL`	References the URL of the currently loaded site document. This is a read/write String property. When being set, the new URL may be relative to the current document or fully qualified. An exception is thrown if the page is not found.
`reload()`	This method will refresh the current document from the server.
`viewSource()`	This method shows the source of the current document in a new window.
`onInit()`	The `init` event handler is defined in iControlWeb Creator (page properties) and specifies what to do when the page is about to be loaded. It doesn&apos;t take any parameter. When documents are embedded (as initial value of a zone) in other documents, the `load` event handler of the parent document will be executed *before* any of the `load` or `init` event handlers of any of its child documents have run.
`onLoad()`	The `load` event handler is defined in iControlWeb Creator (page properties) and specifies what to do when the page is fully loaded. It doesn&apos;t take any parameter. When documents are embedded (as initial value of a zone) in other documents, the `load` event handler of the parent document will only be executed *after* the `load` event handlers of all its child documents have completed.

`onUnload()`	The `unload` event handler is defined in iControlWeb Creator (page properties) and specifies what to do when the page is unloaded. It doesn&apos;t take any parameter.
`includeJS(script)`	This function is used to load an external JavaScript file into the context of the document. By using this method you can create reusable JavaScript libraries that are not bound to an `.mpf` file. These JavaScript files are typically stored in the `Scripts` directory of the current site directory, but you can also specify an absolute URL to load the script from.
`getElementById()`	This function is used to get an object by its script name located within this document. It replaces the direct array access. It takes one `String` argument, which is the script name of the object that is desired. It is assumed that all script names are unique and that only one script name is mapped to one object. If duplicate script names exist, this method will return the last object that was added to this document by that name.
`getBrowserByIndex()`	This function is used to get a web browser by its index located within this document. It replaces the direct array access. It takes one `int` argument, which is the index of the web browser that is desired. Because web browsers are not indexed by script names, but by their index value when added, it is recommended to use this access function in place of the direct array access.
`browsers[]`	An array referencing all the embedded Web browsers (Internet Explorer or Mozilla) defined in the document, indexed by numbers (assigned in order). **Deprecated.** *As of iControl 3.0 this direct access of the array has been deprecated. Although still supported for backwards compatibility, the recommended approach is to use `getBrowserByIndex()` with the desired `int` index value.*
`buttons[]`	An array referencing all the named buttons defined in the document, indexed by button name. **Deprecated.** *As of iControl 3.0 this direct access of the array has been deprecated. Although still supported for backwards compatibility, the recommended approach is to use `getElementById()` with the desired `String` script name.*
`players[]`	An array referencing all the named video stream players defined in the document, indexed by player name. **Deprecated.** *As of iControl 3.0 this direct access of the array has been deprecated. Although still supported for backwards compatibility, the recommended approach is to use `getElementById()` with the desired `String` script name.*
`vtrControls[]`	An array referencing all the named VTR controls defined in the document, indexed by VTR name. **Deprecated.** *As of iControl 3.0 this direct access of the array has been deprecated. Although still supported for backwards compatibility, the recommended approach is to use `getElementById()` with the desired `String` script name.*

zones[]	An array referencing all the zones defined in the document, indexed by zone name. **Deprecated.** *As of iControl 3.0 this direct access of the array has been deprecated. Although still supported for backwards compatibility, the recommended approach is to use* **getElementById()** *with the desired* **String** *script name.*

Top Quick Reference Example

WebBrowser

The `WebBrowser` object represents an embedded Web browser (either Internet Explorer or Mozilla, depending on platform).

Property/Method	Description
`parent`	References the zone that contains the browser. See the `Zone` object for more information.
`window`	References the top-level window that this browser belongs to. See the `Window` object for more information.
`document`	References the `Document` that contains the browser. In the case of nested documents, this always refers to the innermost document that immediately contains the browser.
`url`	References the URL shown in the Web browser. This is a read/write `String` property.
`onStatusChange()`	An event handler that can be attached to the Web browser in order to be notified when the status text for the browser window changes. It takes a single `String` parameter representing the new browser status text.

Top Quick Reference

Button

The Button object represents a script-enabled button within an iControlWeb page.

Property/Method	Description
parent	References the zone that contains the button. See the Zone object for more information.
window	References the top-level window that this button belongs to. See the Window object for more information.
document	References the Document that contains the button. In the case of nested documents, this always refers to the innermost document that immediately contains the button.
alarms	References the set of alarm descriptors associated to the button. See the AlarmSet object for more information.
status	Read-only dynamic property referencing the current selected status(from the dropdown menu) of the alarm associated with this button.
currentStatus	Read-only dynamic property referencing the current status of the alarm in GSM.
latchedStatus	Read-only dynamic property referencing the current latched status of the alarm in GSM.
acknowledgementStatus	Read-only dynamic property referencing the current acknowledgement status of the alarm in GSM.
operationalMode	Read-only dynamic property referencing the current operational mode of the alarm in GSM.
NORMAL	A possible value for status.
WARNING	A possible value for status.
ERROR	A possible value for status.
UNKNOWN	A possible value for status.
DISABLED	A possible value for status.
NON_EXISTENT	A possible value for status.
PENDING	A possible value for status. This is a not a GSM status, it's just a client status to indicate that we have not received the GSM status yet. All iControl Web alarms are initiated with the PENDING status. If a GSM service is stopped, then all iControl Web alarms activated on this GSM will have their status reset to PENDING.

`blinking`	A read/write `boolean` property; when `true`, the status indicator of the button will blink.
`image`	The URL of the script-specified base image for the button. If this is non-`null`, it will override the base image for the button (specified in iControlWeb Creator). If it is `null` it will effectively remove any previously set base image. The URL may be relative to the parent document or fully specified.
`selectedImage`	The URL of the script-specified selected image for the button. If this is non-`null`, it will override the selected image for the button (specified in iControlWeb Creator). If it is `null` it will effectively remove any previously defined selected image. The URL may be relative to the parent document or fully specified. *(since 3.30)*
`name`	Read-only string property representing the button name, as specified in the "Script" tab in iControlWeb Creator.
`value`	The value of the text displayed by this button. This is a read/write property.
`group`	Read-only string property representing the button group the button belongs to, as specified in iControlWeb Creator.
`selected`	Read/write boolean property representing whether the button is selected or not.
`textColor`	References the button text color. This is a read/write `string` property, where the color is specified using CSS 2.1 syntax. These examples all specify the same color (red): `#f00` (`#rgb`), `#ff0000` (`#rrggbb`), `rgb (255,0,0)`, `rgb(100%, 0%, 0%)`. (Introduced in 2.70)
`visible`	References the button visibilty. This is a read/write `boolean` property. Note that if a button is invisible, it will still be notified of status and text changes on its associated alarms.
`toolTip`	References the button tool tip. This is a read/write `string` property.
`click()`	The `click()` method simulates the `click` event on the `Button`.
`onClick()`	The `click` event handler is defined in iControlWeb Creator and specifies what to do when the button is clicked. It takes a single parameter, called `event`, of type `MouseEvent`.
`onMouseOver()`	The `mouseOver` event handler is defined in iControlWeb Creator and specifies what to do when the mouse cursor hovers over the button. It takes a single parameter, called `event`, of type `MouseEvent`.
`onMouseOut()`	The `mouseOut` event handler is defined in iControlWeb Creator and specifies what to do when the mouse cursor stops hovering over the button. It takes a single parameter, called `event`, of type `MouseEvent`.

onStatusChange()	The statusChange event handler is defined in iControlWeb Creator and specifies what to do when the text or status of the alarm that the button monitors changes. It doesn't take any parameters. Note that when the handler is called, status and value already have their new value. Note also that if you change the text label of the button from the statusChange handler, the handler will be triggered again, so be careful in order to avoid infinite loops.
openServicePanel()	A utility method that opens a ServicePanel object in its own window. This method takes a single string parameter representing the MirandaLongID for the device whose panel to show, and returns a handle that can be saved and used to later close the window by passing the handle to Button.closeServicePanel() or Window.closeServicePanel(). This version should be favored over Window.openServicePanel() when responding to button clicks, as it tries to ensure the dialog will open close to the button that was clicked.
closeServicePanel()	A utility method that closes a ServicePanel object previously opened by Button.openServicePanel() or Window.openServicePanel(). This method takes a single string parameter, the handle that was returned by openServicePanel().

Top Quick Reference Example

AlarmSet

The `AlarmSet` object represents a container of alarm descriptors associated to a component within an iControlWeb page.

Property/Method	Description
`status`	The status alarm descriptor. If set to `null` or `undefined`, the status of the component will show up as `PENDING` (white) and no alarm registration will take place.
`text`	The text alarm descriptor. If set to `null` or `undefined`, the text of the component will be removed if a text alarm was previously assigned, and nothing will happen (i.e., the existing initial text will stay in place) otherwise.

Top

Player

The `Player` object represents a script-enabled Player within an iControlWeb page.

