

 **The tool of thought for expert programming**

|  |
| --- |
| pdfgreyline  |

Dyalog™ for Windows

**MiServer User Guide**

**Version 2.0**

Dyalog Limited

Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

|  |  |
| --- | --- |
| tel: | National (01256) 830030 |
| International +44 1256 830030 |

fax: +44 (0)1256 830031
email: support@dyalog.com
http://www.dyalog.com

*Dyalog is a trademark of Dyalog Limited*

*Copyright © 1982-2011 by Dyalog Limited.*

*All rights reserved.*

*Version 2.0*

***First Edition August 2011***

*No part of this publication may be reproduced in any form by any means without the prior written permission of Dyalog Limited.*

*Dyalog Limited makes no representations or warranties with respect to the contents hereof, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Dyalog Limited reserves the right to revise this publication without notification.*

*TRADEMARKS:*

*Unix is a trademark of X/Open Ltd.*

*Linux is a trademark of Linus Torvalds.*

*Windows is a trademark of Microsoft Corporation.*

*Intel and Core are trademarks of Intel Corporation*

*All other trademarks and copyrights are acknowledged.*

**Contents**

[An APL Window to the Internet 1](#_Toc304986442)

[1.1 Web Server Fundamentals 1](#_Toc304986443)

[1.2 Introducing MiServer 1](#_Toc304986444)

[1.3 Why MiServer? 2](#_Toc304986445)

[MiServer Architecture 3](#_Toc304986446)

[2.1 Web Servers and Sites and Pages, Oh My! 3](#_Toc304986447)

[MiServer Core Functionality 3](#_Toc304986448)

[MiServer Skins 3](#_Toc304986449)

[The MiSite 3](#_Toc304986450)

[The MiPage 3](#_Toc304986451)

[2.2 What you’ll need to know to build MiSites 4](#_Toc304986452)

[A Bit of OO 4](#_Toc304986453)

[A Bit of HTML 4](#_Toc304986454)

[XHTML 5](#_Toc304986455)

[Scripted files 5](#_Toc304986456)

[2.3 Directory Structures 5](#_Toc304986457)

[MiServer 5](#_Toc304986458)

[MiSite Template 7](#_Toc304986459)

[Getting Started 8](#_Toc304986460)

[3.1 Installing and Running MiServer 8](#_Toc304986461)

[Prerequisites 8](#_Toc304986462)

[Installation 8](#_Toc304986463)

[3.2 Server Configuration 8](#_Toc304986464)

[Server.xml Parameters 9](#_Toc304986465)

[3.3 Testing MiServer 9](#_Toc304986466)

[Your First MiSite 11](#_Toc304986467)

[4.1 Copying and Configuring the MiSite Template 11](#_Toc304986468)

[You’re ready to go 11](#_Toc304986469)

[4.2 MiPages 11](#_Toc304986470)

[Requirements of a MiPage 11](#_Toc304986471)

[The Index Page 11](#_Toc304986472)

[4.3 Building Your First MiPage 12](#_Toc304986473)

[Building a basic page 12](#_Toc304986474)

[Where is the extra HTML coming from? 12](#_Toc304986475)

[4.4 Getting Tricky 13](#_Toc304986476)

[Reverse.dyalog 13](#_Toc304986477)

[Getting Information from the Browser 14](#_Toc304986478)

[Forms: Quick and Dirty 15](#_Toc304986479)

[Getting Stylish 16](#_Toc304986480)

[5.1 Styles 16](#_Toc304986481)

[Style Basics 16](#_Toc304986482)

[5.2 In-Line Styles 16](#_Toc304986483)

[5.3 Cascading Style Sheets 17](#_Toc304986484)

[Selectors 17](#_Toc304986485)

[Associating CSS with a web page 17](#_Toc304986486)

[External CSS Sheets 18](#_Toc304986487)

[5.4 Inserting Styles 18](#_Toc304986488)

[Under the Covers 20](#_Toc304986489)

