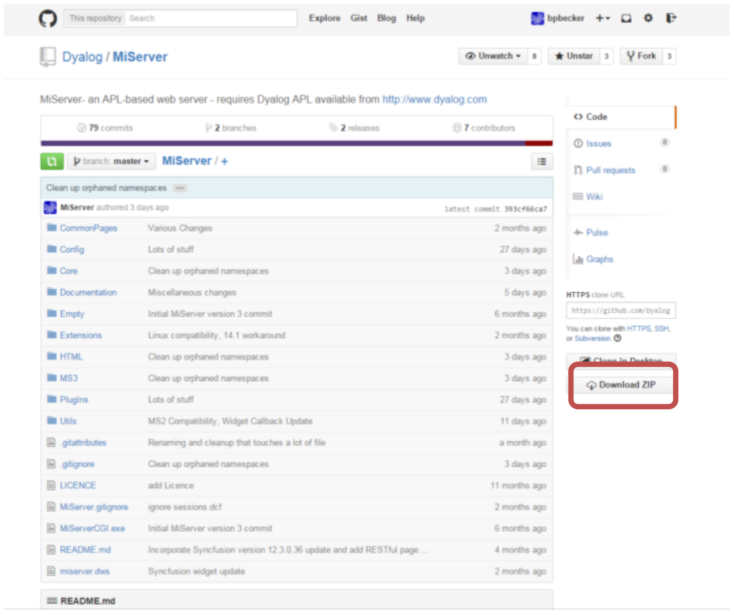
MiServer 3.0 Workshop Survival Guide

DYNA

April 2015 - Princeton, USA

Installing and Running MiServer 3.0

1) Download the latest MiServer by going to
   https://github.com/Dyalog/MiServer and clicking the "Download Zip" button
   The zip file is also available on flash drives from the instructors



MiServer- an APL-based web server - requires Dyalog APL available from http://www.dyalog.com

Clean up orphaned namespaces

MiServer authored 3 days ago — latest commit 393cf66ca7

CommonPages	Various Changes	2 months ago
Config	Lots of stuff	27 days ago
Core	Clean up orphaned namespaces	3 days ago
Documentation	Miscellaneous changes	5 days ago
Empty	Initial MiServer version 3 commit	6 months ago
Extensions	Linux compatibility, 14.1 workaround	2 months ago
HTML	Clean up orphaned namespaces	3 days ago
MS3	Clean up orphaned namespaces	3 days ago
Plugins	Lots of stuff	27 days ago
Utils	MS2 Compatibility, Widget Callback Update	11 days ago
.gitattributes	Renaming and cleanup that touches a lot of file	a month ago
.gitignore	Clean up orphaned namespaces	3 days ago
LICENCE	add Licence	11 months ago
MiServer.gitignore	ignore sessions.dcf	2 months ago
MiServerCGI.exe	Initial MiServer version 3 commit	6 months ago
README.md	Incorporate Syncfusion version 12.3.0.36 update and add RESTful page ...	4 months ago
miserver.dws	Syncfusion widget update	2 months ago

2) Extract the zip file to a location of your choosing.
   For the purposes of this document and workshop we'll use "c:\dynams3\"
   So, substitute your location whenever you see "c:\dynams3\"

3) Load the miserver workspace and start the "dyna" MiSite

```
      )load c:\dynams3\miserver
c:\dynams3\miserver saved...
      Start 'MS3'         ⍝ Run the MiServer v3.0 sample site
      )fns
Load    Restart Start   Stop
      1 Start 'dyna'      ⍝ Start the dyna MiSite
```

{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

4) Use the browser of your choice to navigate to **http://localhost:8080**

A Few 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
`EAWC`	a template to make all widgets and HTML elements available without namespace prefixes
`Widget`	a small program that you can embed in your web page
`API`	Application Programming Interface – the protocol for how to talk to a widget
`Level 0 API`	A class exists that can be called from your MiPage
`Level 1 API`	Includes all necessary linked files
`Level 2 API`	Can generate any HTML infrastructure when rendered
`Level 3 API`	Implements widget features in a manner more "natural" to use in APL
`HTTPRequest`	class which models an HTTP Request

Important `MiPage` Class Elements and Methods

`Elements`	
`_Request`	the current HTTPRequest that's being serviced (see HTTPRequest later in this document)
`_event`	used in callbacks – the event name that was triggered
`_what`	used in callbacks – the id for the element that triggered the event
`_PageData`	namespace that contains any data provided by the request
`OnLoad`	string containing JavaScript to execute upon page load (analogous to ⎕LX)
`Methods`	
`{proto} Get names`	
`Get`	retrieve data from _PageData
`names`	space-delimited vector of names to retrieve
`proto`	prototype to return if name is not found, also determines datatype (character or numeric)

Important `HTTPRequest` Class Elements and Methods

Elements	
Input	the current HTTPRequest that's being serviced (see HTTPRequest later in this document)
Headers	HTTP headers sent with the request
Command	the HTTP command (generally GET or POST)
Filename	the filename specified in the request
PeerAddr	the IP address of the client
Cookies	any cookies included in the request
Response	Namespace containing response to the HTTP Request
Session	If using sessioning (default is yes) this is a reference to the session namespace
Methods	
r←GetCookie name	
GetCookie	retrieve cookie value
name	name of the cookie to retrieve
r	character string value of the cookie ('' if the name was not found)
SetCookie name value path days	
SetCookie	sets the value of a cookie in the client's browser
name	name of the cookie
value	value to be set
path	in which to put the cookie
days	number of days before the cookie expires
DelCookie name path	
DelCookie	deletes a cookie (by setting its expiry date in the past)
name	name of the cookie
path	path where the cookie is found

Working with Elements and Widgets and Snippets, Oh My!

Just about every object that gets rendered in your browser, be it an HTML element, a widget, a script, even the web page itself, is derived from the HtmlElement base class.

Note: if your page is based on the EAWC class, you do not need to specify the namespace for an element/widget.

Namespace	**Contains**
_html	all "raw" HTML5 elements – all lower case names
_HTML	Dyalog "enhanced" elements based on similarly named html entries, except the HTML name begins with a capital letter e.g. `_HTML.Table` is the enhanced version of `_html.table`
_JQ	Contains jQuery widgets
_SF	Contains Syncfusion widgets
_DC	Contains Dyalog developed widgets
_JSS	Not based on HtmlElement, but contains utilities to generate useful JavaScript snippets

Creating/Editing Pages

The easiest way is to copy and modify an existing page, but, if you want to build your own from scratch...

A page should have the follow characteristics:

- be derived from either the `MiPage` or `RESTful` class
- contain a public method named `Render`
- all methods which process callbacks must be public
- any callbacks which do not use named callback functions will be processed by the `APLJax` method, provided it exists and is public

```
:Class myFirstPage : DyNApage
    ∇ Render
      :Access public
      Add h2'Hello'
    ∇
:EndClass
```

To save your page. use the `]save` user command

```
      ]save myFirstPage c:\mysite\
```

will save the definition of `myFirstPage` in the c:\mysite\ folder.

To modify your page the development environment must be loaded either using the `Load` function in the miserver workspace, or calling the `Start` function with a left argument of 1, then:

```
      ]load c:\misite\myFirstPage
```

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

any form data that is on the page is serialized and returned using the names of the form input elements.

`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()	
	`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'`
	`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
	`ejModel`	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 – 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 Prepent 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.

RESTful-style Web Services

A page which implements a RESTful web service:

- must be based on the `RESTful` template
- have a public `Render` method that returns a result
- may return any string as its result, e.g. JSON, XML, HTML, text