



**X O J O**

# **Web Control SDK**

# Preface

# Conventions

This document uses screen snapshots taken from the Windows, OS X and Linux versions of Xojo. The interface design and feature set are identical on all platforms, so the differences between platforms are cosmetic and have to do with the differences between the Windows, OS X, and Linux graphical user interfaces.

- **Bold type** is used to emphasize the first time a new term is used and to highlight important concepts. In addition, titles of books are *italicized*.
- When you are instructed to choose an item from one of the menus, you will see something like:

“choose File → New Project”

This is equivalent to “choose New Project from the File menu.”

- The items within the parentheses are keyboard shortcuts and consist of a sequence of keys that should be pressed in the order they are listed. On Windows and Linux, the Ctrl key is the modifier; on Macintosh, the ⌘ (Command) key is the modifier. For example, when you see the shortcut “Ctrl+O” or “⌘-O”, it means to hold down the Control key on a Windows or Linux computer and then press the “O” key or hold down the ⌘ key

on Macintosh and then press the “O” key. You release the modifier key only after you press the shortcut key.

- Something that you are supposed to type is quoted, such as “GoButton”.
- Some steps ask you to enter lines of code into the Code Editor. They appear in a shaded box:

```
ShowURL(SelectedURL.Text)
```

When you enter code, please observe these guidelines:

- Type each printed line on a separate line in the Code Editor. Don’t try to fit two or more printed lines into the same line or split a long line into two or more lines.
- Don’t add extra spaces where no spaces are indicated in the printed code.

Whenever you run your application, Xojo first checks your code for spelling and syntax errors. If this checking turns up an error, an error tab appears for you to review.

# Creating Custom Web Controls

---

This Guide describes how to integrate 3rd party web controls into Xojo.

# Overview

## Basic Concepts

Creating Web Controls in Xojo is a give and take relationship that requires careful planning for optimal execution.

Read each section of this guide so that you have a good working knowledge of how communications with 3rd party controls works.

## Namespaces

For the purposes of avoiding conflicts between framework code, your code and anything else that you might put in your application, you must use a namespace for your controls.

The root namespace for custom controls is XojoCustom, which you should use for all your JavaScript code. Refer to the [Namespaces](#) section for more information.

## Control Naming Conventions

To prevent name conflicts and to make it easier to identify in lists, you should prefix your classes. For example, a company named Acme might name their controls Acme\_Button or Acme\_SourceList.

## Event Order

Understanding the initial sequence of events that fire when your control is created and then added to a web page is crucial for successful deployment and the Xojo developer experience.

For more information about the Events that are available to you and the order in which they fire, please see the [Events](#) section.

# Getting Started

## A Basic HTML Control

This is a quick tutorial on how to create a custom control for your web applications. You are going to create a simple control for displaying HTML. To get started:

1. Create a new Web Project.
2. Add a Class to the project
3. Set the name of the class to MyHTMLArea
4. Set the Super of the class to [WebControlWrapper](#)
5. We need a place to store the HTML for this control. Add a public property to the class: HTML As String
6. Create a Private Constant named JavascriptNamespace and set the value to "Example".

This creates a namespace for this control called `XojoCustom.Example`. All controls are required to return a namespace, even if there will be no browser-side code associated with it. For more information, refer to the [Namespaces](#) section.

7. In the [SetupHTML](#) event handler, add this code:

```
Return "<div id=\"" + self.ControlID + "\">" + html + "</div>"
```

8. Controls are initially invisible so you have time to set them up before they appear. In the **SetupCSS** event handler, add this code:

```
Styles(0).Value("visibility") = "visible"
```

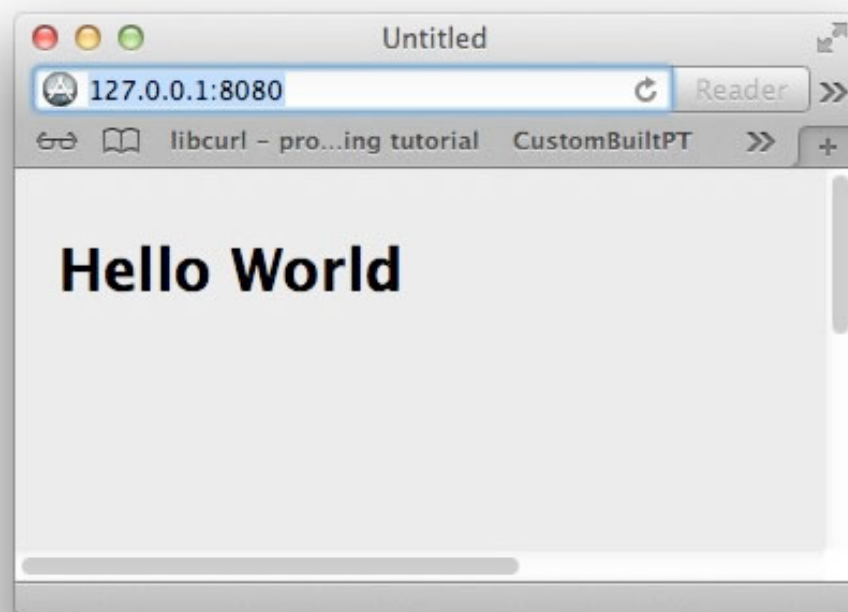
9. On WebPage1, drag an instance of your new MyHTMLArea control to the Page.