<table>
<tr><th>Property/Method</th><th>Description</th></tr>
<tr><td><code>assignment[]</code></td><td>Array of assignement associated to the player. The following parameters can be accessed:
<table>
<tr><td><code>video1.url</code></td><td>The video address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>video1.label</code></td><td>The video label</td><td>to be displayed as UMD</td></tr>
<tr><td><code>video1.serviceID</code></td><td>The <b>TODO</b></td><td></td></tr>
<tr><td><code>video1.vbi</code></td><td>The VBI address</td><td><code>rtsp://..._Line21</code></td></tr>
<tr><td><code>ralm1.url</code></td><td>The first RALM address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>audio1.url</code></td><td>The first audio streaming address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>ralm2.url</code></td><td>The first RALM address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>audio2.url</code></td><td>The first audio streaming address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>audio1.label</code></td><td>The audio or ralm label</td><td>to be displayed as UMD</td></tr>
<tr><td><code>vector1.url</code></td><td>The vector address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>vector1.label</code></td><td>The vector label</td><td>to be displayed as UMD</td></tr>
<tr><td><code>scope1.type</code></td><td>The video source type</td><td>"0" for analog (ex: VCP-1021) and "1" for digital (ex: SCP-1121)</td></tr>
<tr><td><code>waveform1.url</code></td><td>The waveform address</td><td><code>rtsp://...</code></td></tr>
<tr><td><code>waveform1.label</code></td><td>The waveform label</td><td>to be displayed as UMD</td></tr>
<tr><td><code>control1.url</code></td><td>The control address</td><td><code>rtsp://...</code>), used to select the line</td></tr>
</table>
Please note that <code>ralm1.url</code> and <code>audio1.url</code> (as well for <code>ralm2.url</code> and <code>audio2.url</code>) are <b>mutually exclusive.</b></td></tr>
<tr><td><code>window</code></td><td>References the top-level window that this player belongs to. See the <code>Window</code> object for more information.</td></tr>
<tr><td><code>document</code></td><td>References the <code>Document</code> that contains the player. In the case of nested documents, this always refers to the innermost document that immediately contains the player.</td></tr>
<tr><td><code>visible</code></td><td>References the player visibility. This is a read/write <code>boolean</code> property. The player is stopped when made invisible, and restarted when made visible again.</td></tr>
<tr><td><code>audioMute</code></td><td>Mute the audio if the player has streaming audio. This is a read/write <code>boolean</code> property. The audio is muted when set to true and unmuted when set to false.</td></tr>
</table>

aspectRatio	Set the video aspect ratio. This is a read/write **int** property, set values are 0=4:3, 1=16:9, 2=Auto, and -1 is returned if the aspect ratio is not available. The affect of this property change is immediate.
play()	Starts the streams associated to the player
stop()	Stops all the streams associated to the player

Top Quick Reference Example

VTRControl

The `VTRControl` object represents a script-enabled VTR control within an iControlWeb page.

Property/Method	Description
`parent`	References the zone that contains the button. See the `Zone` object for more information.
`window`	References the top-level window that this button belongs to. See the `Window` object for more information.
`document`	References the `Document` that contains the button. In the case of nested documents, this always refers to the innermost document that immediately contains the button.
`longID`	The `longID` of the VTR service that this VTR control is connected to. This is a read/write property.

`takeControl()`
`giveUpControl()`
	Those methods behave like toggleing the *Take* button on the VTR control.

`play()`
`stop()`
`rewind()`
`fastForward()`
`record()`
`eject()`
`preroll()`
`standby()`
	Those methods behave like the corresponding buttons on the VTR control

`cue("00:00:00:00")`
`mark("00:00:00:00")`
`cueToMark()`
	Cues the tape to a specific timecode. For instance, `cue("01:02:00:10")` will move the tape to 1 hour, 2 minutes, 10 frames

`currentTime` `currentMark`	(Read-only) `string` properties that return a timecode depending on the VTR control current position. The returned `string`s are in the form `"01:02:00:10"`.

Top Quick Reference Example

Widget

The `Widget` object represents an iControlWeb widget (an `.mwf` file). A widget is a subclass or child implementation of the `document` object. It has very similar behaviour as a `document` but with expanded features.

Property/Method	Description
`parent`	References the zone that directly contains the Widget.
`background`	A read-only string property specifying what background is used for the widget.
`parameters[]`	A parsed representation of the query string of the widget URL. The value is an `Array` with the keys of the array being the parameter names and the values being the parameter values. The parameter values are already URL-decoded.
`window`	References the top-level window that this widget belongs to. See the `Window` object for more information.
`URL`	References the URL of the currently loaded site widget. This is a read/write String property. When being set, the new URL may be relative to the current widget or fully qualified. An exception is thrown if the widget is not found.
`name`	Read-only string property representing the widget name, as specified in the "Script name" in iControlWeb Creator.
`reload()`	This method will refresh the current widget from the server.
`viewSource()`	This method shows the source of the current widget in a new window.
`onInit()`	The `init` event handler is defined in iControlWeb Creator (widget properties) and specifies what to do when the widget is about to be loaded. It doesn't take any parameter. When widgets are embedded (as initial value of a zone) in other pages, the `load` event handler of the parent page will be executed *before* any of the `load` or `init` event handlers of any of its child widgets have run.
`onLoad()`	The `load` event handler is defined in iControlWeb Creator (widget properties) and specifies what to do when the widget is fully loaded. It doesn't take any parameter. When Widgets are embedded (as initial value of a zone) in other pages, the `load` event handler of the parent page will only be executed *after* the `load` event handlers of all its child widgets have completed.
`onUnload()`	The `unload` event handler is defined in iControlWeb Creator (widget properties) and specifies what to do when the widget is unloaded. It doesn't take any parameter.

`includeJS(script)`	This function is used to load an external JavaScript file into the context of the Widget. By using this method you can create reusable JavaScript libraries that are not bound to an `.mpf` file. These JavaScript files are typically stored in the `Scripts` directory of the current site directory, but you can also specify an absolute URL to load the script from.
`getElementById()`	This function is used to get an object by its script name located within this Widget. It replaces the direct array access. It takes one `String` argument, which is the script name of the object that is desired. It is assumed that all script names are unique and that only one script name is mapped to one object. If duplicate script names' exist, this method will return the last object that was added to this Widget by that name. **Please note:** that the arrays browsers[], buttons[], players[], vtrControls[], and zones[] can not be accessed directly. Access to any component located within the widget is through the getElementById() method.
`getBrowserByIndex()`	This function is used to get a web browser by its index located within this Widget. It replaces the direct array access. It takes one `int` argument, which is the index of the web browser that is desired. Because web browsers are not index by script names, but by their index value when added, it is recommended to uses this access function in place of the direct array access.

Top Quick Reference Example

Alarm Panel

The `AlarmPanel` object represents a script-enabled alarm panel within an iControlWeb page.

Property/Method	Description
`parent`	References the zone that contains the alarm panel. See the `Zone` object for more information.
`window`	References the top-level window that this alarm panel belongs to. See the `Window` object for more information.
`document`	References the `Document` that contains the alarm panel. In the case of nested documents, this always refers to the innermost document that immediately contains the alarm panel.
`alarms[]`	References an array of `GSMAlarm`s that is based upon the alarms currently in the alarm panel. This array does not represent the alarms currently being displayed; it is an array of what the alarm panel reflects. This is a read/write property. To set new data for the alarm panel create a new array of `GSMAlarm`s and assign it to this property.
`name`	Read-only `string` property representing the alarm panel name, as specified in the "Script" tab in iControlWeb Creator.
`visible`	References the alarm panel's visibilty. This is a read/write `boolean` property. Note that if a alarm panel is invisible, it will still be notified of status and text changes on its associated alarms.
`enabled`	Enables or disables a panel. A disabled panel does not receive any alarm status or text updates, it does not receive folder change events, and is painted with a grey mask indicating a disabled state. This is a read/write `boolean` property.
`showCurrentColumn`	Shows the current column. This is a read/write `boolean` property.
`showLatchColumn`	Shows the latched column. This is a read/write `boolean` property.
`showAcknowledgeColumn`	Shows the acknowledged column. This is a read/write `boolean` property.
`showClientLatchColumn`	Shows the client latch column. This is a read/write `boolean` property.
`showHeaders`	Shows the header of the columns. This is a read/write `boolean` property.
`skipDisabledAlarms`	Determines if this panel will show disabled alarms within folders. When true if an alarm is specifically specified and it is disabled that alarm will be displayed. But, if a folder is specified and contains a disabled alarm, that disabled alarm will not be shown under the folder when the folder is displayed. When false all alarms are shown. This is a read/write `boolean` property.

STATUS_ALARM_TYPE	This is a convenience property to easily use to create a status alarm entry. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
FOLDER_TYPE	This is a convenience property to easily use to create a folder entry. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
FOLDER_AS_TEXT_TYPE	This is a convenience property to easily use to create a folder as text entry; where the folder entry is drawn without the status alarms. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
TEXT_ALARM_TYPE	This is a convenience property to easily use to create a text alarm entry. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
TEXT_AND_STATUS_TYPE	This is a convenience property to easily use to create a text and status alarm entry. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
FOLLOW_RUNTIME_TYPE	This is a convenience property to easily use to create an entry based upon the type retrieved from GSM. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
TITLE_TEXT_TYPE	This is a convenience property to easily use to create a text (or string) only entry. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
COMPRESSED_ALARMS_TYPE	This is a convenience property to easily use to create a compressed alarms. Compressed alarms appear side by side on a single entry line. A max of four alarms can be displayed side by side on a single entry line. It is meant to be use with the addEntry() method as the type field. This is a read only integer property.
SEPARATOR	This is a convenience property to easily use the separator string when making compressed alarms. This is a read-only string property.
clear()	Will clear all alarms and folders in the panel, leaving it empty.
refresh()	Reload the alarms that are currently in the panel.

`addEntry()`	This function takes up to four parameters: - The first is an object, which can be a representation of an alarm, an `Array` of alarms, or a `string` representing multiple alarms. If using a single alarm or a JavaScript `Array` of alarms, they can be either `GSMAlarm`s or alarm URI `string`s. (Please note that for text only entries this value can be `null` or empty string.) If using a `string` representing multiple alarms, the `string` must be composed of URI `string`s concatenated by the `SEPARATOR` character. - The second is the integer type of the entry to make. Use one of the following types: `STATUS_ALARM_TYPE`, `FOLDER_TYPE`, `FOLDER_AS_TEXT_TYPE`, `TEXT_ALARM_TYPE`, `TEXT_AND_STATUS_TYPE`, `FOLLOW_RUNTIME_TYPE`, or `TITLE_TEXT_TYPE` from the alarm panel's instance. - The third is the desired text the user wishes to see displayed. If `null` or `undefined`, the default name of the alarm will be used. - The fourth is whether the entry is to have the look of a header; passing `true` will produce this effect. The default is `false`. It will add the created entry to the panel.
`removeEntry()`	This function will remove an alarm from the panel. It takes a single argument that can either be a `GSMAlarm` or a URI `string`.
`dynamicAssignment()`	The purpose of this function is to provide the user with a quick way to easily have the alarm panel's entries point from one location to another. By reassigning the URI of the alarm descriptors of the status alarms (including compressed ones), text alarms, and folder entries. This function takes two `string` arguments. The first is a regular expression `string` and the second is a replacement `string`. If either argument is empty or `null` then this function does nothing. This function will traverse the entries within this panel's model and inspect the URI of each of the entries. Any URI that has a match for the regular expression `string` will have a new entry created with the matching regular expression `string` replaced with the specified replacement `string`. The attributes of the entries will be carried over (such as header style, display text, type, ...). If a match is not found when inspecting a URI then that entry remains unchanged. For example, text entries serving as titles will never be changed by this function, they are kept as is. Following is a table summarizing regular expression constructs that can be used with this method. Remember when creating a regular expression `string` to use double backslashes ("`\\`") for the reserved tokens and a single backslash ("`\`") for literal characters. Please see the examples

section for some details.