[6.1 Going Deeper 20](#_Toc304986490)

[6.2 Browser to Server Communication 20](#_Toc304986491)

[GET and POST HTTP Requests 20](#_Toc304986492)

[Sending Data Encoded HTTP Requests 20](#_Toc304986493)

[Forms 21](#_Toc304986494)

[Submitting a Form 22](#_Toc304986495)

[6.3 How MiServer Gets Client Data: HTTPRequest 22](#_Toc304986496)

[Customizing MiServer Behavior 23](#_Toc304986497)

[7.1 Limitless possibilities. 23](#_Toc304986498)

[7.2 MiServer Skins: Overriding MildServer Behaviour 23](#_Toc304986499)

[Session handling 23](#_Toc304986500)

[HTML Wrapping 23](#_Toc304986501)

[Error Handling 24](#_Toc304986502)

[Logging 24](#_Toc304986503)

[Cleanup 24](#_Toc304986504)

[A MiServer Skin: DemoServer 24](#_Toc304986505)

[7.2 Session Handling 24](#_Toc304986506)

[Cookies 25](#_Toc304986507)

[In-Session Data storage 25](#_Toc304986508)

[Access Control and Authentication 25](#_Toc304986509)

[Data Enabling Your MiSite 27](#_Toc304986510)

[8.1 Data Enabling 27](#_Toc304986511)

[8.2 Relational Databases 27](#_Toc304986512)

[Interacting with Relational Databases 27](#_Toc304986513)

[Setting up Datasources.xml 27](#_Toc304986514)

[SQL.ConnectTo, SQL.Do and SQL.CloseAll 28](#_Toc304986515)

[8.2 Displaying Data in your MiPage 29](#_Toc304986516)

[Displaying Table Data Using HTMLInput.Table 29](#_Toc304986517)

[JQ.Tablesorter 29](#_Toc304986518)

[Client Side Scripting: JQuery 31](#_Toc304986519)

[9.1 Client Side Scripting 31](#_Toc304986520)

[Widgets 31](#_Toc304986521)

[9.2 JavaScript and JQuery 32](#_Toc304986522)

[9.3 Using JQuery in Your MiSite 32](#_Toc304986523)

[What’s Needed to Use JQuery? 32](#_Toc304986524)

[The JQ Namespace 32](#_Toc304986525)

[Utilities Functional References 33](#_Toc304986526)

[HTML Namespace Functional Reference 33](#_Toc304986527)

[Functions 33](#_Toc304986528)

[HTMLInput Namespace Functional Reference 34](#_Toc304986529)

[Functions 34](#_Toc304986530)

[JQ Namespace Functional Reference 39](#_Toc304986531)

[Path Variables 39](#_Toc304986532)

[Functions 39](#_Toc304986533)

[Cover Functions 40](#_Toc304986534)

[SQL Namespace Functional Reference 42](#_Toc304986535)

[Functions 42](#_Toc304986536)

[HTTPRequest Reference 43](#_Toc304986537)

[The Request Object 43](#_Toc304986538)

[Parsing the HTTP Request 43](#_Toc304986539)

[Namespaces 43](#_Toc304986540)

[Functions 44](#_Toc304986541)

[The Future of MiServer 46](#_Toc304986542)

[The MiServer Project 46](#_Toc304986543)

[Directions for Development 46](#_Toc304986544)

# C H A P T E R 1

An APL Window to the Internet

### 1.1 Web Server Fundamentals

Since you’re reading this document, it’s probably safe to assume that you’ve used a web browser to access web pages before. You probably know that a web page is a resource residing somewhere on a computer. When you enter a web address (URL) into your browser’s address field your browser sends a request for that resource to the computer on which the resource should be stored. A process on that computer, called a web server, listens for requests, interprets them, tries to locate the requested resource and finally return it to the browser.