10. In the Open event handler on the control, add this code:

```
Me.HTML = "<h1>Hello World</h1>"
```

11. Run the project. You should get some big bold text rendered in HTML!

**Figure 1.1** Simple HTML Control Output



## Runtime HTML Changes

So what happens if you want to be able to change the HTML at runtime? It gets a little trickier because you can't send a JavaScript command to your control until it's been rendered to the page, and you can't be sure of that until after the Shown event has fired.

1. On MyHTMLArea, convert the **HTML** property to a Computed Property (right-click on it and select Convert to Computed Property) and add this code to the Setter:

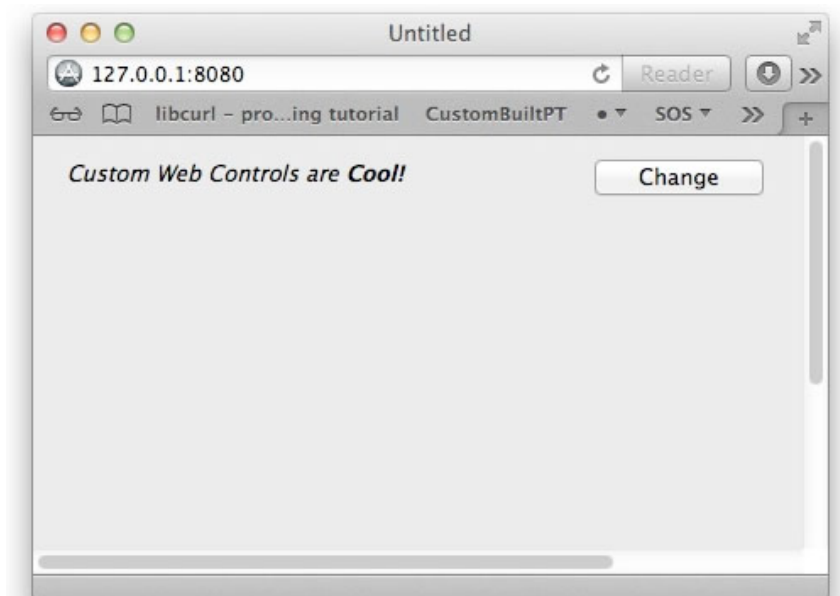
```
mHTML = value
If ControlAvailableInBrowser() Then
    ExecuteJavaScript("Xojo.get('" + me.ControlID + "').innerHTML = '" + value + "';")
End If
```

2. Go back to WebPage1 and add a Button to the page.
3. Change its Caption to "Change"
4. In the Action() event for the button add the following code:

```
MyHTMLArea1.HTML = "<i>Custom Web Controls are " + _
    "<b>COOL!</b></i>"
```

5. Run the project. Clicking the button changes the HTML in the control.

**Figure 1.2** Changing HTML Dynamically





## Two-Way Communication

Now that you have created a custom control on the page and updated it at runtime, it would be really nice to be able to send events back to the server in response to user input.

With these steps, you can make it so that any links that are embedded in the user-supplied HTML are sent to the server instead of being blindly followed within the control.

1. First, add the event that will fire when the user clicks a link. Go to the MyHTMLArea control and add an Event Definition:

```
LinkClicked(URL As String)
```

2. Add a public Method to the control:

```
Function FixLinks(html As String) As String
    Dim returnValue As String

    returnValue = HTML.ReplaceAll("<a", "<a onclick=""Xojo.triggerServerEvent('" + _
    Self.ControlID + "', 'LinkClicked', [this.href]);return false;""")

    returnValue = returnValue.ReplaceAll("""", "\"\"")
    returnValue = returnValue.ReplaceAll("'", "\'")

    Return returnValue
End Function
```

3. Go to the SetupHTML Event and change the code to:

```
Dim htmlCode As String
htmlCode = "<div id="" + Self.ControlID + "">" + FixLinks(HTML) + "</div>"

Return htmlCode
```

4. Go to the Setter for the HTML Computed Property. Change the code to this:

```
mHTML = value

If ControlAvailableInBrowser() Then
    Dim s As String
    s = "Xojo.get('" + Self.ControlID + "').innerHTML = '" + _
        FixLinks(value) + "';"
    ExecuteJavaScript(s)
End If
```

5. Now you need to listen for events from the browser. In the Event Handler for **ExecuteEvent** add the following code:

```
Select Case Name
Case "LinkClicked"
    Dim URL As String = Parameters(0)
    If Left(URL, 7) = "msgbox:" Then
        MsgBox(Mid(URL, 8))
    Else
        LinkClicked(Parameters(0))
    End If
End Select
```

6. Back on WebPage1, change the code in the Change button's Action event to:

```
MyHTMLArea1.HTML = "<i>Custom Controls are " + _  
    "<b><a href=""msgbox:Very Cool!"">COOL!</a></b></i>"
```

7. Run the project, click the button to change to the new code and then click the link!

**Figure 1.3** Custom Control Displaying a Link



# Namespaces

## Basic Concepts

All user controls and code must be created in the root namespace, XojoCustom. This is necessary for these reasons:

1. Prevent pollution of the Global JavaScript namespace.
2. Reduce the possibility of controls from different developers conflicting with each other.
3. Reduce the possibility of conflicting with an existing JavaScript framework (jQuery, YUI, etc.)

