MiServer 3.0  
Quick Start Guide

September 2015

Table of Contents

[Download MiServer 3.0 3](#_Toc506212338)

[Install MiServer 3.0 4](#_Toc506212339)

[Start MiServer 3.0 5](#_Toc506212340)

[A Few Terms 6](#_Toc506212341)

[Creating a MiSite 7](#_Toc506212342)

[Recommended Folder Structure for a MiSite 7](#_Toc506212343)

[Recommended Location for Your MiSite 7](#_Toc506212344)

[Sample MiSite 7](#_Toc506212345)

[Building MiPages – Basic Concepts 8](#_Toc506212346)

[Widgets, Controls, and Elements 9](#_Toc506212347)

[Writing MiPage Content 10](#_Toc506212348)

[Add-Pitfalls 11](#_Toc506212349)

[Putting it all together - your first Misite and .mipage 12](#_Toc506212350)

[Event Handling – Specifying Handers 13](#_Toc506212351)

[Event Handling - ClientData 14](#_Toc506212352)

[Event Handling - Retrieving Client Data from Callback Functions 15](#_Toc506212353)

[Event Handling – Sending Responses Back to the Client 16](#_Toc506212354)

[Function Reference 17](#_Toc506212355)

[#.Start 17](#_Toc506212356)

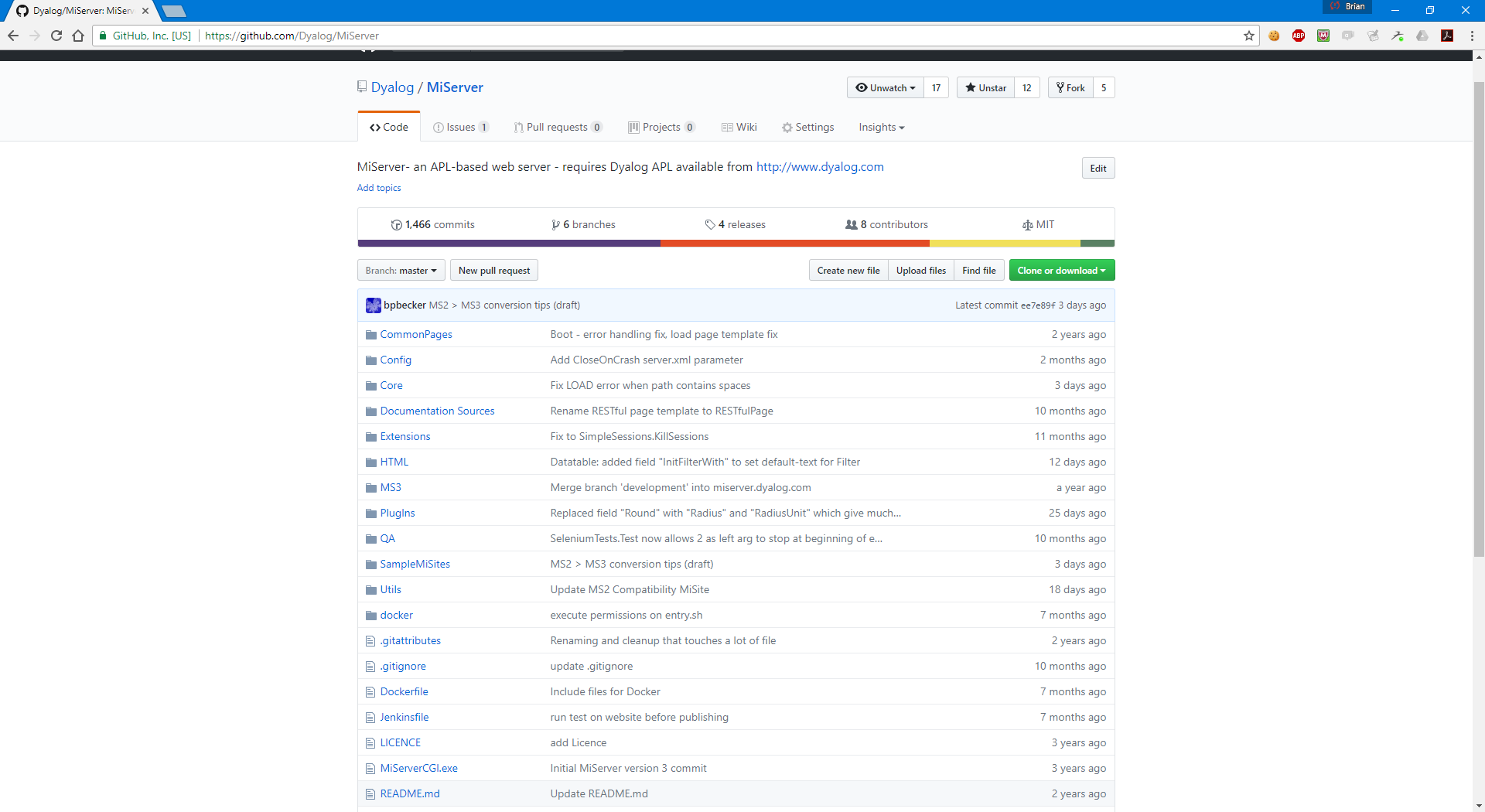
[Configuration Settings 18](#_Toc506212357)