Summary of regular-expression constructs

Construct	**Matches**
Characters	
x	The character *x*
`\\`	The backslash character
`\0`*n*	The character with octal value `0`*n* (0 <= *n* <= 7)
`\0`*nn*	The character with octal value `0`*nn* (0 <= *n* <= 7)
`\0`*mnn*	The character with octal value `0`*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7)
`\x`*hh*	The character with hexadecimal value `0x`*hh*
`\u`*hhhh*	The character with hexadecimal value `0x`*hhhh*
`\t`	The tab character (`'\u0009'`)
`\n`	The newline (line feed) character (`'\u000A'`)
`\r`	The carriage-return character (`'\u000D'`)
`\f`	The form-feed character (`'\u000C'`)
`\a`	The alert (bell) character (`'\u0007'`)
`\e`	The escape character (`'\u001B'`)
`\c`*x*	The control character corresponding to *x*
Character classes	
`[abc]`	`a`, `b`, or `c` (simple class)
`[^abc]`	Any character except `a`, `b`, or `c` (negation)
`[a-zA-Z]`	`a` through `z` or `A` through `Z`, inclusive (range)
`[a-d[m-p]]`	`a` through `d`, or `m` through `p`: `[a-dm-p]` (union)
`[a-z&&[def]]`	`d`, `e`, or `f` (intersection)
`[a-z&&[^bc]]`	`a` through `z`, except for `b` and `c`: `[ad-z]` (subtraction)
`[a-z&&[^m-p]]`	`a` through `z`, and not `m` through `p`: `[a-lq-z]` (subtraction)
Predefined character classes	
`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`
POSIX character classes (US-ASCII only)	
`\p{Lower}`	A lower-case alphabetic character: `[a-z]`
`\p{Upper}`	An upper-case alphabetic character:`[A-Z]`

\p{ASCII}	All ASCII: [\x00-\x7F]
\p{Alpha}	An alphabetic character: [\p{Lower}\p{Upper}]
\p{Digit}	A decimal digit: [0-9]
\p{Alnum}	An alphanumeric character: [\p{Alpha}\p{Digit}]
\p{Punct}	Punctuation: One of !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph}	A visible character: [\p{Alnum}\p{Punct}]
\p{Print}	A printable character: [\p{Graph}]
\p{Blank}	A space or a tab: [\t]
\p{Cntrl}	A control character: [\x00-\x1F\x7F]
\p{XDigit}	A hexadecimal digit: [0-9a-fA-F]
\p{Space}	A whitespace character: [\t\n\x0B\f\r]
Classes for Unicode blocks and categories	
\p{InGreek}	A character in the Greek block (simple block)
\p{Lu}	An uppercase letter (simple category)
\p{Sc}	A currency symbol
\P{InGreek}	Any character except one in the Greek block (negation)
[\p{L}&&[^\p{Lu}]]	Any letter except an uppercase letter (subtraction)
Boundary matchers	
^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input
Greedy quantifiers	
X?	*X*, once or not at all
X*	*X*, zero or more times
X+	*X*, one or more times
X{n}	*X*, exactly *n* times
X{n,}	*X*, at least *n* times
X{n,m}	*X*, at least *n* but not more than *m* times
Reluctant quantifiers	
X??	*X*, once or not at all
X*?	*X*, zero or more times
X+?	*X*, one or more times
X{n}?	*X*, exactly *n* times
X{n,}?	*X*, at least *n* times
X{n,m}?	*X*, at least *n* but not more than *m* times

Possessive quantifiers	
`X?+`	*X*, once or not at all
`X*+`	*X*, zero or more times
`X++`	*X*, one or more times
`X{n}+`	*X*, exactly *n* times
`X{n,}+`	*X*, at least *n* times
`X{n,m}+`	*X*, at least *n* but not more than *m* times
Logical operators	
`XY`	*X* followed by *Y*
`X|Y`	Either *X* or *Y*
`(X)`	X, as a capturing group
Back references	
`\n`	Whatever the *n*th capturing group matched
Quotation	
`\`	Nothing, but quotes the following character
`\Q`	Nothing, but quotes all characters until `\E`
`\E`	Nothing, but ends quoting started by `\Q`
Special constructs (non-capturing)	
`(?:X)`	*X*, as a non-capturing group
`(?idmsux-idmsux)`	Nothing, but turns match flags on - off
`(?idmsux-idmsux:X)`	*X*, as a non-capturing group with the given flags on - off
`(?=X)`	*X*, via zero-width positive lookahead
`(?!X)`	*X*, via zero-width negative lookahead
`(?<=X)`	*X*, via zero-width positive lookbehind
`(?<!X)`	*X*, via zero-width negative lookbehind
`(?>X)`	*X*, as an independent, non-capturing group

Top Quick Reference Example

LogViewer

The LogViewer object represents an embedded log viewer (for events or incidents).

<table>
<tr><th>Property/Method</th><th>Description</th></tr>
<tr><td>window</td><td>References the top-level window that this object belongs to. See the Window object for more information.</td></tr>
<tr><td>document</td><td>References the Document that contains this object. In the case of nested documents, this always refers to the innermost document that immediately contains the object.</td></tr>
<tr><td>parent</td><td>References the zone that contains the object. See the Zone object for more information.</td></tr>
<tr><td>type</td><td>Read-only string property representing the log viewer's type, such as "event" for the event log viewer, and "incident" for the incident log viewer.</td></tr>
<tr><td>compactMode</td><td>Read/Write boolean property that gets/sets the "compact" mode on the log viewer. When in compact mode, only the results are displayed; the search fields and the toolbar are hidden.</td></tr>
<tr><td>results</td><td>Read-only array property representing the log entries that are currently displayed in this log viewer. Each entry has the following properties:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>id</td><td>Read-only integer property representing the entry's ID.</td></tr>
<tr><td>uri</td><td>Read-only string property representing the entry's URI.</td></tr>
</table>
</td></tr>
<tr><td>selectedEntry</td><td>Read-only property representing the currently selected entry in this log viewer. Refer to results for the description of the entry's properties.</td></tr>
<tr><td>search()</td><td>This function performs a search based on the given criteria. It takes a single parameter of type Object providing the properties that specify the criteria. The tables below describe the supported properties for each type of log entry:
<table>
<tr><th colspan="2">Event Log Entries</th></tr>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>id</td><td>Unique ID of the entry. All other criteria will be ignored if this criterion is used.</td></tr>
<tr><td>minTimestamp</td><td>Minimum time value for the timestamp.</td></tr>
<tr><td>maxTimestamp</td><td>Maximum time value for the timestamp.</td></tr>
<tr><td>alarmURI</td><td>Text value to be matched against the alarm URI.</td></tr>
<tr><td>alarmName</td><td>Text value to be matched against the alarm name.</td></tr>
</table>
</td></tr>
</table>

`deviceURI`	Text value to be matched against the device URI.
`deviceClass`	Text value to be matched against the device class.
`user`	Text value to be matched against the user name.
`path`	Text value to be matched against the path.
`text`	Text value to be matched against the alarm text.
`extraInfo`	Text value to be matched against the extra information including the text.
`type`	The event type. Accepts one of the possible values for `GSMAlarm.type`.
`oldState`	The previous state. Accepts one of the possible values for `GSMAlarm.status`.
`newState`	The new state. Accepts one of the possible values for `GSMAlarm.status`.
`oldLatch`	The previous latch value. Accepts one of the possible values for `GSMAlarm.status`.
`newLatch`	The new latch value. Accepts one of the possible values for `GSMAlarm.status`.
`oldAck`	The previous acknowledgment value. Accepts one of the possible values for `GSMAlarm.status`.
`newAck`	The previous acknowledgment value. Accepts one of the possible values for `GSMAlarm.status`.
`parentAlarm`	The exact URI of a virtual alarm. The search will retrieve log entries about a virtual alarm and any of its sub-alarms.

Incident Log Entries	
Property	**Description**
`id`	Unique ID of the entry. All other criteria will be ignored if this criterion is specifed.
`name`	Text value to be matched against the name.
`trigger`	Text value to be matched against the trigger URI.
`startTimeMin`	Minimum value for the start time.
`startTimeMax`	Maximum value for the start time.
`acknowledged`	Boolean value that specifies the "acknowledged" state.
`ackTimeMin`	Minimum value for the last time the incident was acknowledged (if applicable).
`ackTimeMax`	Maximum value for the last time the incident was acknowledged (if applicable).
`cleared`	Boolean value that specifies the "cleared" state.
`clearTimeMin`	Minimum value for the last time the incident was cleared (if applicable).
`clearTimeMax`	Maximum value for the last time the incident was cleared (if applicable).
`escalations`	Minimum number of escalations.
`subsIncluded`	Boolean value that specifies whether or not sub-incidents will be included (the default is `false`).

	Notes: - The specified criteria will be grouped by conjunction (`AND`). - A criterion will be ignored if corresponding property is unspecified. The search will match all entries if none of the properties is specified. - Time-based values can be expressed as an elapsed time or as an absolute time value. Examples: `"12 hours ago", "2006-12-30 16:34:29.453"` - Text values are case-insensitive and the '`%`' character can be used as a wildcard to match an arbitrary sequence of text. Use '`\%`' to make it not be interpreted as a wildcard character. Examples: `"overall", "virtualAlarm://%Audio%Overall", "%Audio\%20Overall"`
`onSelect()`	An event handler that can be attached to this object in order to be notified about **selection** changes. There is a single parameter representing the log entry that has been selected. See the log entry's properties in the description of `results`.
`onUpdate()`	An event handler that can be attached to this object in order to be notified about changes on the **results**, e.g. entries are added/removed/updated. There are two parameters representing the "old value" and "new value" of the log entry that has been affected. See the log entry's properties in the description of `results`.