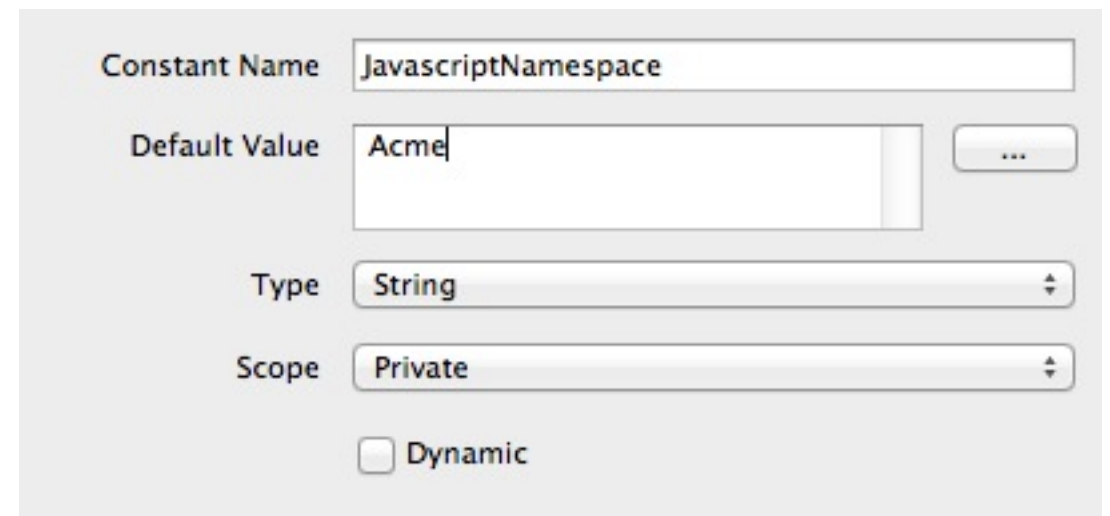
While the IDE currently does not check for namespace conflicts, it will in the future, so it is extremely important that you follow our guidelines in this area so that your users will not be affected when this change is made.

Controls that were created to use the RSCustom namespace will continue to work.

## Specifying Your Namespace

In an attempt to prevent different developers' code from conflicting, you need to specify your namespace using a constant named **JavascriptNamespace**.

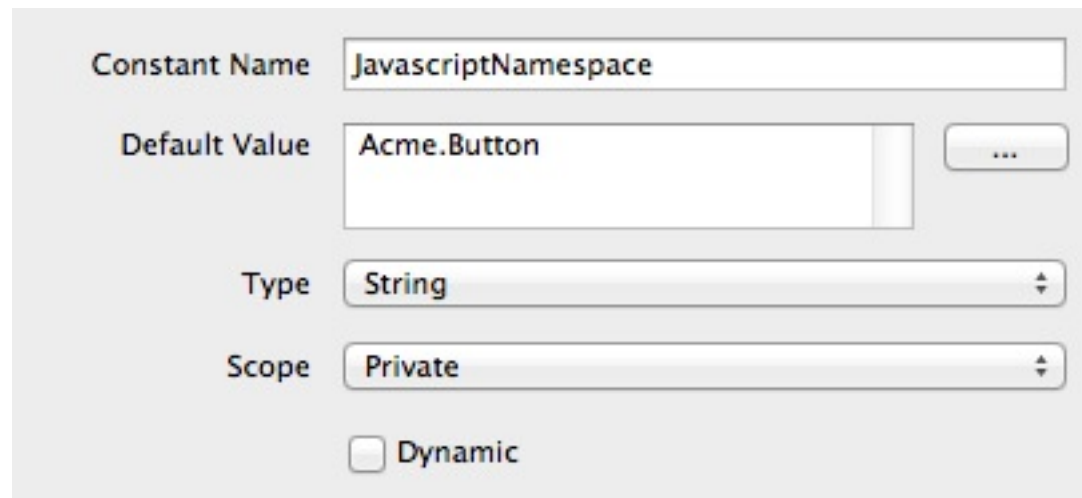
**Figure 1.4** - Defining a simple Javascript namespace



The screenshot shows the 'Define Constant' dialog box in the Xojo IDE. The 'Constant Name' field is set to 'JavascriptNamespace'. The 'Default Value' field contains 'Acme' and has a small '...' button to its right. The 'Type' dropdown menu is set to 'String'. The 'Scope' dropdown menu is set to 'Private'. At the bottom, there is an unchecked checkbox labeled 'Dynamic'.

If you need more precise control of the namespace, perhaps because you are creating an entire library of controls, you can specify additional namespace levels using dot notation as in [figure 1.5](#).

**Figure 1.5** - Defining an advanced Javascript namespace



The string you return will be prefixed by XojoCustom such that the created namespace is `XojoCustom.Acme.Button`.

Failing to define a namespace will cause a `NamespaceException` at runtime. In a future version of the IDE, you may also get a compile-time error.