More specifically, a web server is a process that listens to a TCP/IP port for incoming connection requests. Once a connection has been established between the web server and a client, typically a web browser, communication begins between them according to a set of standards called the Hypertext Transfer Protocol (HTTP). When the server receives a request for a resource, it performs a number of preparations on that resource for transmission, responds back to the browser and closes the connection. If all goes well, the browser displays a web page.

Today’s servers often have a number of other useful capabilities. Servers can track individual clients through the use of sessions and cookies, saving data between requests. This supports features like shopping carts and multi-part forms. Many servers also support interactions with databases, user authentication, and resource specific access restrictions.

### 1.2 Introducing MiServer

MiServer is a web server implemented in Dyalog APL, which allows you to bring the power of APL to the web. It is a workspace with tools to facilitate the development and hosting of web pages from within Dyalog APL. The first version of MiServer was developed by Morten Kromberg, inspired by Stefano Lanzavecchia’s WildServer, which used the object oriented features of Dyalog APL v11.0 to create a highly modular APL web server. The goal of the MiServer project has been to create a simple framework that allows anyone who can write an APL function to build a web front end, without using any components that are not included in a standard installation of Dyalog APL.

The MiServer project has two main goals.

1. Make it possible for anyone who can write an APL function to turn it into hosted web content without having to learn much about the underlying nuances of web page implementation. That stated, the full breadth of HTML and related technologies remain at the developer’s disposal.
2. Allow code to be easily organized into modules. This would and form the basis for collaborative software development, specifically an “Open Source” project in APL. The architecture is intended to encourage knowledgeable users to write extensions, like support for additional user-interface “widgets”, database interfaces, or more sophisticated session management. More information about the MiServer Project can be found in Appendix III.

One note, the term “MiServer” is used in a number of contexts, including:

* The MiServer Project, an open source initiative involving MiServer.
* The MiServer class[[1]](#footnote-1) which implements the core web server functionality.
* An instance of the MiServer class, referred to as “a MiServer”

### Why MiServer?

There are a number of circumstances where you may want to use MiServer.

* APL is your preferred development environment.
* You want to host your existing APL functionality on a web page.
* You want to use a web browser as a user interface.
* You want to merge APL with the vast array of tools available to web developers.
* You want to integrate almost any Dyalog APL utility within your web content. Including:
	1. SQAPL: which provides access to any ODBC-compliant database
	2. SAWS: which consumes and hosts web services.
	3. SharpPlot: a business graphic package included with Dyalog APL.

MiServer is a stable web delivery environment. We expect to add new features in collaboration with the APL community. However, a more “industrial” web server such as Microsoft IIS, Apache, and WebSphere may have features that are currently not present in MiServer. You may opt to use one of these servers in one of these following situations:

* You need to use a third party technology like Adobe Cold Fusion.
* You need to use technology that integrates specifically with another web server.
* You require extensive trace and debugging capabilities.
* You desire to also run an SMTP or FTP server. Conga, Dyalog’s TCP/IP communications tool, contains sample servers of each type, but these are not integrated into MiServer at this time.

An alternative for enabling APL functionality to deliver web content is through ASP.NET as described in the Dyalog .NET Interface Guide. Finally, it should also be noted that it is possible to use the MiServer to deliver content which passes through a commercial web server framework like IIS or Apache, allowing you to combine the lightweight flexibility of APL with management and security features of the commercial web frameworks.

# C H A P T E R 2

MiServer Architecture

### 2.1 Web Servers and Sites and Pages, Oh My!

Browsers request resources, which can be almost any file type. If this resource is rendered as a user interface in the browser, the resource is called a web page. When you type a URL into a browser’s address bar, you are accessing a resource on a computer somewhere. A URL, like [www.dyalog.com](http://www.dyalog.com), references the root of a directory of such resources, called a website. A website can be as simple as a single file in a folder or can be an expansive collection of content and functionality to process and format that content. As discussed in Chapter 1, a web server hosts web sites and makes them available for request.

#### MiServer Core Functionality

MiServer is a web server. It has an architecture that is implemented across a number of files, requiring a specific directory structure which is described in Chapter 2.3.