Top Quick Reference

Zone

The Zone object represents an area within an iControlWeb page whose content can be dynamically changed. Each document can contain multiple zones.

Property/Method	Description
parent	References the document that contains the zone. See the Document object for more information.
window	References the top-level window that this zone belongs to. See the Window object for more information.
content	References the content of this zone. If the content object is not script-accessible then the value will be null.
document	If the contents of this Zone is a page then this property will reference the page, otherwise it will be undefined.
x	x-coordinate of the left side of the zone, with respect to the left side of the enclosing Document. Positive values are to the right. This is a read/write property (note: if you intend to modify both x and y, the moveTo() method will be more efficient).
y	y-coordinate of the top of the zone, with respect to the top of the enclosing Document. Positive values are downwards. This is a read/write property (note: if you intend to modify both x and y, the moveTo() method will be more efficient).
screenX	x-coordinate of the left side of the zone, with respect to the left side of the screen. Positive values are to the right. This is a read-only property.
screenY	y-coordinate of the top of the zone, with respect to the top of the screen. Positive values are downwards. This is a read-only property.
width	The width of the zone, in pixels. This is a read/write property (note: if you intend to modify both width and height, the resizeTo() method will be more efficient).
height	The height of the zone, in pixels. This is a read/write property (note: if you intend to modify both width and height, the resizeTo() method will be more efficient).
border	This property specifies an optional border around the zone. The value must be a Java object of type javax.swing.border.Border. For instance, to obtain a single-pixel red border around a zone, use: `zone.border = Packages.javax.swing.BorderFactory.createLineBorder(java.awt.Color.RED);` The default value is null and specifies that no border should be used. This is a read/write property.
view	Refers to the view of the Zone.
visible	This read/write boolean property determines whether the zone (and its contents) are currently visible.
loadDocument()	A utility method that creates a Document object and uses it as the content of the zone. The method takes a single URL string as a parameter. The URL may be relative to the parent document or fully specified. This method may throw an exception that can be caught if the page is not found. *Since* 3.30, this method returns the newly created Document.

`loadWidget()`	A utility method that creates a `Widget` object and uses it as the content of the zone. The method takes a single URL `string` as a parameter. The URL may be relative to the parent document or fully specified. A relative URL must be something like `"folder/widgetName.mwf"`, and a fully specified URL must have the full path to the desired widget. This method may throw an exception that can be caught if the page is not found. *Since 3.30*, this method returns the newly created `Widget`.
`moveTo()`	This method takes two integer parameters specifying the x and y coordinates to which the zone should be moved relative to its containing document. *Since 3.30*, this method also returns the Zone itself for convenience, so multiple calls can be chained easily.
`moveBy()`	This method takes two integer parameters specifying the adjustments in x and y coordinates by which the zone should be moved. *Since 3.30*, this method also returns the Zone itself for convenience, so multiple calls can be chained easily.
`resizeTo()`	This method takes two integer parameters specifying the new width and height for the zone. *Since 3.30*, this method also returns the Zone itself for convenience, so multiple calls can be chained easily.
`resizeBy()`	This method takes two integer parameters specifying the adjustments in width and height by which the zone should be resized. *Since 3.30*, this method also returns the Zone itself for convenience, so multiple calls can be chained easily.
`showAlarmBrowser()`	Sets the contents of the zone to a multi GSM alarm browser. This method also can take a `String` as an argument, which is the URI of a GSM alarm. If the alarm exists within a GSM it will select that GSM and expand to that alarm.
`showLogViewer()`	A utility method that creates a `GSMLogViewerModel` object and uses it as the content of the zone. This method takes a `string` parameter representing the GSM look-up server address. It can also take a second (optional) `boolean` parameter which, when set to `true`, will disable the `Delete` button in the user interface, preventing users from deleting data from the logs. The default when not specified is to leave the button enabled.
`showContextualLogViewer()`	A utility method that creates a `GSMLogViewerModel` object and uses it as the content of the zone. This method takes a `string` parameter representing the device URI (or MirandaLongID) for the device to monitor. It can also take a second (optional) `boolean` parameter which, when set to `true`, will disable the `Delete` button in the user interface, preventing users from deleting data from the logs. The default when not specified is to leave the button enabled.
`showIncidentViewer()`	A utility method that creates a `GSMIncidentViewerModel` object and uses it as the content of the zone. This method takes a `string` parameter representing the GSM look-up server address. It can also take a second (optional) `boolean` parameter which, when set to `true`, will disable the `Delete` button in the user interface, preventing users from deleting data from the logs. The default when not specified is to leave the button enabled.
`showServicePanel()`	A utility method that creates a `ServicePanel` object and uses it as the content of the zone. This method takes a single string parameter representing the MirandaLongID for the device whose panel to show.
`showINavigator()`	A utility method that sets the content of the zone to an instance of iControl Navigator.
`showVNCViewer()`	A utility method that creates a Vnc viewer object and uses it as the content of the zone. This method takes up to three parameters: the host name or IP address of the Vnc server, as a String, the port name as an integer, and (optionally) the password to be used to log in to the VNC server. The port name is usually 5900 + the display number, with the display number starting at 0. If only two parameters are used (i.e. no password is specified), then the user will be prompted to enter a password at connection time.

`showWebBrowser()`	A utility method that creates a Web Browser object and uses it as the content of the zone. This method takes two `String` parameters: the web page URL and the toolbar type (which can be one of `NO_NAVIGATION_TOOLBAR`, `LIMITED_NAVIGATION_TOOLBAR`, `FULL_NAVIGATION_TOOLBAR`).
`showMIBBrowser()`	A utility method that creates a MIB Browser object and uses it as the content of the zone. This method takes four parameters: 1. IP address or host name of the target device : string 2. port number of the target device : integer 3. class or model name of the target device : string 4. OID or variable name to look up : string Example: `myZone.showMIBBrowser("10.4.5.25", 161, "Kaleido K2", ".1.3.6.1.4.1.3872.9.5.1");`
`NO_NAVIGATION_TOOLBAR`	A possible value for a WebBrowser toolbar. Hiding the toolbar.
`LIMITED_NAVIGATION_TOOLBAR`	A possible value for a WebBrowser toolbar. Showing the navigation button.
`FULL_NAVIGATION_TOOLBAR`	A possible value for a WebBrowser toolbar. Showing a complete navigation toolbar.

Top Quick Reference

Gateway

The `Gateway` object represents an iControl XML Gateway, which is used to communicate with remote gateway-enabled processes using an XML-based protocol.

Property/Method	Description
constructor	The constructor takes up to two arguments: a `string` representing the host name or IP address of the remote gateway you wish to control, and an optional port number as a `number` (the default port is 14000).
`sendCommand()`	Sends an XML command to the gateway (as a `string`). An exception will be thrown if the gateway sends a NACK reply.

Top Quick Reference

KaleidoAlto

The `KaleidoAlto` object is used to communicate with a Kaleido Alto or Quad multi-image display device from Miranda. The user is responsible for creating instances of this class, using the constructor explicitly. The constructor requires a single argument string, representing the host name or IP address of the Kaleido Alto or Quad you wish to talk to.

Property/Method	Description
constructor	The `KaleidoAlto()` constructor is used to specify the host name or IP address of the remote Alto unit to control. It takes a single `string` parameter representing this address.
`audioOut`	A read/write property which can take three types of values: `null` To indicate that no audio source is currently being routed to the audio output on the Kaleido. a number To indicate which audio card input is routed to the audio output on the Kaleido. an object To indicate that an embedded audio is routed to the audio output on the Kaleido. The object has three numeric properties named `input`, `group` (from 1 to 4) and `aes` (either 1 or 2) to indicate which embedded audio is used. *Note that this requires version 3.01 or above of the Alto software.*
`audioOutMode`	A read/write property specifying the current audio output mode. Valid values are `NORMAL`, `MUTE` and `-20dB`.
`audioOutVolume`	The audio monitoring volume, as a read/write numeric property ranging between -90dB and 0dB.
`layout`	This is the name of the layout currently being displayed on the Kaleido, without the layout file extension. It is a read/write `string` value which can be used to load a different layout on the Kaleido. This replaces the old `loadLayout()` method, which remains for backwards compatibility.
`layouts[]`	The (read-only) list of all layouts available on the Alto, as an `Array` of `string`s (the names of the layouts without the extensions).
`verticalOffset`	This write-only property specifies the number of lines (between 0 and 175) to offset the graphics vertically on the display.
`closeID()`	Closes the connection with the Alto gateway. It takes a single `string` argument representing the IP address of the Alto.

`fireAction()`	This method is used to fire any programmed actions on the Kaleido. It takes a single `string` argument representing the action name saved on the Kaleido.
`ftpLayout()`	The `ftpLayout()` method is used to change the layout being displayed on the Alto. It takes a single `string` parameter representing the name of the layout as defined on the Alto. This uses the FTP interface on the Alto (not the gateway). The other methods all use the gateway interface.
`openID()`	Opens the connection with the Alto gateway. It takes a single `string` argument representing the IP address of the Alto.
`resetCountdownTimer()`	This method is used to reset a countdown timer on the Kaleido display to its original state. It takes a single `string` argument, representing the ID of the countdown timer to reset.
`saveLayout()`	This method saves the current layout being displayed to disk. It takes a single `string` parameter representing the name of the saved layout (without an extension).
`setAssignment()`	This method is used to assign a channel to a monitor on the Kaleido display. It takes two `string` arguments representing the channel name and the monitor name.
`setCountdownTimer()`	This method is used to configure countdown timers on the Kaleido display. It takes four arguments representing: the timer ID (as a `string`), the preset time (a `string` in the `"HH:MM:SS"` format), the direction (a `boolean` value of `true` for `"DOWN"` or `false` for `"UP"`), and the loop (a `boolean` value of `true` for `"ON"` or `false` for `"OFF"`).
`setStatus()`	This method is used to set any status component on the Kaleido display. It takes three arguments representing the ID of the status component (as a `string`), the status (`"OK"`, `"DISABLE"`, `"WARNING"`, `"ERROR"`, or one of the `GSMAlarm` status constants) and a status message (as a `string`).
`setText()`	This method is used to set any dynamic text component on the Kaleido display. It takes two `string` arguments representing the address or ID of the text component and the value to be set.
`startCountdownTimer()`	This method is used to start a countdown timer on the Kaleido display. It takes a single `string` argument, representing the ID of the countdown timer to start.
`stopCountdownTimer()`	This method is used to stop countdown timers on the Kaleido display. It takes a single `string` argument, representing the ID of the countdown timer to stop.