## Registering Your Namespace

Xojo is maintaining a Javascript namespace registry to help developers avoid namespace conflicts. Developers may register a root namespace for use with all of their controls. Using the examples above, the registered namespace would be “Acme”. From then on, you would be permitted to create:

- Acme.Button
- Acme.Listbox
- Acme.SourceList

This does NOT permit you to use `AcmeControls` or `AcmeProducts` without creating a separate namespace registration.

To request a root namespace registration, send an email to [WebNamespace@xojo.com](mailto:WebNamespace@xojo.com) and tell us what root namespace you would like to use. You will receive an email from us confirming your namespace choice.

Namespaces will be registered on a first come first served basis. We reserve the right to deny a namespace request that infringes on the trademarks of another company or would be otherwise confusing or offensive.

## Guidelines

For the sake of clarity, namespaces:

- Should be at least 3 characters in length
- May only contain letters, numbers and \_
- May not be a [Javascript or ECMAScript keyword](#)

**NOTE:** The namespace “Example” is reserved for use in our example projects. Please do not use this namespace for controls you intend to distribute.

We will not be enforcing namespace case-sensitivity. If you register “Acme”, you may also use “acme” or “ACME” or any combination you can think of as long as the characters themselves do not change. That being said, remember that Javascript **is** case-sensitive.

**Figure 1.6** Javascript/ECMAScript Reserved Words

abstract, arguments, Array, Boolean, boolean, break,  
byte, case, catch, char, class, const, continue, Date,  
debugger, decodeURI, decodeURIComponent,  
default, delete, do, double, else, encodeURI, enum,  
Error, escape, eval, EvalError, export, extends, final,  
finally, float, for, function, Function, goto, if,  
implements, import, in, Infinity, instanceof, int,  
interface, isFinite, isNaN, long, Math, NaN, native,  
new, Number, Object, package, parseFloat, parseInt,  
private, protected, public, RangeError, ReferenceError,  
RegExp, return, short, static, String, super, switch,  
synchronized, SyntaxError, this, throw, throws,  
transient, try, TypeError, typeof, undefined, unescape,  
URIError, var, void, volatile, while, with

# WebControlWrapper

In addition to the events that all Web Controls have available, custom controls also have the following events (listed in the order that they are normally called):

## Initialization

### *Open()*

Fires when your control is initially created. During this event handler, your control does not yet exist in the browser. You should not be sending any commands to manipulate your control to the browser. You may, however, send commands to load external libraries or do other simple initialization.

### *SetupHTML() As String*

It is here that you create the HTML code necessary to initially draw your control. At the bare minimum, you should return a DIV tag with an ID that matches your control like this:

```
Return "<div id=\"" + Self.ControlID +  
      "\"></div>"
```

If you use a different id for the outer element, you may not receive events from the framework.

### *SetupCSS(ByRef Styles() As **WebControlCSS**)*

In this event, you add to the CSS parameter the properties necessary to make your control appear correctly. See the Styles section for more info. You will be passed a single stylesheet for the ID of the control you are creating.

Refer to the **Styles** section for more information.

### *SetupJavaScriptFramework() As String*

If your control requires any sort of framework (like jQuery or YUI) and you will be including the JavaScript code, return that code here.



## Runtime

At this time, all of the data that you (and any other controls) have provided is sent to the browser. Once the framework executes all of the commands, your control will receive a Shown event.

### ***Shown()***

This event is called once your control has been received and created by the browser. You may now send commands directly to the control and your framework.

**Note:** Once the Shown event has fired the first time, *ControlAvailableInBrowser* method will return True. The Shown event will also fire if your control has been hidden and shown again.

### ***ExecuteEvent(Name As String, Parameters() As Variant) As Boolean***

This event is called when an event is received from the web page. Name is the name of the event, Parameters is an array of parameters in the order they were specified in the **triggerServerEvent** method. Return True if you are handling the event.

### ***FrameworkPropertyChanged(Name As String, Value As Variant)***

This event is called when a framework property has changed, giving you an opportunity to modify your control's appearance. For example, if you receive an event where name = "enabled" you would send the JavaScript command(s) to enable or disable your control.

## Methods

### ***EventIsImplemented(ObjectType As Introspection.TypeInfo, EventName As String) As Boolean***

Allows you to find out whether the named event has actually been implemented by a subclass or instance. The reason for this method is because you may not need to listen for certain events on the browser and possibly not even send data back to the server if the user doesn't need the information.

```
If EventIsImplemented(GetTypeInfo(YourClassName),  
"Click") Then  
    ...Send code to browser for handling the "Click"  
    event...  
End If
```

### ***ControlAvailableInBrowser() As Boolean***

You can call this method to determine if the control is already rendered to the users web browser. This method is useful for determining whether or not a property change should be sent directly to the browser when it is changed. If it returns false, you should render the property change as part of the setup events or defer the change until after the Shown event has fired.

***LoadLibraries(CallbackFunction As String, ParamArray URLs As String)***

***LoadLibraries(CallbackFunction As String, URLs() As String)***

Loads the specified JavaScript libraries, in the order that you specify, and then calls the callback function. If the library has already been loaded it will not be loaded again.

- CallbackFunction: The JavaScript function to call when this script has finished loading.
- URLs: The URLs of the JavaScript files that you wish to load.

