MiServer 3.0
Quick Start Guide

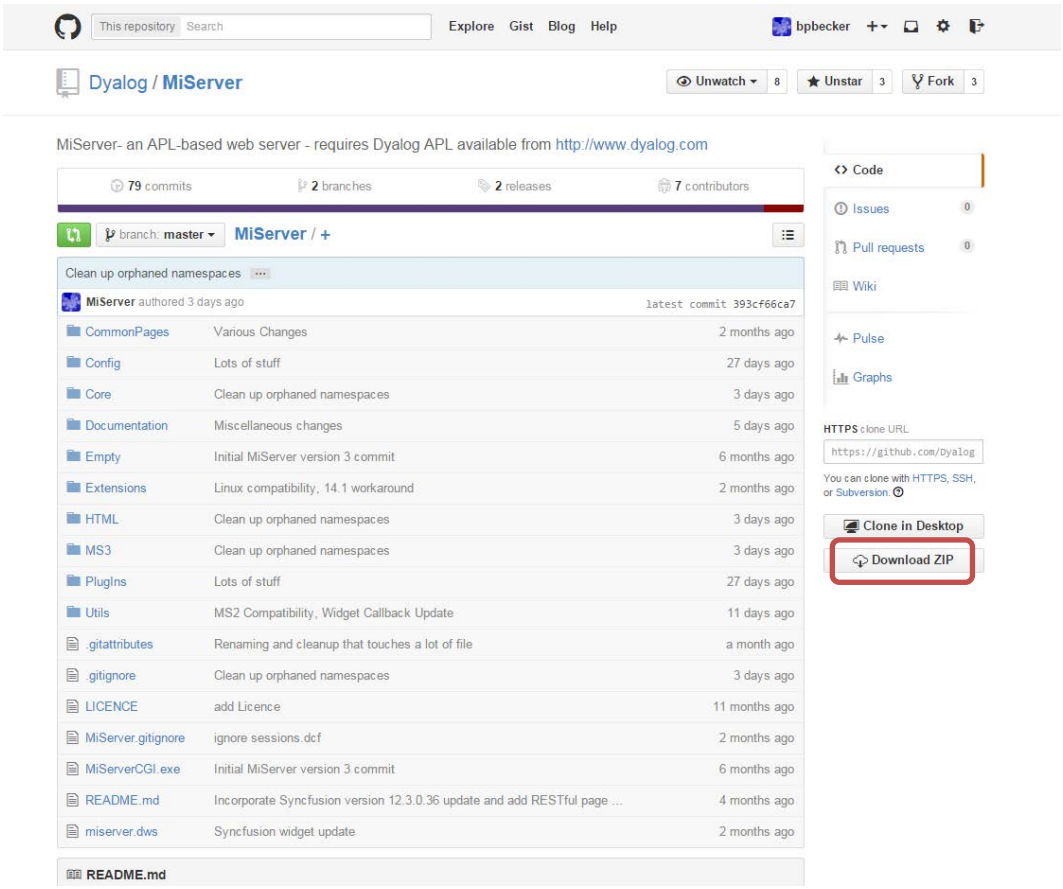
September 2015

Table of Contents

Download MiServer 3.0

You can download MiServer by

1. going to **https://github.com/Dyalog/MiServer** and
2. clicking the "Download Zip" button



TIP

There is also the option to clone the MiServer 3.0 repository in the Desktop version of Git. 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 folders include:

- Your desktop folder – the miserver workspace will be easily clickable
- Your My 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.

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
```

2) 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.

3) 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 not 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 MiSite folder under the MiServer folder. You can copy the MiSite 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
```

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 =)
- 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.

```
      ('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>
```

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:

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

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.