Top Quick Reference Example

KaleidoK2

The `KaleidoK2` object is used to communicate with a Kaleido K2 multi-image display device from Miranda. The user is responsible for creating instances of this class, using the constructor explicitly. The constructor requires a single argument `string`, representing the host name or IP address of the Kaleido K2 you wish to talk to.

Property/Method	Description
constructor	The `KaleidoK2()` constructor is used to specify the host name or IP address of the remote K2 unit to control. It takes a single `string` parameter representing this address.
`audioOut`	A read/write property which can take four types of values: `null` To indicate that no audio source is currently being routed to the audio output on the Kaleido. a number To indicate which audio card input is routed to the audio output on the Kaleido. an object with `input`, `group` and `aes` properties To indicate that an embedded audio is routed to the audio output on the Kaleido. The object has three numeric properties named `input`, `group` (from 1 to 4) and `aes` (either 1 or 2) to indicate which embedded audio is used. an object with `host` and `feed` properties To indicate that an audio stream on the network is routed to the audio output on the Kaleido. The object has two `string` properties named `host` (an IP address) and `feed` (the ID of the feed to use from the specified host).
`audioOutMode`	A read/write property specifying the current audio output mode. Valid values are `NORMAL`, `MUTE` and `-20dB`.
`audioOutVolume`	The audio monitoring volume, as a read/write numeric property ranging between -90dB and 0dB.
`layout`	This is the name of the layout currently being displayed on the K2, without the layout file extension (`.kg2`). It is a read/write `string` value which can be used to load a different layout on the K2. This replaces the old `getLayout()` and `loadLayout()` methods, which remain for backwards compatibility.

layouts[]	This Array contains the list of all layouts available on the K2, each one represented as a string object containing the name of the layout without its extension. This is also accessible using the getLayouts() method for backwards compatibility.
openID()	This method is used to open the connection with the K2 gateway. It takes one single string argument representing the IP address of the K2.
closeID()	This method is used to close the connection with the K2 gateway. It takes a single string argument representing the IP address of the K2.

getMetadata()	This method takes a single parameter, the name (as a string) of a multidata component configured with metadata (for instance XDS data) on the K2. It returns an XML object representing the metadata published by the component, as documented in the K2 gateway manual. The object can be accessed using the E4X syntax. For example, to read the network name of the station on input #5, you can write:
	`var metadata = k2.getMetadata(`*`dataName`*`);`
	`var networkName = metadata.input.(@id==5).@networkName;`

saveLayout()	This method saves the current layout being displayed on disk. It takes a single string parameter representing the name of the saved layout (no extension).
setText()	This method is used to set any dynamic text component on the K2 display. It takes two string arguments representing the address or ID of the text component and the value to be set.
setStatus()	This method is used to set any status component on the K2 display. It takes three arguments representing the ID of the status component (as a string), the status ("OK", "DISABLE", "WARNING", "ERROR", or one of the GSMAlarm status constants) and a status message (as a string).
setAssignment()	This method is used to assign a channel to a monitor on the K2 display. It takes two string arguments representing the channel name and the monitor name.
setCountdownTimer()	This method is used to configure countdown timers on the K2 display. It takes four arguments representing the timer ID (as a string), the preset time (a string in the "HH:MM:SS" format), the direction (a boolean value of true for "DOWN" or false for "UP"), and the loop (a boolean value of true for "ON" or false for "OFF").
resetCountdownTimer()	This method is used to reset a countdown timer on the K2 display to its original state. It takes a single string argument, representing the ID of the countdown timer to reset.

`startCountdownTimer()`	This method is used to start a countdown timer on the K2 display. It takes a single `string` argument, representing the ID of the countdown timer to start.
`stopCountdownTimer()`	This method is used to stop countdown timers on the K2 display. It takes a single `string` argument, representing the ID of the countdown timer to stop.
`fireAction()`	This method is used to fire any programmed actions on the K2. It takes a single `string` argument representing the action name saved on the K2.
`enableAlarmGroup()`	This method is used to enable any alarm group programmed on the K2. It takes a single `string` argument representing the alarm group name saved on the K2.
`disableAlarmGroup()`	This method is used to disable any alarm group programmed on the K2. It takes a single `string` argument representing the alarm group name saved on the K2.

Top Quick Reference

KaleidoX

The KaleidoX object is used to communicate with a Kaleido X multi-image display device from Miranda. The user is responsible for creating instances of this class, using the constructor explicitly. The constructor requires a single argument string, representing the host name or IP address of the Kaleido X you wish to talk to.

Property/Method	Description
constructor	The KaleidoX() constructor is used to specify the host name or IP address of the remote Kaleido X unit to control. It takes a single string parameter representing this address.
room	This is the name of the room currently selected, or null when in the global context.
rooms[]	This Array contains the list of all rooms available on the KX, each one represented as a string object containing the name of the room.
audioOut	A read/write property which can take four types of values: null To indicate that no audio source is currently being routed to the audio output on the Kaleido. a number To indicate which audio card input is routed to the audio output on the Kaleido. an object with input, group and aes properties To indicate that an embedded audio is routed to the audio output on the Kaleido. The object has three numeric properties named input, group (from 1 to 4) and aes (either 1 or 2) to indicate which embedded audio is used. an object with host and feed properties To indicate that an audio stream on the network is routed to the audio output on the Kaleido. The object has two string properties named host (an IP address) and feed (the ID of the feed to use from the specified host).
layout	This is the name of the layout currently being displayed for the current room on the KX, without the layout file extension (.kg2). Accordingly, it will only work in the context of a specific room, i.e. when room is not null. It is a read/write string value which can also be used to load a different layout for the current on the KX.
layouts[]	This Array contains the list of all layouts available for the current room on the KX, each one represented as a string object containing the name of the layout without its extension. When in the global context, this will list all layouts in all rooms using the following format: ***room/layout***.
open()	This method is used to open the connection with the KX gateway whose IP address was specified in the constructor.
closeID()	This method is used to close the connection with the KX gateway.

`setText()`	This method is used to set any dynamic text component on the KX display. It takes two `string` arguments representing the address or ID of the text component and the value to be set.
`setStatus()`	This method is used to set any status component on the KX display. It takes three arguments representing the ID of the status component (as a `string`), the status (`"OK"`, `"DISABLE"`, `"WARNING"`, `"ERROR"`, or one of the `GSMAlarm` status constants) and a status message (as a `string`).
`setAssignment()`	This method is used to assign a channel to a monitor on the KX display. It only works in the context of a specific room, i.e. when `room` is not `null`. It takes two `string` arguments representing the full path to the channel (ex.: `/Input A/Channel 1`) and the monitor name (which is available in the XEdit layout under `Properties | Assignments | Name`).

Top Quick Reference

RCP100

The **RCP100** object is used to communicate by an iControl RCP-100 control panel from Miranda. The user is responsible for creating instances of this class, using the constructor explicitly. Note that the RCP-100 unit must be placed in *Router Follow* mode for this to work.

Property/Method	Description
constructor	The **RCP100()** constructor is used to specify the host name or IP address of the remote RCP-100 unit to control. It takes a single string parameter representating this address.
close()	The **close()** method is used to place the RCP-100 in a mode where no card is selected and it is awaiting another command. It will normally show **NO CARD ASSIGNED** on the LCD.
quit()	The **quit()** method is used to release resources (sockets) that are used by this object, when it is not required anymore.
show()	The **show()** method is used to change the card currently under control in the RCP-100. It takes a single string parameter representing the **longID** of the card to load.
showGroup()	The **showGroup()** method is used to change the set of cards currently under control in the RCP-100. The first argument to this method is an **Array** of **longID**s (as **String**s) of the cards to load for selection. If a second **String** argument is added to the method, it will be the **longID** of the card that should be pre-selected from the set defined in the first argument.

Top Quick Reference

Router

The Router represents an A/V router (or switcher) that has been discovered on the network.

Property/Method	Description
name	A read-only string property containing the router name.
type	A read-only string property describing the type of router.
numLevels	A read-only number property indicating the number of levels defined for the router.
numInputs	A read-only number property indicating the number of inputs/sources defined for the router.
numOutputs	A read-only number property indicating the number of outputs/ destinations defined for the router.
service	Provides direct access to the router service.
connectCrosspoint()	This method switches a single router cross point. It takes three number parameters: the level, the source and the destination to connect (as numbers).
getConnectedSource()	This method returns (as a number) the currently connected source for a given combination of level and destination (the two number parameters to this method).
getSourceLabel()	This method returns (as a string) the label of the specified source on a given level. It takes two number parameters: the level index and the source index.
getSourceIndex()	This method returns (as a int) the one-based index of the specified source label on a given level. It returns -1 if the label is not found. It takes two parameters: the level index and the source label.
getDestinationLabel()	This method returns (as a string) the label of the specified destination on a given level. It takes two number parameters: the level index and the destination index.
getDestinationIndex()	This method returns (as a int) the one-based index of the specified destination on a given level. It returns -1 if the label is not found. It takes two parameters: the level index and the destination label.
lockCrosspoints()	Locks a router destination at a given level. It takes three parameters: the level and the destination (as numbers) and the lock message.
unlockCrosspoints()	Unlocks a router destination at a given level. It takes two parameters: the level and the destination.

`isLocked()`	Tells whether a router destination at a given level is locked. It takes two numeric parameters: the level and the destination.
`takeSalvo()`	Execute salvo. It takes only one parameter: the salvo name as defined in the salvo editor. If at least one crosspoint cannot be taken, none is taken.

Top Quick Reference

WorkBook

The `WorkBook` object is used to represents an Excel WorkBook. Any of the spreadsheets can be accessed as an array element with the spreadsheet index as an array index.