## Shared Methods

***IsLibraryRegistered(Session as WebSession, JSNamespace as String, LibraryName as String, minVersion as integer = 1) as Boolean***

Allows you to determine if a named/versioned library has already been loaded for a particular WebSession. [New in 2013r1.](#)

***RegisteredLibraryVersion(Session as WebSession, JSNamespace as String, LibraryName as String) as Integer***

Returns the version number of a registered library. This is helpful if you have a master library that evolves over time and need to check for minimum compatibility. [New in 2013r1.](#)

***RegisterLibrary(Session as WebSession, JSNamespace as String, LibraryName as String, Version as Integer = 1)***

Registers a named library so you can look it up later. If you have already registered the library, the version number must be greater than the stored version to change it. If you need to lower the version number, you must first call UnregisterLibrary. [New in 2013r1.](#)

***UnregisterLibrary(Session as WebSession, JSNamespace as String, LibraryName as String)***

Unregisters a named library. This command is not version specific. [New in 2013r1.](#)

## Hooks

***HTMLHeader(CurrentSession as WebSession) as String***

Override this *shared* method to dynamically add items to the <head> section of the session as it loads. This method is called when an end user's session is first being created and no control instances have been created yet. You may use the CurrentSession parameter to examine the Session that is being created to load different headers based on Session conditions. See the jQueryCalendar project for a usage example. [New in 2014r2.](#)

## Constants

Configuring your control for use in the IDE is done through the use of constants. To avoid user confusion, these constants should be Private.

### ***APIVersion as Integer***

The current API version, matching the version that is available from the JavaScript method `Xojo.APIVersion()`.

2012r2 = 1

2013r1 = 2

2014r2 = 3

### ***IncludeInTabOrder***

This constant is used to define whether or not your control needs to be included in the tab order. If set to `True`, it will be included when the user is setting the tab order for a `WebPage`. This value is ignored for tray-only non-visual controls. [New in 2013r1](#).

### ***JavascriptNamespace***

This constant is used to define the Javascript namespace for this control. For more information, see the [Namespaces](#) section.

### ***LayoutEditorIcon***

This constant is used to override the appearance of your control in the Layout Editor. Using the `WebSDKIconConverter` project in the `WebSDK Tools` folder, you can convert an icon to Base64 and paste the result into the value of this constant. The next time the IDE requests the icon for your control, it will use this one instead. See the [IDE Integration](#) section for more information.

*Note: You may need to close your project and re-open it for the icon to change because we cache the icon for speed.*

### ***NavigatorIcon***

This constant is used to override the appearance of your control in the Navigator. Using the `WebSDKIconConverter`, place the Base64 icon data into the value of this constant.

*Note: You may need to close your project and re-open it for the icon to change because we cache the icon for speed.*

### ***ShowInTrayArea***

When creating a non-visual control, rather than having them appear on the layout itself, you may want them to appear in the tray at the bottom of the editor. To make your control appear in the tray, set this constant to `True`. See the [IDE Integration](#) section for more information.

### ***ShowStyleProperty***

Unlike the other properties of a `WebControl`, the `Style` property is not controlled by the Inspector Behavior dialog. If you will not be supporting user applied styles in your control, set this constant to `False`. See the [IDE Integration](#) section for more information.

# JavaScript Methods

There are ten JavaScript methods you can use in your controls. As of Xojo 2013r1, these methods are not sent to the browser unless there is at least one WebSDK control in the user's project.

## Events

***Xojo.triggerServerEvent(controlID As String, eventName As String, userData As Array)***

This method is used to send an event from a browser, back to your control.

**Note:** You will also use `Xojo.triggerServerEvent` to send property updates to the server. For simplicity and efficiency, you should create a specific event that will be used for all property updates and set alternating values of the `userData` array to the name of the property and the value. For more complex data structures, you may want to create a JSON object, pass that as the `userData` and then use the `JSONItem` class to parse it on the server. All of the supported browsers now have built-in JSON libraries.

- `ControlID`: the DOM ID of your control, as generated by the framework.
- `eventName`: the name of the event as you want it to be sent to `ExecuteEvent` in your control.

- `userData`: A JavaScript array of properties that you want to send with your event. They will be available in the `Parameters()` array in `ExecuteEvent`.

***Xojo.addListener(controlID As String, eventName As String, fn As Function)***

This method is a wrapper for a cross-platform version of the `addEventListener/AttachEvent` method in JavaScript. You must use this method for adding events if you want to use `triggerBrowserEvent` (see below) on all supported browsers.

***Xojo.removeListener(controlID as String, eventName as String, fn as Function)***

This method is a wrapper for a cross-platform version of the `removeEventListener/detachEvent` method in JavaScript. You must use this method for removing events if you want to use `triggerBrowserEvent` (see below).

### ***Xojo.triggerBrowserEvent(controlID as String, eventName as String, properties as Array)***

This method triggers a browser side event. When used with Xojo.addListener and Xojo.removeListener, it offers a fallback for browsers that do not support custom events.

## **Styles**

### ***Xojo.addClass(Control As String/Object, ClassName As String)***

Adds a class to the specified control, if it does not already exist.

### ***Xojo.hasClass(Control as String/Object, ClassName as String) as Boolean***

Returns True if the specified control has the class applied to it.

### ***Xojo.removeClass(Control as String/Object, ClassName as String)***

Removes a class from the specified control, if it exists.