The files representing the core functionality are found in the SiteRoot/Core/ directory, which contains:

* the MiServer class – this class implements the core functionality of the server.
* the HTTPRequest class – this class parses the HTTP request and generates the HTTP response.
* the Boot namespace – this namespace contains functionality to start and stop a MiServer.
* the MiPage class – this server as the base class for all MiPages.

#### MiServer Skins

In general, it should not be necessary to modify any of these files. MiServer is designed to make site specific server behaviour easy to implement by exploiting its object oriented nature and having a number of overridable methods which can be redefined by a class derived from the MiServer base class. This is called a MiServer Skin. A MiServer Skin can implement specific session behaviour, usage logging, error handling and an HTML “wrapper” to create a consistent look and feel for your website.

#### The MiSite

MiServer works by ‘booting’ an instance of itself that is associated with a particular directory. Contained within that directory must be a basic structure, including directories and files that without which MiServer will not work. Because its particular structure is unique to the MiServer environment, we’ve chosen to call these specifically formatted websites MiSites. A description of the MiSite directory structure can be found near the end of this chapter.

#### The MiPage

A MiPage is a .dyalog scripted file that contains an APL class derived from the MiPage base class. MiPages generate Hypertext Mark-up Language from APL code that is used to create webpages.

### 2.2 What you’ll need to know to build MiSites

To construct MiSites, you will need an understanding of basic Object Oriented programming concepts. To build MiPages and generate web pages, you will need to understand something about Hypertext Markup Language. Finally, you’ll need to know how to manage and edit APL scripted files.

#### A Bit of OO

Object oriented (OO) programming is a programming paradigm centered on structures called **objects**. Objects are independent **instances** of a **class**, an object blueprint which describes a set of related functions and/or data. Each object can contain unique data.

A class may derive from another class, referred to as a base class. The derived class acquires the methods, fields and properties of its base class on top of its own. In OO speak, this is called inheritance. A base class may specify a method as overridable, which means that a derived class can define its own behaviour for a method of the same name.

The elements of OO programming will not be unfamiliar to an APLer, but use different terminology. Functions are called methods. Variables are called either fields or properties. There are additional attributes that are specific to the OO versions of these elements not described here.

For more information on OO, check out the Introduction to OO in the Language Help of Dyalog APL and the Introduction to Object Oriented Programming for APL Programmers found in the documentation supplied with Dyalog APL or at http://docs.dyalog.com.

#### A Bit of HTML

Hypertext Markup Language (HTML) is the fundamental building block of web content. HTML uses ‘tags’ or key words surrounded by angle brackets to format content. Most tags operate in pairs, with an opening and closing tag surrounding a section of content. Web browsers take these tags and render them into a User Interface.

Here are a few tags that you might find useful as you dive into web development.

|  |  |
| --- | --- |
| <html> </hmtl> | Wraps all HTML documents.  |
| <head> </head> | Wraps around the unrendered section of the document that deals with preparations for page load and resource loading. |
| <body></body> | Wraps all HTML marked for page render.  |
| <br /> | Inserts a line break |
| <a href="http://www.dyalog.com">Dyalog</a> | Creates a hyperlink on the text surrounded by the tags. |
| <div> </div> | A tag that has no meaning on its own, a division |
| <img src="/images/dragon2011.png" /> | Displays an image stored within your site directory |
| <ul><li>Item1</li><li>Item2</li></ul> | An unordered list. |

An HTML tag can be defined by a number of ‘attributes’ contained within the leading tag. These attributes might be widely used among many tags, like the ‘name’ attribute, or specific to the functionality of the tag type, like the ‘size’ attribute of a text, password or file type input tag.

For more information, a comprehensive list of tags and best practices can be found at the W3 Schools at http://www.w3schools.com.