Property/Method	Description
constructor	The `WorkBook()` constructor is used to build a parser on a given Excel `.xls` file. It takes a single string parameter as an URL representation.
`sheets[]`	Array of spreadsheets (corresponding to tabs in Excel), which can be indexed either by the 0-based integer tab index, or by the tab name.
`url`	References the url of the currently loaded workbook.

Spreadsheet

The `Spreadsheet` object is used to represents an Excel spreadsheet. Any of the rows can be accessed as an array element of the `Spreadsheet` object with the row index as an array index. Note that when accessed that way (it is recommended to use the `rows[]` array instead), the array index is obtained by substracting 2 from the row label shown in Excel.

Property/Method	Description
`name`	A read-only `string` referencing the name of the tab containing the spreadsheet.
`rows[]`	A read-only `array` referencing the rows defined in the spreadsheet, as `Row` objects. The array indices are numeric and correspond exactly to the row labels in Excel (1 to 65536).

Row

The `Row` object is used to represents a row in an Excel sheet. Any of the cells can be accessed as an array element with the cell index as an array index. It is usually easier to reference the cells using the `cells` array, however.

Property/Method	Description
`cells[]`	A read-only `array` referencing the cells defined in the row, as `Cell` objects. The array indices are strings and correspond exactly to the column labels in Excel (`A` to `IV`).

Cell

The `Cell` object is used to represents an Excel cell.

Property/Method	Description
`value`	References the cell value.

Top Quick Reference Example

GSMAlarm

The `GSMAlarm` component provides introspection into an alarm published on GSM.

Property/Method	Description
`deviceClass`	A `String` that identifies the type of device this alarm is attached to. This is a read-only property.
`deviceURI`	A `String` that identifies the URI of the device to which this alarm belongs. The device URI serves as a unique identifier for a device, and all alarms that come from the same device have the same URI. This is a read-only property.
`name`	The human-readable representation of the alarm, as a `String`.
`pathString`	The path of the alarm in the GSM alarm tree, as a `/`-separated `String`.
`type`	The type of the alarm, as a combination of flags from `TYPE_STATUS`, `TYPE_TEXT` and `TYPE_NOT_LOGGED` (read-only).
`status`	The current status of the alarm.
`previousStatus`	The previous status of the alarm.
`latchedStatus`	Read-only dynamic property referencing the current latched status of the alarm in GSM.
`previousLatchedStatus`	Read-only dynamic property referencing the previous latched status of the alarm in GSM.
`acknowledgementStatus`	Read-only dynamic property referencing the current acknowledgement status of the alarm in GSM.
`previousAcknowledgementStatus`	Read-only dynamic property referencing the previous acknowledgement status of the alarm in GSM.

`operationalMode`	Read-only dynamic property referencing the current operational mode(s) of the alarm in GSM (`OFFLINE` or `MAINTENANCE` or a combination of both).
	This property is a bit-field, so to check if a given operational mode (say, `OFFLINE`) is set, do something like:
	`if (alarm.operationalMode & GSMAlarm.OFFLINE)`
	`{ // Deal with offline alarm here... }`

`previousOperationalMode`	Read-only dynamic property referencing the previous operational mode(s) of the alarm in GSM.
`text`	The current text of the alarm, for text alarms.
`uri`	The URI (unique identifier) of the alarm.
`timestamp`	A read-only `Date` property indicating the time at which the alarm occured, as determined by the alarm publisher.
`timecode`	A read-only `string` property indicating the timecode at which the alarm occured, when known. The returned `string` is in the form: `"hh:mm:ss:ff"`. The value of the property might be `null` when no timecode information is available.
`NORMAL`	A possible value for `status`.
`MINOR`	A possible value for `status`.
`MAJOR`	A possible value for `status`.
`CRITICAL`	A possible value for `status`.
`UNKNOWN`	A possible value for `status`.
`DISABLED`	A possible value for `status`.
`NON_EXISTENT`	A possible value for `status`.
`PENDING`	A possible value for `status`. This is a not a GSM status, it's just a client status to indicate that we have not received the GSM status yet. All iControl Web alarms are initiated with the `PENDING` status. If a GSM service is stopped, then all iControl Web alarms activated on this GSM will have their status reset to `PENDING`.
`NO_MODE`	The value of `operationalMode` when no mode is set.
`OFFLINE`	A possible value for `operationalMode`.
`MAINTENANCE`	A possible value for `operationalMode`.
`TYPE_STATUS`	A possible value for `type`.
`TYPE_EVENT`	A possible value for `type`.
`TYPE_TEXT`	A possible value for `type`.
`TYPE_FOLDER`	A possible value for `type`.
`resetLatch()`	This method resets the latch on the alarm in GSM.
`acknowledge()`	This method acknowledges the alarm in GSM.
`setOperationalMode()`	This method sets or unsets the specified operational mode on the alarm in GSM. It takes two parameters of types `int` and `boolean`. The first parameter specifies the mode, and the second specifies whether to set or unset (true or false). Note: `NO_MODE` cannot be used.

Top Quick Reference

TreeTable

The `TreeTable` component is a cross between a tree and a table. You can see it as a table which contains a tree in its first column, with expandable/collapsable nodes. It can be used as a simple tree or a simple table when required.

Property/Method	Description
constructor	You use the constructor explicitly to create instances of this object, which is always placed within an existing zone. The constructor takes two parameters: the zone where the component should be placed (a `Zone` object), and an optional model object for the table. You can get a suitable model object from the following locations (the list may not be exhaustive): • `GSM.getTreeTableModel()` The component will immediately replace the current contents of the zone, if any.
`columns[]`	A read/write `Array` property referencing the columns displayed in the tree table. The `Array` (currently) contains `String`s referencing the name of the GSM alarms that should be displayed as columns in the table. Note that currently the tree itself is always the first column and needs to be included.
`maximumDepth`	A read/write `number` property referencing the maximum tree depth to display (experimental).
`rootPath[]`	A read/write `Array` property referencing the path of the GSM folder to use as the tree root (this is the same property as specified as the third parameter of the constructor). Each path element is a `string` referencing a GSM folder, including the root GSM folder `"iControl alarms"`.
`selectedColumn`	A read-only `number` property referencing the index of the currently selected cell's column (0-based).
`selectedRow`	A read-only `number` property referencing the index of the currently selected cell's row (0-based).
`selectedValue`	A read-only property referencing the contents of the currently selected cell.
`expandAll()`	Expands all tree nodes in the tree table.
`expandRow(row)`	Expands the selected table row in the tree. It takes a single `number` parameter representing the (0-based) row index to expand, as currently seen on screen.
`hideColumn()`	Hides a column from the current view. It takes a single `number` parameter representing the (0-based) column index to remove, as currently seen on screen.

`onClick()`	An optional event handler that will be called when the user clicks somewhere in the table. It takes a `MouseEvent` argument.
`valueAt()`	This method takes two parameters, specifying the row and column of the cell whose value will be returned.

Top Quick Reference

PopupMenu

Allows the creation of a script-controlled pop-up menu.

Note that each action menu item can have additional fields (aside from **`label`** and **`action`**) that will be accessible in the **`action`** function as fields of **`this`** (which refers to the menu item in this case).

Property/ Method	Description
constructor	The constructor can take up to two parameters. The first one is mandatory and it represents the structure of the menu. The structure is defined as an **`Array`** of menu items. There are currently three types of menu items available: action, sub-menu and separator. For action items, you need to specify a **`string`** label and an action **`Function`** properties. For sub-menu items, you need to specify a **`string`** label and a menu (as an **`Array`**, another structure) properties. For separators, you only need to specify a **`boolean`** separator property set to **`true`**. Note that for action and sub-menu items, the label **`string`** can actually contain HTML mark-up, in which case it *must* being with the string **`"<html>"`**. This can be used to change the label's font color or style or even to add an image to the menu item, for example. The second parameter of the constructor is optional and it represents the maximum number of rows the popup menu should display per column. When this number is reached, the popup menu with automatically create another column. That's it; use this parameter to create multiple column pop-up menus. You can also specify an optional numeric rows property when defining a sub-menu item to reproduce this multiple columns behavior at a sub-menu level. Here is an example on how to create a popup menu:

```
var menuStructure = new Array();

var action1 = new Object();
action1.label = "Do this";
action1.action = function() {
   window.alert("Did this! -- " + this.label);
};

var action2 = new Object();
action2.label = "Do that";
action2.action = function() {
   window.alert("Did that! -- " + this.label);
};

var sep = new Object();
sep.separator = true;

var subMenu = new Object();
subMenu.label = "Sub-menu"
```

```
var subMenuStructure = new Array();

var subAction1 = new Object();
subAction1.label = "Do this";
subAction1.action = function() {
   window.alert("Did this from sub-menu! -- " + this.label);
};

var subAction2 = new Object();
subAction2.label = "Do that";
subAction2.action = function() {
   window.alert("Did that from sub-menu! -- " + this.label);
};

subMenuStructure.push(subAction1);
subMenuStructure.push(subAction2);

subMenu.menu = subMenuStructure;

menuStructure.push(action1);
menuStructure.push(action2);
menuStructure.push(sep);
menuStructure.push(subMenu);

var popup = new PopupMenu(menuStructure);
```

You can also use a shorthand notation such as the following to initialize the menu:

```
var popup = new PopupMenu(
   [
      {
         label: "Do this",
         action: function() {
            window.alert("Did this! -- " + this.label);
         }
      },
      {
         label: "Do that",
         action: function() {
            window.alert("Did that! -- " + this.label);
         }
      },
      { separator: true },
      {
         label: "Sub-menu",
         menu: [
           {
             label: "Do this",
             action: function() {
                window.alert("Did this from sub-menu! -- " + this.label);
             }
           },
           {
              label: "Do that",
              action: function() {
                 window.alert("Did that from sub-menu! -- " + this.label);
              }
```

<table>
<tr><td></td><td><pre>
              }
           ]
         }
      ]
   );
</pre></td></tr>
<tr><td><code>show(x,y)</code></td><td>This method actually shows the pop-up menu. It takes two arguments which specify the <code>x</code> and <code>y</code> coordinates where the menu should be shown, with respect to the root document of the window.</td></tr>
</table>

Top Quick Reference

MouseEvent

The `MouseEvent` object is a standard W3C DOM level 2 event used as a parameter to event handlers like `onClick()`.