## **Misc**

### ***Xojo.get(controlID As String)***

This method is an alias for "document.getElementById"

### ***Xojo.loadScript(controlID As String, URL As String, JSCallback As Function, Location As String, Source As String)***

This method is for dynamically loading JavaScript scripts into the application. It mainly exists for the purpose of loading third-party libraries that you may need for your controls to work.

- **ControlID:** The DOM ID for this script. Included so that you can remove it later if your control is to be removed from a page. This value should be prefixed by the ControlID of your control, e.g. `Self.ControlID + "_framework"`.
- **URL:** The URL of the script to load. You may pass an empty string if you want to send the code directly.
- **JSCallback:** A JavaScript function to call when the script has been loaded onto the page.
- **Location:** The place where this code will be added. You can pass either an identifier, a tag (like "head") or a DOM element.
- **Source:** If a URL is not supplied, you can provide JavaScript source code here. If a URL is supplied, this value is ignored.

## **Properties**

### ***Xojo.apiVersion()***

Returns the current version number of the framework API. This number is updated whenever significant changes are made to the operation or functionality of the Framework or JavaScript API.

# Javascript Events

Starting in Xojo 2013r1, there is a new Javascript “disconnect” event available which fires when a user’s browser disconnects from the web application and goes offline.

This event may be helpful if you are creating a control such as a movie or audio player and you wish to have the control automatically stop playing if the browser disconnects.

To use this event,

```
Xojo.addListener("XojoSession", ↵  
    "disconnect", disconnectMethod);
```

# Styles

## Overview

When your control is first being rendered to the HTML document that is being sent to the browser, you have an opportunity through the **SetupCSS** event to change the default CSS that is provided.

The **SetupCSS** event has a single parameter which is an array of **WebControlCSS** objects. The initial array will contain a single ID selector for your control with a stylesheet that reflects the positioning information that was specified in the Web Page Layout Editor. You may add more **WebControlCSS** objects to the array as necessary for your control to function. If you will be adding more stylesheets, use the control ID as a prefix to the selector to prevent conflicts with the framework itself or other 3rd party controls.

Keep in mind that by default, users can apply a **WebStyle** object to your control and that it may change the appearance. If you wish to support this behavior, make sure you apply the appropriate class in the **SetupHTML** event and be sure to test your control with a wide variety of styles and properties.

```
dim tag as string
tag = "<div id=\"" + self.controlid + "\""

if Right(self.style.name,8) <> "_default" then
    tag = tag + " class=\"" + self.style.name + "\""
end if

tag = tag + "></div>"

return tag
```

## Notes

It is important to remember that when a user positions a control in the Web Page Editor, they have a reasonable expectation that the control will appear in that position, using the dimensions that they specified. The only reason that you should be changing properties that would alter the position of your control is to make sure that the user's expectations will be maintained. For example, if you are wrapping a control that renders HTML above or below the supplied area, you should change the top and height properties so that the control will still render within the boundaries set by the user. Failure to follow these guidelines may

cause your control to draw over (or under) other controls and make them unusable. For an example of how to do this correctly, refer to the YUI Rich Text Editor example.



# JavaScript Libraries

In addition to creating controls from scratch, you can also wrap existing JavaScript control libraries. As you can see from the [Events](#) section, each control is given an opportunity to send JavaScript to the browser as part of its setup process. If you will be using a centralized library, make sure the common code is only sent to the browser only once.

One solution would be to send a loader to the page which keeps track of which libraries have already been loaded. When an event comes from the server asking about a particular library, it checks its local cache (or perhaps directly at the page itself). If the library is not present, it would send an event back to the server requesting it, or load it from a global repository in the way the YUI library is distributed. For more information about installing JavaScript onto a web page at runtime, refer to the [LoadLibraries](#) method.

# Embedding Images

For the greatest simplicity in delivering your controls, we suggest embedding any images you may need for your control to work right into the class as constants. You can do so by using the same technique that we use for [rendering your icon in the IDE](#).

Basically the technique involves creating [WebPicture](#) objects inside your class at runtime, and then using the URLs provided to return an image to the browser.

1. Convert the data from your image to Base64. You can do this using the tool we provide for [IDE Integration](#).
2. Create a string constant in your control to represent the image data. Insert the Base64 text into the value field.
3. Create a property on your class to represent this image at runtime of type [WebPicture](#). If the image will be common to all instances of your control, make sure it is Shared to save memory!
4. In the Open event, set the [WebPicture](#) property by converting the image data back into a picture, using code like this:

```
If PicProperty = Nil Then
    PicProperty = New WebPicture
    PicProperty.Data = DecodeBase64(PicConstant)
    //If the image is shared, unset the Session
    PicProperty.Session = Nil
End If
```

5. In your HTML, Javascript or CSS template, use an easy to remember and easy to see placeholder for the image URL. To use the picture, just access the URL property of the [WebPicture](#):

```
Dim URL = PicProperty.URL
html = html.ReplaceAll("<<PicURL>>", URL)
```

This technique can also be used for other kinds of files that need to be deliverable to the browser at runtime.

To see an example of using shared images, please see the iOS Background example project.

# Localization

Dynamic constants can be used to localize web applications. It is important that your controls support this functionality for your international audience.