#### XHTML

All included toolsets which generate HTML conform to a more stringent standard called XHTML. A major issue with HTML how prolific bad code is on the internet. Browsers have to be versatile enough to accommodate a number of poor coding techniques. This generally makes the code difficult to read, and nearly impossible to be parsed by other programs. To deal with these issues, HTML 4.01 was blended with Extensable Markup Language, or XML, to create Extensable Hypertext Markup Language, or XHTML. Unlike HTML, XHTML elements must:

* Be properly nested.
* Always be closed
* Be in lowercase
* Have one root element.

It is important to, once you have a handle on the basics of HTML, to use the XHTML standards.

#### Scripted files

All of the classes and namespaces that compose MiServer, save the workspace itself, are kept on UTF-8 encoded files with a .dyalog extension. SALT, the Simple APL Library Toolkit, is a utility for importing scripted files. It is intended to provide a mechanism for APL users to develop and share code, which aligns nicely with the goal of MiServer as an open source project.

Any of these scripted files can be edited both from inside an APL session and from a text editor.

To bring a scripted class or namespace into your workspace, use ]Load.

###  2.3 Directory Structures

#### MiServer

This is the basic directory structure required of MiServer. While the files contained in SiteRoot\Core\ represent the basic functionality of the web server, the other files provide vital extensions to the server, as well as a number of tools used to in the processing of web pages.

|  |  |
| --- | --- |
| C:\MiServer\  | The root directory of MiServer |
|  Core\ | Contains the core components of MiServer |
|  Boot.dyalog | Namespace containing functions to start and stop MiServer |
|  HTTPRequest.dyalog | Class which encapsulates all information for an HTTP request |
|  MiPage.dyalog | Class which serves as a base class for all MiPages. |
|  MildServer.dyalog | Class which implements the MiServer core and serves as a base class for all MiServer Skins. |
|  Extensions\ | Folder for extensions to MiServer to implement additional functionality. |
|  SimpleAuth.dyalog | Implements basic HTTP authentication. |
|  SimpleSessions.dyalog | Implements stateful interaction using sessions. |
|  ContentEncoder.dyalog | An interface that content encoding schemes, like HTTP compression can be implemented using. |
|  deflate.dyalog | Implements the default compression style. |
|  Documentation\ | Documentation associated with the MiServer. |
|  ErrorPages\ | The pages sent to the browser during an error. |
|  Demo\ | A demonstration MiSite. |
|  MiSiteTemplate\ | A basic MiSite. |
|  PlugIns\ | Third party plug ins. |
|  JQuery\ | Files associated with the JQuery JavaScript library. |
|  css\ | Styles associated with JQueryUI objects. |
|  development-bundle\ | Contains the files for JQuery and JQueryUI. |
|  js\ | Contains all JavaScript files used in the current implementation of the JQ namespace. |
|  index.html | A test page for JavaScript. |
|  Utils\ | Utility classes and namespaces useful for generating HTML. |
|  Base64.dyalog | Functions for encoding and decoding messages in base 64. |
|  Dates.dyalog | Functions dealing with dates. |
|  DrA.dyalog | Error logging functions. |
|  Files.dyalog | Functions to manipulate files. |
|  HTML.dyalog | Functions to assist in the creation of HTML. |
|  HTMLInput.dyalog | Functions to assist in the creation of HTML, focused on form and input objects. |
|  JQ.dyalog | Functions that integrate the JQuery JavaScript library into your MiPages. |
|  SQL.dyalog | Functions that integrate the database interaction capacities of SQAPL into your MiPages. |
|  SMTPMail.dyalog | For sending mail messages via SMTP. |
|  Strings.dyalog | Functions for the handling of strings. |
|  XML.dyalog | Functions to convert XML into namespaces and vice versa. |
|  mserver.dws | The MiServer workspace. |

#### MiSite Template

MiServer comes with a sample MiSite in SiteRoot/Demo, this site contains a number of sample MiPages and configuration files to demonstrate some of the more powerful functions of the MiServer.