[The Configuration Settings You Probably Care About 18](#_Toc506212358)

## Download MiServer 3.0

You can download MiServer by

1. going to **https://github.com/Dyalog/MiServer** and
2. clicking the "Clone or download" button



What's the difference between cloning or downloading?

Downloading will just download a ZIP file that represents a snapshot of MiServer as it is right now.

If you want to update to a newer version after downloading, you will need to download and unzip a new ZIP file.

Cloning will make a local copy of the git repository on your local drive. You can then use the desktop version of GitHub, or any git client to maintain the repository. This is a good option if you want to keep up to date with the latest developments in MiServer and/or participate in the MiServer community by submitting your code for consideration of inclusion in the GitHub repository. If you choose to clone, skip to Install MiServer 3.0.

## Install MiServer 3.0

Whether you have a zip file or have decided to clone the GitHub repository, you'll need to select a folder in which to install MiServer.



How you plan to use MiServer may influence where you install it.

If you're planning on running MiServer locally for your own use, or as a development environment, good locations include:

* Your desktop folder – the miserver workspace will be easily clickable
* Your Documents folder
* A folder off of the root folder (e.g. c:\miserver\)

If you're planning to host this installation of MiServer

* A folder off of the root folder (e.g. c:\miserver\)



It is NOT recommended that you install MiServer where Dyalog APL is installed. The sample MiSites (MiServer web sites) are installed under the MiServer folder, but you should install your own sites elsewhere as it will become problematic when you update your version of MiServer in the future.

For the remainder of this document we'll use "c:\miserver3\" as the place where we've installed MiServer. So, substitute your actual location whenever you see "c:\miserver3\"

## Start MiServer 3.0

1. Load the miserver workspace

**)load c:\miserver3\miserver**

c:\miserver3\miserver.dws saved...

Start 'MS3' ⍝ Run the MiServer v3.0 sample site

1. Start MiServer

Cursor up, insert a 1 before Start and press enter

**1 Start 'MS3'** ⍝ Run the MiServer v3.0 sample site   
Development environment loaded

MiSite "c:\miserver3\MS3" loaded

MiServer for "MS3" started on port: 8080 Running in Debug mode (configured by setting <Production> in /Config/Server.xml)

What's all this mean?