## Why Non-Dynamic Constants Don't Work

The first question that usually gets asked is "Why don't normal constants work?", and the answer has to do with where the server code is actually running. If you think about how non-dynamic Constants typically work, when compiled, the value that matches the build platform and language are used and everything else gets stripped out. Conversely, every language of a dynamic constant is included.

## Dynamic Constants in Web Apps

To enable values that change on a per browser basis, web applications pull the language of the web browser when a WebSession starts so that the application can deliver correct localization data. Additionally, if the web application doesn't support the language in question, the developer has the ability to set the language code at runtime. This means that your controls should fluidly adapt to text changes, either by changing their size or clipping contents.

## Right to Left Languages

In addition to the localization options, WebSessions support Right-to-Left text, such as this:

بالإضافة إلى خيارات الترجمة، دورات على شبكة الإنترنت الآن أيضا دعم من اليمين  
لى اليسار النص.

Remember that text may wrap differently and be right justified for these languages. You can find out what the document is currently set to by checking the "dir" attribute of the body tag. it will be set to "rtl" for these languages and should be missing for left to right text.

# IDE Integration

## Default Control Size

You can change the default size of your control by setting the default Width and Height properties. You can do this by right-clicking on the control in the Navigator and selecting “Inspector Behavior”.

## Property Inspector

If there are properties in your control that you would like to make visible in the Property Inspector in the IDE, you can also do that in the Inspector Behavior window, as well as add a special heading and set the editor types.

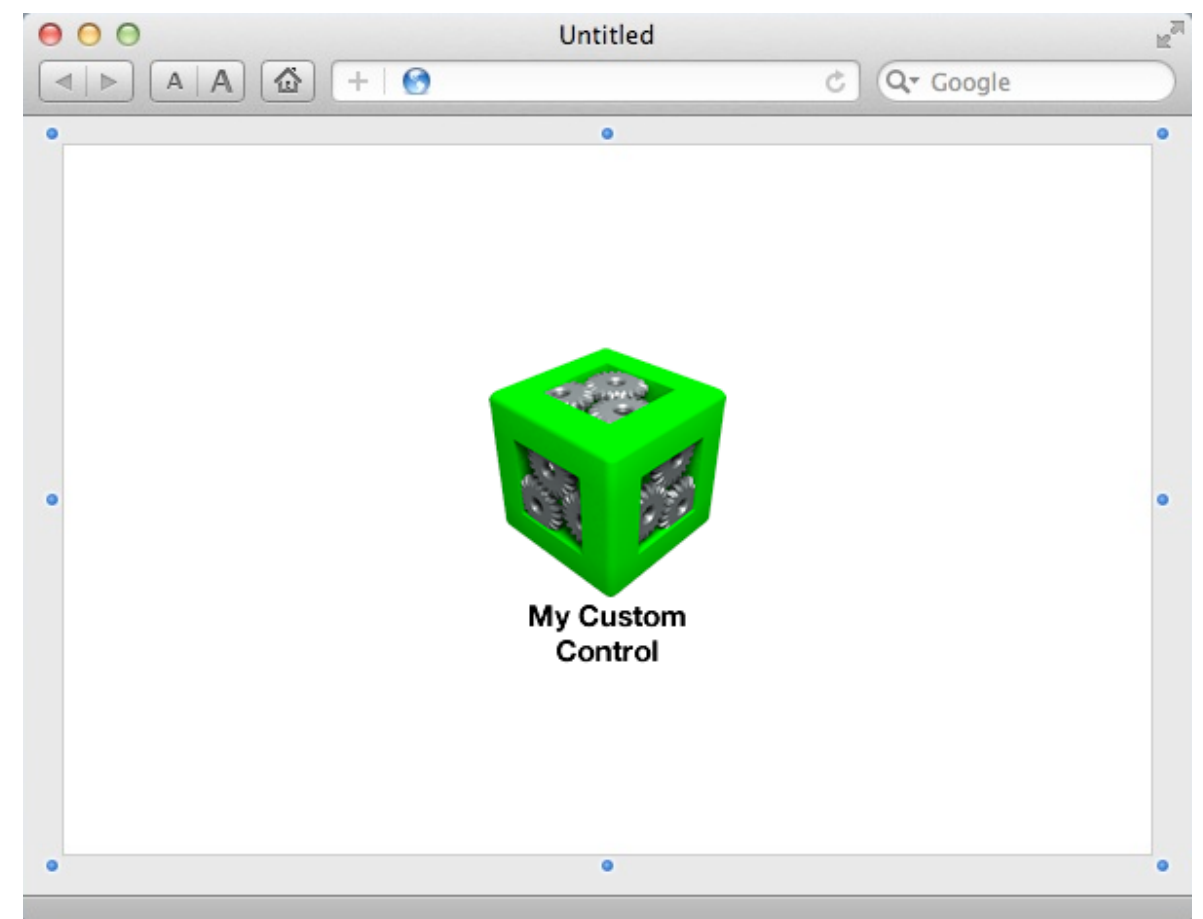
## Layout Editor Icons/Previews

You can provide a static image for your controls so that your developers will be able to tell at a glance what controls are on their layouts.

To do so, you create a custom icon or image, which is then displayed in the center of the control area.

In the WebSDK Tools folder, there is a project that converts your image into a Base64 encoded string which you can then add to

your control as a constant called **LayoutEditorIcon**. If you wish to customize your control’s appearance in the Navigator, use the same technique and add a constant named **NavigatorIcon**.