|  |  |
| --- | --- |
| C:\YourSiteRoot\ | The root of the site |
|  Admin\ | Contains .dyalog files that control configuration settings |
|  EditPage.dyalog | A page to demonstrate in-browser editing of UTF-8 files |
|  Code\ | Contains code to modify the server |
|  Demoserver.dyalog | A class derived from the ‘MiServer’ class that renders HTML for a web page. In the “Demo” site and the MiSite template, the file is named DemoServer.dyalog |
|  Config\ | A group of XML files used for site specific server configuration |
|  Server.xml | An XML file that sets the site specific configuration settings |
|  Users.xml | A list of users and passwords |
|  Groups.xml | A list of groups and permissions associated with the groups |
|  DrA\ | Contains error logs |
|  Styles\ | Contains Cascading Style Sheets |
|  error.css | Cascading style sheet for error pages |
|  style.css | General cascading style sheet for the site, referenced in the wrap method of DemoServer.dylog |
|  TempFiles\ | Where MiServer temporarily stores files |
|  Index.dyalog | The default page |
|  MiPageTemplate.dyalog | A prototype dynamic MiPage |

# C H A P T E R 3

Getting Started

### 3.1 Installing and Running MiServer

#### Prerequisites

MiServer requires Dyalog APL version 12.1 or later and version 2.2 Conga, Dyalog’s TCP/IP communications tool.

#### Installation

Download the files and unzip them into any directory. In all following demonstrations, MiServer is assumed to have been installed in C:\MiServer.

### 3.2 Server Configuration

When a MiServer is started, it expects to find a number of configuration files in the directory SiteRoot\Config\, including the site specific server configuration file Server.xml. MiServer references Server.xml start up, so any changes to the server configuration will not be applied until it is restarted. Edits can be made to an XML file from most text editors.

Server.xml[[2]](#footnote-2)

<Server>

 <Name>**MiServer Demo**</Name>

 <ClassName>**DemoServer**</ClassName>

 <lang>**en**</lang>

 <Address>**http://localhost:8080**</Address>

 <Port>**8080**</Port>

 <TempFolder>**TempFiles**</TempFolder>

 <DefaultPage>**index.dyalog**</DefaultPage>

 <SessionHandler>**SimpleSessions**</SessionHandler>

 <Authentication>**SimpleAuth**</Authentication>

 <UseContentEncoding>**1**</UseContentEncoding> <!-- for HTTP compression -->

 <SupportedEncodings>**deflate**</SupportedEncodings> <!-- name(s) of class(es) that use ContentEncoder interface -->

 <LogMessageLevel>**¯1**</LogMessageLevel> <!-- 1-error/important, 2-warning, 4-informational -->

 <TrapErrors>**0**</TrapErrors><!-- Valid: 0=Trap and Log errors, 1=Crash -->

 <Debug>2</**Debug**><!-- Valid: 0=No Debug Info, 1=Debug Info, 2=Allow Editing -->