Property/Method	Description
`screenX`	The `screenX` property of the `MouseEvent` object controls the horizontal (x-coordinate) positioning within the computer screen in which the event occurred.
`screenY`	The `screenY` property of the `MouseEvent` object controls the vertical (y-coordinate) positioning within the computer screen in which the event occurred.
`clientX`	The `clientX` property of the `MouseEvent` object controls the horizontal (x-coordinate) positioning within the Web page in which the event occurred.
`clientY`	The `clientY` property of the `MouseEvent` object controls the vertical (y-coordinate) positioning within the Web page in which the event occurred.
`ctrlKey`	`true` if the Ctrl key was pressed while the mouse event occurred, `false` otherwise.
`shiftKey`	`true` if the Shift key was pressed while the mouse event occurred, `false` otherwise.
`altKey`	`true` if the Alt key was pressed while the mouse event occurred, `false` otherwise.
`metaKey`	`true` if the Meta key was pressed while the mouse event occurred, `false` otherwise.
`button`	Refers to which mouse button was pressed or clicked. The value returned is 1, 2, or 3, representing the left, middle, and right mouse buttons, respectively.
`detail`	Specifies the event detail. For `click` events, this field records the click count.

Top Quick Reference Example

AsiControl

The `AsiControl` object provides you with the ability to communicate with the ASI service. With this object you can get access custom TreeTableModels that provide specific data produced by the ASI service. This TreeTable models are pluged into the `TreeTable`

Property/Method	Description
`setClient(hostname, name)`	The `setClient` method is used to bind the AsiControl proxy of an ASI service using the `hostname` of the appserver and the `name` of the service (short id).
`service`	The `service` property is used to get/set the service proxy of an ASI service. To get this service proxy you need to use the `Navigator` object to get the service object using its long id.
`longId`	The `longId` property is used to get/set the longID of an ASI service. This value must be the same as the value used to get the service object.
`getIoRateTableModel(io)`	The `getIoRateTableModel` method returns a TreeTableModel that provides a view that shows that total data rate of each stream present in the ASI stream. The io parameter can have a value between 1 to 4.
`getOtherRateTableModel(io)`	The `getOtherRateTableModel` method returns a TreeTableModel that provides a view that shows data rate of all combined streams, null packets, and the total data rate of the ASI stream. The io parameter can have a value between 1 to 4.
`getTSTreeTableModel(io)`	The `getTSTreeTableModel` method returns a TreeTableModel that contains tree representation of the transport stream of the selected ASI io. The io parameter can have a value between 1 to 4.
`getEPGTableModel(io, stream)`	The `getEPGTableModel` method returns a TreeTableModel of the electronic program guide of a specific stream in the selected ASI io. The io parameter can have a value between 1 to 4. The stream parameter must have the following format "Stream #" where # is replaced with the number of the desired stream as seen in GSM. It is not the stream id (sid).
`streamingOn(io, sid, bitRate)`	The `streamingOn` method is used to start the Windows Media streaming from the ASI probe. This method returns the URL that is used to initialize the media player (or IE) so the stream can be viewed. The ASI probe only supports one stream at a time. The io parameter can have a value between 1 to 4. The sid must be valid for the selected io. The bitRate can have maximum of value of 450 (currently ignored, only the default configuration of the ASI probe is used).
`streamingOff(io, sid)`	The `streamingOff` method must be called before starting a new stream. The io parameter can have a value between 1 to 4. The sid must be valid for the selected io.
`thumbnailOn(io, sid, secPerFrame)`	The `thumbnailOn` method is used to turn on thumbnailing of a stream. The io parameter can have a value between 1 to 4. The sid must be valid for the selected io. The secPerFrame parameter is used to configure how many seconds between frames.

thumbnailOff(io, sid)	The **thumbnailOff** method is used to turn off thumbnailing of a stream. The io parameter can have a value between 1 to 4. The sid must be valid for the selected io.
tsStatsOn(io, sid)	The **tsStatsOn** method returns a TreeTableModel that provides a view of the data rates of each component present in the selected transport stream. The io parameter can have a value between 1 to 4. The sid value must be a valid stream id present in the ASI stream.
tsStatsOff(io, sid)	The **tsStatsOff** method should be called when the TreeTableModel is not required. This will prevent unecessary code from being executed. The io parameter can have a value between 1 to 4. The sid value must be a valid stream id present in the ASI stream.

Top Quick Reference

XMLHttpRequest

XMLHttpRequest is a JavaScript object that was initially created by Microsoft and later adopted by Mozilla. You can use it to easily retrieve data via HTTP. Despite its name, it can be used for more than just XML documents.

Basic usage

Using **XMLHttpRequest** is very simple. You create an instance of the object, open a URL, and send the request. The HTTP status code of the result, as well as the result document are available in the request object afterwards.

Example

```
var req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', false);
req.send(null);
if (req.status == 200) {
   dump(req.responseText);
}
```

Note that this example works synchronously, so it will block the user interface if you call this from your JavaScript. You should not use this in practice.

Asynchronous usage

If you intend to use **XMLHttpRequest** from an extension, you should let it load asynchronously. In asynchronous usage, you get a callback when the data has been received, which lets the application continue to work as normal while your request is happening.

Example

```
var req = new XMLHttpRequest();
req.open('GET', 'http://www.mozilla.org/', true);
req.onreadystatechange = function() {
   if (req.readyState == 4) {
      if (req.status == 200) {
         dump(req.responseText);
      } else {
         dump("Error loading page\n");
      }
   }
};
req.send(null);
```

Reference

Property/Method	Description
`pollInterval`	This read/write numeric field indicates the interval (in milliseconds) to use between successive requests for the HTTP page when using the asynchronous mode of operation. If set to a positive non-zero value prior to calling `send()`, the page will be requested repeatedly at the specified interval until the value is set back to zero. The default value is `0`, and if not modified it will result in a single HTTP request.
`readyState`	Represents the state of the request. Read-only. The following values are defined: 0 (`UNINITIALIZED`) The object has been created, but not initialized (the `open` method has not been called). 1 (`LOADING`) The object has been created, but the `send` method has not been called. 2 (`LOADED`) The `send` method has been called, but the status and headers are not yet available. 3 (`INTERACTIVE`) Some data has been received. Calling the `responseText` property at this state to obtain partial results will return an error, because status and response headers are not fully available. 4 (`COMPLETED`) All the data has been received, and the complete data is available in the `responseText` property.
`responseText`	Represents the response entity body as a `string`. Read-only.
`responseXML`	If you load an XML document, the `responseXML` property will contain the document as an `XMLDocument` object that you can manipulate using DOM methods. For loaded non-XML documents (whose `Content-Type` header does not indicate XML content), the `responseXML` property is `null`.
`status`	Represents the HTTP status code returned by a request. Read-only.
`statusText`	Represents the HTTP response line status. Read-only.
`onreadystatechange()`	Specifies the event handler to be called when the `readyState` property changes. Read/write.
`abort()`	Cancels the current HTTP request.
`getAllResponseHeaders()`	Retrieves the values of all the HTTP response headers.

`getResponseHeader()`	This method can be used to get an HTTP response header from the server response. It takes a single `string` parameter containing the case-insensitive header name.
`open()`	Initializes an HTTP request, and specifies the method, URL, and authentication information for the request. It takes up to five parameters, the last three of which are optional. The first parameter is the HTTP method to use as a `string`, such as `GET`, `POST`, `PUT`, or `PROPFIND`. The second argument represents the requested URL, either relative or absolute, as a `string`. The third (optional) argument is a `boolean` indicating whether the call is asynchronous, the default being `true` (the method returns immediately). If set to `true`, attach an `onreadystatechange` property callback so that you can tell when the `send()` call has completed. The (optional) fourth and fifth arguments are the username and password to use for authentication, if required.
`send()`	Sends an HTTP request to the server and receives a response.
`setRequestHeader()`	This method can be used to set an HTTP header on the request before you send it. It takes two `string` parameters: the header name and the header value.

Top Quick Reference

Quick reference

Connecting a router crosspoint:

```
navigator.connectCrosspoint("routerLongID", level, source, dest);
```

Distinguishing between right and left mouse button clicks:

```
if (
```

event.button== 1) { *// Left (main) button pressed*} **if**(event.button== 3) { *// Right (secondary) button pressed*}

Performing different actions based on the alarm status when it changes (in a button event handler):

```
switch (
```

status) { **case**NORMAL: *// Handle normal status*break; **case**WARNING: *// Handle warning status*break; **case**ERROR: *// Handle error status*break; **case**UNKNOWN: *// Handle unknown status*break; **case**DISABLED: *// Handle disabled alarm*break; **case**NON_EXISTENT: *// Handle missing alarm*break; **case**PENDING: *// Handle pending alarm*break; **default**: *// Some other (future?) value***break**; }

Reading document parameters (commonly known as the *query string* in Web parlance). This example will print each key/value pair individually:

```
for (i in
```

parameters) { window.alert(i + " <=> " + parameters[i]); }

To add a method to the `Document` object when it is loaded, you can use the following syntax in the page load event handler:

```
this.myMethodName = function myMethodName() {
   window.alert("myMethodName called on object " + this);
}
```

Exception handling:

```
try {
   // This is where you put the code that could fail
} catch (exception) {
   // This is where you handle any exception that might occur
 // If no exception occurs, this code will be skipped
   window.alert("Exception caught: " + exception);
}
```

If your script fails in an unexpected way, where you didn't handle the generated exception, a default error message will be shown. You can use explicit exception handling to show a different message or otherwise handle the exception differently. It can also be used to gain more fine-grained control over the handling of exceptions in your scripts. If your script needs to perform multiple actions, you can decide how the failure of the first should affect the execution of the second. For instance, this will not attempt to execute action #2 if action #1 fails:

```
try {
   // Action #1
 // Action #2
} catch (exception) {
   window.alert("Exception caught in action #1 or #2: " + exception);
}
```