NOTE: Although the image can technically be any size, for best results we suggest that you create an icon that is approximately

128x128 pixels. If the control is scaled below the size of the icon, it will automatically scale to fit. Control icons are cached, so if you change your icon, you will need to reload your project to see the new icon.

To see how to use an icon, refer to the jQuery Calendar example project.

## Style Property

The property that allows an end-user to apply a style to a control is created at runtime and cannot be controlled by the Inspector Behavior editor. If you do not want to allow users to apply a style to your control, you can add a boolean constant to your control named **ShowStyleProperty** with a value of **False**.

## Non-Visual Controls

If your control has no visual component associated with it, you may want to have your control appear in the tray at the bottom of the Web Page editor. To do this, add a constant to your control named **ShowInTrayArea** with a value of **True**. Non-Visual controls have their Style property automatically hidden.

The icon that appears in the tray is a 28x28 pixel black-only icon. This icon will be derived from the LayoutEditorIcon by using its mask which has been applied to a black picture. For best results, use a transparent PNG file.

For the best user experience, we suggest that you use the Inspector Behavior editor to hide all of the built-in properties except Index, Name and Super, as the rest of them are meant for visual controls.

To see an example of a non-visual control, please see the iOS Background project in the examples folder.

## Tab Order

If your control has textfield, textarea or button controls within it, you will probably want end users to be able to include your control in the tab order for the WebPage. To do so, add boolean constant to your control named **IncludeInTabOrder** with a value of **True**. Please note, Non-visual controls cannot be included in the tab order, regardless of the value of this constant.

# Testing, Compatibility and Bugs

## Testing and Compatibility

The standard framework controls have been designed to behave as consistently as possible on all supported browsers. For the best developer experience, your custom controls should do the same. Supported browsers are:

**Mac:** Safari 5+, Chrome, and Firefox 10+

**Windows:** Internet Explorer 8+, Chrome, Firefox 10+, Safari 5+

**Linux:** Firefox 10+

**iOS:** Mobile Safari, Chrome

**Android 3+:** Android Browser, Firefox 10+

If you will **not** be supporting one or more of these browsers, please make sure your users are aware of so that they do not inadvertently report bugs.

Because of the rapidly changing browser market, Xojo will re-evaluate the supported browser list with each major release.

## Bug Reports

For maximum efficiency in resolving framework bugs that affect your controls, your users should report bugs regarding your controls directly to you.

If you determine that there is an issue or problem in our web framework, **you** should file a bug report using Feedback and notify your users to mark it as a favorite and possibly add it to their top 5 cases.

# Important Notes

## Xojo Events

It is important to remember that developers using your controls will not expect to see the Web SDK events in their controls. Any events that you do not implement should have at least a comment in the code so developers will not be confused (and possibly break your control). This includes:

- `ExecuteEvent`
- `FrameworkPropertyChanged`
- `SetupCSS`
- `SetupHTML`
- `SetupJavascriptFramework`

## Memory and Bandwidth

Memory and bandwidth usage is of the utmost importance for web applications because of the often limited resources of mobile devices, especially when combined with general Internet latency.

- **Memory:** Web applications are all one page, so ensure your controls are not taking up valuable resources unnecessarily.

- **Bandwidth:**

- If you are monitoring events that fire quickly (like mouse movement), you should throttle the events so that you're only sending as many as the user needs to accomplish the task. Overloading the framework with events not only makes your controls slow, it affects the responsiveness of the entire web page and application.
- Make sure you are only sending the least amount of data needed. This is especially important for developers that are targeting mobile devices. For instance, if you are creating a toggle control, send "1" and "0" instead of "true" and "false". When dealing with mobile devices and those with limited Internet connections, every byte makes a difference.

## 3rd-Party Libraries

If you wrap a third-party control or library, be sure to document the usage license and provide links so your users can make an informed choice about their usage of the code.

## CSS

It is important to remember that when a user positions a control in the IDE, they have a reasonable expectation that the control will appear in that position, using the dimensions that they specified. The only reason that you should be changing properties that would alter the position of your control is to make sure that the user's expectations will be maintained. For example, if you are wrapping a control that renders HTML above or below the supplied area, you should change the top and height properties so that the control will still render within the boundaries set by the user. Failure to follow these guidelines may cause your control to draw over (or under) other controls and make them unusable. For an example of this, please see the YUI Rich Text Editor example.

## Xojo DOM & Framework

You may not add, remove or modify any part or property of the Xojo namespace or framework at design-time or runtime for any reason. You may not provide developer patches to the web framework or API.

Xojo reserves the right to modify the undocumented parts of the web framework without notice. Therefore you should not call undocumented parts of the web framework or rely on its internal structure. Doing so may cause your controls to fail in customer projects.

When it comes to the Document Object Model (DOM), you are only permitted to manipulate the content within your own

controls. Changing the structure of built-in controls or the controls of other 3rd-Party developers is not permitted.

## Shared Environment

Don't forget that your control will live in a shared environment, with the web framework itself and possibly also with other developer's controls. When writing code for your controls, be careful to make changes that only affect your controls. For instance, you should not be adding a class to the <body> tag or even to an entire WebPage. Styles should not be added to a particular tag.