 <MailMethod>**NONE**</MailMethod> <!-- Valid: SMTP|NET|NONE -->

 <MailRecipient></MailRecipient>

 <SMTP\_Gateway></SMTP\_Gateway>

</Server>

#### Server.xml Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Example | Explanation |
| Name | MiServer Demo | The name of the server. |
| ClassName | DemoServer | Valid: MildServer or the class name of a MiServer Skin which is in SiteRoot/Code/. |
| Lang | en | The language encoding of the majority of content on the site. This is primarily used by websites to determine dictionary and voice settings. |
| Address | [http://localhost:8080](http://localhost:8080/) | Not currently used. |
| Port | 8080 | The port on which the server will listen for incoming connections. NOTE: Port 80 is the default port number used by HTTP servers. If you don't already have a web server installed you might want to use 80 to avoid having to specify a port number when browsing the site. |
| SessionHandler | SimpleSessions | The name of the class which will handle sessions.  |
| Authentication | SimpleAuth | The name of the class which will handle authentication.  |
| UseContentEncoding | 1 | 1 or 0 - HTTP compression on or off |
| SupportedEncodings | deflate | The names of the classes that use the ContentEncoder interface, separated by commas in the order of usage preference. |
| LogMessageLevel | 1 | A parameter that can be used to determine the types of log messages passed through MiServer.Wrap. - |
| DefaultPage | index.dyalog | The name of the page to return if no page name is given by the browser. |
| TrapErrors | 0 | Valid: 0=Trap and Log errors, 1=Crash |
| Debug | 2 | Valid: 0=No Debug Info, 1=Debug Info, 2=Allow Editing |
| MailMethod | NONE | Valid: SMTP,NET,NONE |
| MailRecipient |  | Email address to send SMTP mail to |
| SMTP\_Gateway |  | Address for the SMTP server, if using SMTP to send emails. |

### 3.3 Testing MiServer

Load the MiServer workspace and enter the following into the session:

 )load C:\MildServer\mserver.dws

C:\MiServer\mserver saved Tue Sep 20 12:07:45 2011

 Start 'Demo' ⍝ Run the demo

 Start 'Demo' ⍝ Run the demo

MiServer started on port: 8080

Web server 'SRV00000000' started on port 8080

Root folder: C:\MiServer\Demo/

The MiServer Demo is configured with the default Server.xml file. As such, it is set to listen on port 8080 for HTTP requests and will error if there is another program using that port. Also, if you have a firewall installed you may need to grant Dyalog APL internet access.

When the server is booted, open your web browser of choice and enter http://localhost:8080. This directs the browser toward your own computer, targeting port 8080.

You should see the following page:



Take a few minutes and look through the sample pages. The bulleted links are MiPages contained within the Demo directory structure that represent some of what MiServer can do.

# C H A P T E R 4

Your First MiSite

### 4.1 Copying and Configuring the MiSite Template

The folder ServerRoot\MiSiteTemplate/ contains the essential files for a MiSite. Copy and paste the entire directory into any directory on your computer, renaming the root folder as you need it. Anytime we reference your new MiSite, we will assume it to be in C:\MyMiSite\ .

#### You’re ready to go

Start a MiServer on this directory:

 )Load 'C:\MyMiSite'

C:\MiServer\mserver saved Mon Sep 12 16:07:54 2011

 Start 'Demo' ⍝ Run the demo

 Start 'C:\MyMiSite'

DrA defaults applied

Web server 'SRV00000000' started on port 8080

Root folder: C:\MyMiSite/

### 4.2 MiPages

A MiPage is a .dyalog script file that contains an APL class derived from the ‘MiPage’ base class. Like we said before, these pages are responsible for generating HTML that is passed to the browser. They are APL code, and as such have the full power of APL at their disposal.

#### Requirements of a MiPage

These are two requirements for a MiPage:

* + - The class must be specified as derived from the base class ‘MiPage.’
		- A Render method must be included in the class. This is a monadic method that is passed the request reference. The method must pass a character string to req.Return. In the default configuration, this will then be passed to the MiServer Skin’s Wrap method for additional formatting, and will then be sent to the browser.

#### The Index Page

When the root directory of a web site is requested, web servers respond with a default resource. MiServer will serve the file listed in the DefaultPage element of Server.xml. If that resource is not present, it will return a 404 error. If you would like to change the name default page file, simply change it in the configuration file.

### 4.3 Building Your First MiPage

There are two MiPages included with the MiSite Template, the default page, index.dyalog, and a prototype dynamic MiPage, called mptemplate.dyalog. They have been included as templates for basic pages.

#### Building a basic page

When requested, this first MiPage places everyone’s favourite message, ‘Hello World!’, within the <body> tags of the HTML being sent to the browser.

In the example below, we will build this MiPage and save it as a .dyalog file in the site root directory.