* Development environment loaded  
  Providing Start a left argument of 1 loads the development environment which loads all the classes needed to be able to edit pages from the root of the workspace.
* MiSite "c:\miserver3\MS3" loaded  
  The right argument to Start is the path to your website. If you specify 'MS3' (a relative path), MiServer starts its built-in website located in the /MS3 folder.
* MiServer for "MS3" started on port: 8080   
  This means that MiServer started successfully and listening on port 8080. The port is configurable.
* Running in Debug mode (configured by setting <Production> in /Config/Server.xml)  
  Debug mode was added to MiServer 3.0 to make it easier recover and carry on when an error occurs - generally when some invalid data was passed to a widget.

1. Use the browser of your choice to navigate to **http://localhost:8080**

## A Few Terms

As we talk about various MiServer concepts, it's helpful to know a few of these terms.

|  |  |
| --- | --- |
| **MiSite** | a MiServer-based web site |
| **MiPage** | a MiServer-based web page also the name of the base class for MiServer-based web pages |
| **Template** | a class based on MiPage that contains formatting or other enhancements |
| **Widget Control** | small programs that you can embed in your MiPage, you can use these terms more or less interchangeably. |
| **Element** | refers to a base HTML5 tag |
| **API** | Application Programming Interface – the protocol for how to talk to a widget |
| **HTTPRequest** | class which models an HTTP request  HTTP requests are how your browser communicates with MiServer. |

## Creating a MiSite

A MiSite is a MiServer website. MiServer 3.0 has a built-in MiSite, MS3, which contains examples, samples and documentation. MS3 is a fully functional MiSite, however, it is likely more complex in scope and structure than most users will need.

### Recommended Folder Structure for a MiSite

A MiSite can be installed in a single folder, but we recommend that you adopt a folder structure similar to this:

MiSiteRoot

contains your MiPages, additional folders can be used to organize a large MiSite

\Code

contains any application code you want loaded when the MiSite starts

\Templates

contains any MiPage templates to give your MiSite a specific look and feel

\Config

contains site-level configuration settings

\Styles

contains CSS stylesheets for your MiSite

\Images

contains images used in your styles

### Recommended Location for Your MiSite

Since a MiSite is just a folder, you can put it pretty much anywhere. However, two places that we discourage installing under either the MiServer folder, or the program files folder where Dyalog APL is stored.

### Sample MiSite

A bare-bones sample MiSite can be found in the StarterMiSite folder under the MiServer\SampleMiSites folder. You can copy this folder in its entirety to a new location in order to give yourself a starting point for your own MiSite.

## Building MiPages – Basic Concepts

A MiPage is MiServer web page. It's a Dyalog APL class that is based on the MiPage class.

A MiPage has to follow a few simple rules:

1. It has to ultimately be based on the MiPage class
2. It has to have a public method called Compose
3. If the page uses callbacks, the callback function must be named either APLJax or the name specified in the callback definition. A page can have multiple callback functions, one of which could be APLJax.

The following example is a valid MiPage.

:Class mymipage : MiPageTemplate ⍝ template based on the MiPage class

∇ Compose;h

:Access public

h←Add \_.h3'Welcome to MiServer 3.0!'

h.On'mouseover' 'Callback'

∇

∇ r←Callback

:Access public

r←Execute \_JSS.Alert'Hi'

∇

:EndClass

NB: this page is based on “MiPageTemplate” – a class found in Code\Templates which is based on MiPage. (So “ultimately” in the first rule is fulfilled through the multiple levels of inheritance.)

## Widgets, Controls, and Elements

MiServer implements a number of classes to provide the APL user an easy way to add content to a MiPage. Currently there are APIs defined for all HTML5 elements, many Syncfusion and jQuery widgets, and a number of Dyalog-developed controls that were developed with the APLer in mind.