Whereas this will attempt to execute action #2 whether action #1 fails or not:

```
try {
   // Action #1
} catch (exception) {
   window.alert("Exception caught in action #1: " + exception);
}
try {
   // Action #2
} catch (exception) {
   window.alert("Exception caught in action #2: " + exception);
}
```

To change the layout on a Kaleido Alto:

```
var alto = new

KaleidoAlto("10.5.8.4"); alto.loadLayout("mylayout");
```

To switch player assignment:

```
document.players["playerA"].assignment = document.players["playerB"].assignment;
```

To change player assignment:

```
var id = "m3.icontrol.com_M3_1_Densite_SLOT_9_31";
var assignment = document.players["playerA"].assignment;
assignment["video1.url"] = "rtsp://10.10.30.10/" + id;
assignment["video1.label"] = "label";
assignment["video1.vbi"] = "true";
assignment["video1.serviceID"] = "rtsp://10.10.30.10/" + id + "_Line21";
assignment["ralm1.url"] = "rtsp://10.10.30.10/" + id + "_A";
assignment["ralm2.url"] = "rtsp://10.10.30.10/" + id + "_A";
document.players["playerA"].assignment = assignment;
```

To change button status assignment:

```
this.getAlarm = function getAlarm(alarmDescriptor, newID) {
   var alarmDescriptor = alarmDescriptor.substring(alarmDescriptor.indexOf('@'));
   alarmDescriptor = newID + alarmDescriptor;
   return alarmDescriptor;
}

var id = "m3.icontrol.com_M3_1_Densite_SLOT_9_31";
var alarmDescriptors = buttons["toto"].alarmDescriptors;
alarmDescriptors.alarmDescriptor = this.getAlarm(alarmDescriptors.alarmDescriptor,
id);
buttons["toto"].alarmDescriptors = alarmDescriptors;
```

Examples using alarm panels

Add alarms to the panel, with different parameters specified:

```
// The alarm panel instance
var ap = document.getElementById("alarmPanel0");
// Add a text entry and set it to look like a title
ap.addEntry(null, ap.TITLE_TEXT_TYPE, "Status", true);
// Add simple alarm entries
ap.addEntry("health://appserver/densite/mX/dCPUStatus",  ap.STATUS_ALARM_TYPE,
"#1");
ap.addEntry("health://appserver/densite/mX",              ap.TEXT_AND_STATUS_TYPE,
"#2", false);
ap.addEntry("health://appserver/densite/mX/dFan1Status", ap.FOLLOW_RUNTIME_TYPE,
"#3", true);
// Add a folder entry
ap.addEntry("folderStatus://iControl+alarms/Health+monitoring/Densite+frame+m3+on
+appserver",
            ap.FOLLOW_RUNTIME_TYPE, null, true);


// Add a compressed alarm using a concatenated string of alarm URI strings
ap.addEntry(
                "alarm://scripted/Cycling+initialization+script" + ap.SEPARATOR +
                "test://cycling/user"                            + ap.SEPARATOR +
                "cycling:engine:status:Test+replication",
                ap.COMPRESSED_ALARMS_TYPE, "Compressed", false);
```

```
// Add a compressed alarm using an array of URI strings
var arr = new Array(3);
arr[0] = "test://cycling/user";
arr[1] = "alarm://scripted/Cycling+initialization+script";
arr[2] = "cycling:engine:status:Test+replication";

ap.addEntry(arr, ap.COMPRESSED_ALARMS_TYPE, "Compressed1", true);
```

Using regular expressions to change the locations of the URIs

Simple replacement:

```
// The alarm panel instance
var ap = document.getElementById("alarmPanel0");
// Look for this regular expression within the URIs and replace it with the
// following text
ap.dynamicAssignment("10.10.10.10/densite/HHH", "appserver/densite/mX");
```

Assume we had many URIs with similar locations but referencing different alarms, as the following:

```
health://10.10.10.10/densite/HHH/dCPUStatus
health://10.10.10.10/densite/HHH/dFan1Status
```

Simply what we would search for with our regular expression would be exactly what we want to replace:

```
10.10.10.10/densite/HHH
```

And we replace it with the new location:

```
appserver/densite/mX
```

Resulting with our new URIs pointing to a new location but maintaining what they originally were referencing:

```
health://appserver/densite/mX/dCPUStatus
health://appserver/densite/mX/dFan1Status
```

Generic (advanced) replacement:

```
// The alarm panel instance
var ap = document.getElementById("alarmPanel0");
// Look for this regular expression within the URIs and replace it with the
// following text
ap.dynamicAssignment("\\d+\\.\\d+\\.\\d+\\.\\d+/\\w+/\\w+",
                     "appserver/densite/mX");
```

Assume we had two URIs that looked like the following:

```
health://10.10.10.10/densite/HHH/dCPUStatus
health://15.15.15.15/densiteX/KX/dFan1Status
```

And we wanted to change the location of the URIs without changing what the URI originally pointed to. What we do is search the URI for a match to our expression and replace it with the new desired string location:

```
appserver/densite/mX
```

The new URIs we want look like:

```
health://appserver/densite/mX/dCPUStatus
health://appserver/densite/mX/dFan1Status
```

Here is the regular expression translated in simple language:

```
\\d+\\.\\d+\\.\\d+\\.\\d+/\\w+/\\w+
```

Look for one or more digits, followed by a period, followed by one or more digits, followed by a period, followed by one or more digits, followed by a period, followed by one or more digits, followed by a forward slash, followed by one or more word characters consisting of a letter or number, followed by a forward slash, and followed by one or more word characters consisting of a letter or number.

Simply it is looking for an IP address, separated by a forward slash, followed by a location name, separated by a forward slash, and followed by a location name.

Which in our original URI corresponds to:

```
10.10.10.10/densite/HHH
15.15.15.15/densiteX/KX
```

Resulting with our new URIs pointing to a new location but maintaining what they originally were referencing:

```
health://appserver/densite/mX/dCPUStatus
health://appserver/densite/mX/dFan1Status
```

If we do not use the generic regular expressions it would difficult to change the URIs. The regular expression matches a pattern that the user is interested in.

See Sun's regular expression tutorial for more information.

Example of a channel assignement using a spreadsheet.

```
var wb = new WorkBook("http://10.2.0.254/MBEInfo_4.xls");
this.allChannels = new Array();

for (var i = 0; i < 256; i++) {
   var channel = new Channel();
   channel.receiverID = (wb[0][i][0] == undefined) ? "" : wb[0][i][0].value;
   channel.ptcID = (wb[0][i][1] == undefined) ? "" : wb[0][i][1].value;
   channel.callLetter = (wb[0][i][2] == undefined) ? "" : wb[0][i][2].value;
   channel.cityID = (wb[0][i][3] == undefined) ? "" : wb[0][i][3].value;
   channel.affID = (wb[0][i][4] == undefined) ? "" : wb[0][i][4].value;
   channel.channelNumber = (wb[0][i][5] == undefined) ? "" : wb[0][i][5].value;
   channel.controlPhoneNumber = (wb[0][i][6] == undefined) ? "" : wb[0][i][6].value;
   channel.enginerName = (wb[0][i][7] == undefined) ? "" : wb[0][i][7].value;
   channel.engineerPhoneNumber = (wb[0][i][8] == undefined) ? "" : wb[0][i][8].
value;
   channel.stationPhoneNumber = (wb[0][i][9] == undefined) ? "" : wb[0][i][9].value;
}
```

To control an SNMP device directly from iControl Web (in a page event handler):

Simple example:

```
        var gsmIP = "10.0.4.80";
        var deviceIP = "10.0.4.111";

        this.onPresetButtonClick = function(value) {
           var mib = "WFM-MIB";
           var variable = "presetLoad";

           var expr = "gsm.snmpPlugins[\"" + deviceIP + "\"].snmp.set(\"" + mib +
"\",\"" + variable + "\",\"" + value + "\")";
           navigator.gsms[gsmIP].eval(expr);

           expr = "gsm.snmpPlugins[\"" + deviceIP + "\"].snmp.get(\"" + mib + "\",
\"" + variable + "\")";
           window.alert("New preset: " + navigator.gsms[gsmIP].eval(expr));
        }
```

Advanced example:

```
        function SnmpPlugin (gsmIP, deviceIP) {
           this.gsmIP = gsmIP;
           this.deviceIP = deviceIP;
           this.deviceIPString = "\"" + deviceIP + "\"";

           this.snmpSet = function(mib, variable, index, value) {
              if (undefined == value) {
                 if (undefined == index) {
                    // Only two parameters passed; second parameter is therefore the
      // value, not the variable.
                    value = variable;
                    variable = undefined;
                 } else {
                    // Only three parameters passed; third parameter is therefore the
      // value, not the index.
                    value = index;
                    index = undefined;
                 }
              }
              var args = this.snmpGetOIDArgs(mib, variable, index);
              args += ", \"" + value + "\"";
              var expr = "gsm.snmpPlugins[" + this.deviceIPString + "].snmp.set(" +
args + ")";
              return navigator.gsms[this.gsmIP].eval(expr);
           }

           this.snmpGet = function(mib, variable, index) {
              var args = this.snmpGetOIDArgs(mib, variable, index);
              var expr = "gsm.snmpPlugins[" + this.deviceIPString + "].snmp.get(" +
args + ")";
              return navigator.gsms[this.gsmIP].eval(expr);
           }

           this.snmpGetOIDArgs = function(mib, variable, index) {
              // One parameter specified
              if (undefined == variable) {
                 return "\"" + mib + "\"";
              }

              // Two parameters specified
              if (undefined == index) {
                 return "\"" + mib + "\", \"" + variable + "\"";
              }

              return "\"" + mib + "\", \"" + variable + "\", \"" + index + "\"";
           }
        }

        function Tektronix(gsmIP, deviceIP) {
           this.base = SnmpPlugin;
           this.base(gsmIP, deviceIP);

           this.setPreset = function(preset) {
              this.snmpSet("WFM-MIB", "presetLoad", preset);
           }

           this.getPreset = function() {
              return this.snmpGet("WFM-MIB", "presetLoad");
           }
        }

        var tek = new Tektronix("10.0.4.80", "10.0.4.111");

        this.onPresetButtonClick = function(preset) {
           tek.setPreset(preset);
           window.alert("New preset: " + tek.getPreset());
        }
```

Back