All of the widgets, controls, and elements can be accessed from the \_ (underscore) namespace. We found that using \_, coupled with the convenience of Dyalog APL's autocomplete to be a great way to enter control names. There are a few naming conventions to help you know which type of control you're looking at:

|  |  |
| --- | --- |
| If the control name... | Then it's a... |
| is all lowercase letter (e.g. div, pre, table) | Base HTML5 element Why? HTML5 recommends all element names be lower case |
| begins with an uppercase letter (e.g. RadioButtonGroup, Table) | Dyalog developed control |
| begins with ej | Syncfusion widget  Why ej? Because that's what Syncfusion uses – for enterprise javascript |
| begins with jq | jQuery widget  Why jq? We chose the prefix to avoid name conflicts with other widgets. |

## Writing MiPage Content

The basic way to add content to a MiPage is to use the Add method.

Add 'Hello'

Adds the string "Hello" to your page.

Add \_.pre 'Some text'

Adds a <pre> (HTML5 preformatted) element with the content "Some text" to your page.

You can add just about any content to your page and MiServer will render it correctly.

The left argument to Add allows you to specify the id, class, and other attributes easily in a single statement using the following rules:

* If the left argument is a simple string, treat it as the id for the element, unless it begins with a . (period), in which case treat it as a class. If the first character is a #, it is forced to be treated as an id.
* If the left argument consists of two simple strings (neither of which contain the equals sign =) these two strings will be treated as name of attribute and value.
* Elements that are obviously paired, either by an equals sign or nesting are treated as attribute name/value pairs.
* Everything else is treated as an attribute.

The shy result of the Add-function is the object that was created (and added), so multiple calls to Add can be “chained”.

For the following syntax-examples we’ll assume z←⎕new MiPage. The Render-function can be applied to any element of a page or a page itself and creates the HTML -Representation of these objects. The samples also introduce the “Render”-function that is part of the HTMLElement-Class that everything in MiServer is based upon – even a MiPage is based on it. “Render” creates the HTML necessary to display any HTMLElement in the browser.

('abc' z.Add \_.div 'content').Render

<div id="abc">content</div>

('abc' 'def' z.Add \_.div 'content').Render

<div abc="def">content</div>

('.abc' 'def' z.Add \_.div 'content').Render

<div class="abc" def="def">content</div>

('#abc' 'def' z.Add \_.div 'content').Render

<div id="abc" def="def">content</div>

('id1' ('abc' 'def') z.Add \_.div 'content').Render

<div id="id1" abc="def">content</div>

('id1' ('abc=def ghi=jkl') z.Add \_.div 'content').Render

<div id="id1" abc="def" ghi="jkl">content</div>

### Add-Pitfalls

The following code

('abc'z.Add \_.div'Hello ').Add \_.b 'world!'

demonstrates the chaining mentioned above, but note that in order to demonstrate this (by calling Render), we need to capture the result of the first call to Add and render that:

(all←'abc'z.Add \_.div'Hello ').Add \_.b 'world!' ⋄ all.Render

<div id="abc">Hello <b>world!</b></div>

Simply appending .Render to the chained command will only render the last control:

(('abc'z.Add \_.div'Hello ').Add \_.b 'world!').Render

<b>world!</b>

## Putting it all together - your first Misite and .mipage

Let’s try to go through the exercise of building a small “HelloWorld”-site. So, for a start, create a HelloWorld-site under c:\MiSites and copy SamplesMiSites/StarterMiSite into it.

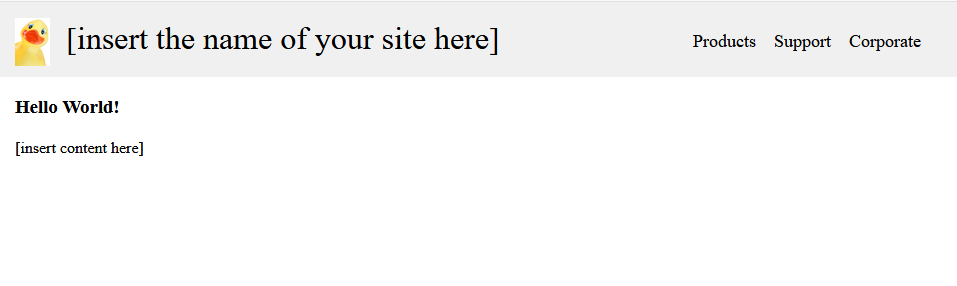
After an )xload of C:\miserver3\miserver.dws, please do

Start'c:\MiSites\HelloWorld'

Dyalog will respond with

MiServer for "c:\MiSites\HelloWorld" started on http://{YourIP}:8080  
Running in Debug mode (configured by setting in /Config/Server.xml)

Ctrl-click on the URL in the output to open the site in your browser. You should see something like this:



So, what happened?

* Note that the URL you clicked on “opened the site” (in a way). This is similar to what you do when accessing [www.dyalog.com](http://www.dyalog.com) – we just call the domain, but do not specify a specific page to open. Since Miserver (like any other webserver) can not just open any page and assume it’ll be the right one, it has a configureable setting for a default page that it opens (index.mipage), so it displays the content of that page.
* Click through the various menu-items and you will see that some content of the page changes whereas the top regions remains more or less unchanged.

### Purpose of the template

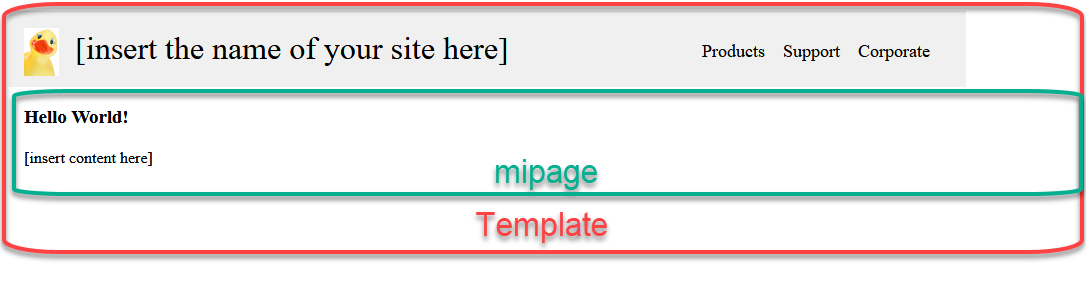
Let’s now examine that index-page. In your APL-Session, type

)ed c:\MiSites\HelloWorld\index.mipage

Let’s examine that code:

|  |  |
| --- | --- |
| :Class index : MiPageTemplate | “index” corresponds to the filename – this should always be the case! The class is based on the site-template. |
| ∇ Compose | The Compose-fn **must** be present! |
| :Access public | And have access public. |
| Add \_.h3'Hello World!' | Outputs some text as a heading (level 3) |
| Add \_.p'[insert content here]' | …and even more text in a paragraph |
| ∇ | And that’s all! |
| :EndClass |  |

Obviously there were more things shown on the screen than we could see in that index-page. That other content came from the template – and the design is such that the template establishes a sort of frame which wraps around the content of individual pages:



The template for your site is in file c:/MiSites/HelloWorld/Code/Templates/MiPageTemplate.dyalog, so let’s also examine that file:

h:\Temp\SNAGHTML4eaea3.PNG

:Class MiPageTemplate : #.MiPage

⍝ This is a template that adds a consistent header to all pages based on it

h:\Temp\SNAGHTML508836.PNG

∇ {r}←Wrap;lang;header;home;menu

:Access Public

⍝ set the tab/window title to the name of the application defined in Config/Server.xml

Add \_.title \_Request.Server.Config.Name

⍝ add a link to our CSS stylesheet

\_CssOverride←'/Styles/style.css'

⍝ wrap the content of the body element in two divs

'#contentblock'Body.Push \_.div

⍝ add the header to the top of the page

header←'#banner'Insert \_.header

home←header.Add \_.A'[insert the name of your site here]' '/' ⍝ click to go home

'src="/Styles/Images/duck.png"'home.Insert \_.img ⍝ logo

menu←'#menu'header.Add \_.nav

{menu.Add \_.A ⍵('/',⍵)}¨'Products' 'Support' 'Corporate'

⍝ Add a bar under the menu item that is currently open

Add \_.style('#menu a[href="',\_Request.OrigPage,'"] {border-bottom: 0.25em solid;}')

⍝ call the base class Wrap function

r←⎕BASE.Wrap

∇

:EndClass

Some comments:

1. As discussed previously, every page needs to be ultimately based on MiPage, so we have index → MiPageTemplate → MiPage.
2. In a template, the key-function is “Wrap”. As seen in this example, it usually sets various general settings and adds the “red” content (as seen on previous image).

## Organizing code / utilities

* When writing an application, you usually have common functions that you wish to call from various pages. Such common code should be organized into .dyalog-files that define Classes or Namespaces, the files should be placed in the {MiSites}\Code-Folder – everything in the Code-Folder will be loaded when Start’MiSite’ is executed, so it will always be present.
* Utilities that you’d like to call in a .mipage can also embedded into the template – this allows easy access (without having to specify long namespace-paths).
* MiServer comes with a load of useful tools in its “Utils”-folder. Again, all of these will be loaded on start. Often used tools (remove multiple blanks etc.) can be found in these libraries, so we recommend searching there before re-inventing the wheel.

## Adding functionality

As soon your page’s functionality goes beyond “Hello World”, you may start to wonder how to implement these things. In general, there are two ways a user may interact with any website:

1. by filling in forms with various input-controls and finally submitting the form, so that the app processes all input and returns a result
2. by manipulating one control after the other and having the page respond to these changes immediately.

Which method you choose depends on the type of application you are designing, but also on the type of audience you are addressing, i.e. option 2 creates more traffic between your application and the user’s browser – so users with slow internet connectivity may perceive your application as “slow” whereas intranet-users may view exactly the same app as being very responsive. Applications of type 2 are often referred to using the synonym “Ajax” (asynchronous Javascript and XML), in the context of MiServer we like to call that APLJax. The basic idea is that you define events (often “change” or “update”) about which you want to be notified – you can then assign an “Event Handler” which needs to respond with some Javascript-Code (don’t fret, MiServer has you covered for a long way there, so you may not need to write any Javascript-code).

Finally, users interact with your site through controls you added to the MiPage, so how do you find controls? We recommend to always have one instance of MiServer running the MS3-Site on your PC, because we think this app provides useful features to learn more:

* the first tab “Introduction” provides linkes to additional resources about creating sites and content, and also advanced topics. You may find useful additional information there after having worked through this document.
* The “Tutorial” (still in its infancy) illustrates solutions to common problems of user interaction and page design
* “Samples” and “Controls” illustrate the use of the approx.. 250 controls contained in MiServer. The “Search”-Box of that table can be used to reduce the list to matching entries, so we hope it makes that large list useable.

(BTW: If you’d like to share any ideas, comments or thoughts you had during your MiServer learning experience, please share them with the team at [miserver@dyalog.com](mailto:miserver@dyalog.com)!)

By clicking on the links in that table, you get access to dedicated samples pages for a control. Note that every sample page (through the MS3-template!) has a common menu bar on the top which has two always visible entries: “Rendered webpage” and “APL code” – use these to toggle between source code and output – so you can easily see how to use the various controls!

## Event Handling – Specifying Handers

**Using an object's On method**

|  |  |
| --- | --- |
| **object ← object.On Events {Callback} {ClientData} {JavaScript}** | |
| **object** | reference to the object to which the handler is attached On returns the same reference |
| **Events** | space delimited vectors of event names to handle |
| **Callback** | name of the callback function to execute If omitted, 'APLJax' is assumed |
| **ClientData** | specifies what data to pass back to the server from the client |
| **JavaScript** | JavaScript to execute prior in client prior to making AJAX call back to server |

**Example:  
 (Add \_.div).On 'click' 'myCallback'**

**Adding a Handler to the page**

|  |  |
| --- | --- |
| **h ← Add handler   {Selectors}{Events}{Callback}{ClientData}{Delegates}{JavaScript}{Page}** | |
| **h.Selectors** | jQuery/CSS selector of the elements to which to bind the handler |
| **h.Events** | space delimited vectors of event names to handle |
| **h.Callback** | name of the callback function to execute If omitted, 'APLJax' is assumed |
| **h.ClientData** | specifies what data to pass back to the server from the client |
| **h.Delegates** | "subordinate" selector for elements that are either dynamically created or too numerous to efficiently bind individual handlers |
| **h.JavaScript** | JavaScript to execute prior in client prior to making AJAX call back to server |
| **h.Page** | the page to which to send the AJAX request (defaults to "this" page) |

**Example:  
 h←Add Handler ⍝ add an event handler  
 h.Callback←'Calc' ⍝ specify the callback function to run  
 h.Events←'change' ⍝ listen for the "change" event  
 h.Selectors←'#mtg input' ⍝ on input elements in the element with id "mtg"  
  
 Add Handler ('#mtg input' 'change' 'Calc')**

## ****Event Handling - ClientData****

**By default the callback mechanism will return:**

**\_event the name of the event  
\_what the id/name of the element that triggered the event**

**\_value the value of the element that triggered the event (if a value exists)**

**\_selector the selector of the handler  
any form data that is on the page is serialized and returned using the names of the form input elements.**

**You can specify other information to be sent to the server:**

|  |  |
| --- | --- |
| **name {selector} {type} {which}** | |
| **name** | the name to give the data on the server side |
| **selector** | jQuery/CSS selector of the element from which to get the data if omitted, use the element to which the handler is bound |
| **type** | the type of data to return. valid types include:   |  |  | | --- | --- | | **type =** | **returns** | | **attr** | an HTML attribute | | **css** | a CSS setting | | **html** | the HTML content | | **is** | specific settings – see jQuery.is() | | **val** | the value of the element | | **eval** | the result of the evaluation of a JavaScript string | | **string** | constant string | | **event** | jQuery event object | | **ui** | jQuery ui object | | **ejModel** | SyncFusion model object | | **argument** | SyncFusion argument object | | **serialize** | all data for a form (unnecessary if there is only a single form on the page) | |
| **which** | dependent on **type**   |  |  |  | | --- | --- | --- | | **type =** | **which =** | **Example** | | **attr** | the attribute to return | 'attr' 'title' | | **css** | the CSS setting to return | 'css' 'font' | | **html** | '' | 'html' | | **is** | the setting to return – see jQuery.is() | 'is' ':checked' | | **val** | '' | 'val' | | **eval** | the JavaScript string to evaluate | 'eval' '2+2' | | **string** | the string to return | 'string' 'constant' | | **event** | the element of the event object | see jQueryUI document | | **ui** | the element of the ui object | see jQueryUI document | | **model** | the element of the model object | see Syncfusion document | | **argument** | the element of the argument object | see Syncfusion document | | **serialize** | '' | 'serialize' | |

**Example:**

**h←Add Handler ⍝ add an event handler  
h.ClientData←('content' '#div1' 'html')**

**('color' '#div2' 'css' 'background-color')  
  
Returns**

* **a variable named "content" which contains the HTML content of the element with id "div1"**
* **a variable named "color" with the background color setting of the element with id "div2"**

### Event Handling - Retrieving Client Data from Callback Functions

Client data can be retrieved in two basic ways:

* Define a public field in your MiPage with the same name as the data you want to retrieve  
  For instance, in the specification used above  
   h.ClientData←('content' '#div1' 'html')  
   ('color' '#div2' 'css' 'background-color')  
  You could defined two fields and they would be populated automatically.  
   :field public content  
   :field public color  
    
  When using this method, it is generally important to clear the value of the fields after sending the response to the client so that subsequent requests don't reuse the same data.
* The other way is to use the Get method to retrieve the data by name. The left argument to Get is the default value to use if the data element is not found. Again, using the example from above, the following could be used.  
   '???' Get 'content'  
   'white' Get 'color'

## Event Handling – Sending Responses Back to the Client

There are four functions which specify actions to be taken on the client side in response to a callback function.

|  |  |
| --- | --- |
| **r ← selector Replace new r ← selector Append new r ← selector Prepend new r ← Execute javascript** | |
| **Replace** | Replaces the HTML content of the element specified by selector with new |
| **Append** | Appends new to the HTML content of the element specified by selector |
| **Prepend** | Prepends new to the HTML content of the element specified by selector |
| **Execute** | Executes javascript string (using JavaScript's eval() function) |
| **selector** | The selector of the elements to update |
| **new** | The new content with which to update |
| **javascript** | A character vector of the JavaScript to execute in the client |

**Example:**

r ← '#result' Replace \_html.h2('Hi')  
 r,← Execute 'alert("Happy Birthday!")'

**Callback functions must return a result, though the result could be '' if no action is to be taken on the client side.**

## Function Reference

### #.Start

|  |  |
| --- | --- |
| **{dev} #.Start MiSitePath** | |
| Start | starts MiServer |
| MiSitePath | the path containing your MiSite, if it's a relative path, it is relative to the miserver workspace location |
| dev | Boolean indicating whether to load the development environment  The development environment loads all the classes necessary to edit MiPages from root (#). |

## Configuration Settings

MiServer stores its configuration settings in a number of XML files in the /Config/ folder.

MiServer uses a 2-tiered configuration system – one at the server level and one at the MiSite level.

The server level settings are comprehensive – every configuration setting is set.

MiSite level configuration settings override their server level counterparts.

A MiSite can run without any MiSite level configuration (it just uses the server level settings).

This gives you a few options:

1. Run with the server level settings – this is generally appropriate when you're running a single MiSite and you don't mind its settings possibly being overwritten when you update MiServer.
2. Run with mostly server level setting, overriding only those you care about at the MiSite level. This allows you to tailor your MiSite behavior, yet still pick up updates to server level setting that you haven't overridden.
3. Copy the server level configuration files to your MiSite. This allows you complete control to change whatever settings you care to, and protects you from settings being changed.

Note that with any of these options, new configuration settings introduced in a MiServer update will still be created at the server level when you update.

### The Configuration Settings You Probably Care About

There are scores of configuration settings in MiServer, most of which you may never change. These will be comprehensively documented in the full MiServer User Guide. For this Quick Start Guide, here are the settings that you'll probably want to know about:

|  |  |  |
| --- | --- | --- |
| **File** | **Setting** | **Significance to you** |
| Server.xml | Port | The port which MiServer listens on. You'll probably want to change this to 80 if you a MiServer visible to the internet. You'll also want to change it if you care to run multiple instances of MiServer on the same machine. |
| Server.xml | ClassName | The class which implements core server functions. Normally, this is "MiServer". However, if you want to customize behavior on server startup or shutdown, or write an application server that interfaces with some business logic components, you'll want to change this. Note: Changing this setting means you'll need to write a class, derived from the MiServer class, that has the same name as the setting you provide. |
| Server.xml | Production | This is set to 0 by default thus enabling MiServer's debugging framework. This should be set to 1 for a production level server. |
| Server.xml | SessionTimeOut | This specifies the number of minutes to allow a client session to remain idle before it's considered timed out and closed. |
| Server.xml | TrapErrors | This is normally set to 0 to allow the developer to interactively debug errors. Set it to 1 on a production server and errors will be logged using the DrA error trapping framework. |