###### Creating helloworld.dyalog

)ed ○helloworld

:Class helloworld : MildPage

∇ Render req;HTML

 :Access Public

 HTML←'Hello World!'

 req.Return HTML

∇

:EndClass

]Save helloworld C:\MyMiSite\

Making sure the MyMiSite server is running, open a web browser and type <http://localhost:8080/helloworld> into your address bar. You will see this in the body of your web page:



###### Page Source: <http://localhost.8080/helloworld>

<HTML><head>

<title>Demo Server</title>

<link href="[/Styles/style.css](http://localhost:8080/Styles/style.css)" rel="stylesheet" type="text/css">

</head>

<body>**Hello World!**</body></HTML>

Notice how the character string is placed between the body tags of the HTML.

#### Where is the extra HTML coming from?

While it is possible to have each page generate all the code necessary, many different content management systems allow you set a theme, or a basic look and feel, across your entire site. MiServer Skins can override the server base class method Wrap to format HTML on its way to the browser after a MiPage has sent it to req.Return.

###### DemoServer.Wrap

 ∇ Wrap req;head;HTML

 ⍝ Wrap HTML body

 :Access Public Override

 HTML←'<head>',NL

 HTML,←req.Response.HTMLHead,NL ⍝ Overwrite with anything the page code set

 HTML,←'<title>Demo Server</title>',NL

 :If ~∨/'rel="stylesheet"'⍷req.Response.HTMLHead ⍝ Add stylesheet if there is none

 HTML,←'<link href="/Styles/style.css" rel="stylesheet" type="text/css">',NL

 :EndIf

 HTML,←'</head>',NL

 HTML,←'<body>',req.Response.HTML,'</body>'

 HTML←'<HTML>',HTML,'</HTML>'

 req.Response.HTML←HTML

 ∇

### 4.4 Getting Tricky

MiPages can do much more than simply stick character vectors between the <body> tags of an HTML document. On top of being APL code, which can consume and manipulate data in ways only limited by your imagination, MiPages can communicate with the browser on the client end.

We will get into the nuts and bolts of that interaction in a few chapters, right now we are going to get the quick and dirty answers for how to build MiPages that talk with browsers.

Note: You should familiarize yourself with the HTMLInput namespace. The MiPage below uses HTMLInput functions for communicating between the server and the browser.. Appendix I has a functional reference of the namespace

#### Reverse.dyalog

Since you are running MyMiSite, copy C:\MiServer\Demo/reverse.dyalog into your site root directory. Because MiServer dynamically loads the resource on each request, there is no need to restart the server to make the resource available. The file contains the Reverse class, which is derived from the MildPage base class. The class contains two methods, Render and DoAction. Render is a required public class that generates the page HTML and passes it to req.Return. DoAction handles the submission of the page form, which will be explained in a moment. The class has been copied in for your convenience below:

Reverse.dyalog

:Class Reverse : MildPage

:Include #.HTMLInput ⍝ Useful functions for creating HTML pages

:Field Public Text←'' ⍝ Name of edit field

:Field Public Action←'' ⍝ All action buttons have this name

∇ Render req;html

 :Access Public

 DoAction ⍝ If a button was pressed, deal with it

 html←'<br>Enter Text: '

 html,←'Text'Edit Text ⍝ An "Edit" called "Text" containing the Text

 html,←'<br><br>'

 html,←'Action'Submit'Reverse' ⍝ A button named 'Action' with Caption 'Reverse'

 html,←'Action'Submit'Clear' ⍝ ... another button named 'Action’

 html←req('post'Form)HTML ⍝ Put a 'submit' form around it

 html,←'a href="/"'Enclose'Home' ⍝ A link back to the index page

 req.Return HTML

 ∇

 ∇ DoAction

 :Select Action

 :Case 'Clear' ⋄ Text←''

 :Case 'Reverse' ⋄ Text←⌽Text

 :EndSelect

 ∇

:EndClass

When this resource is requested, MiServer sends the following HTML to the browser:

Page Source: http://localhost:8080/reverse?Name=Beethoven

<HTML>

<head>

<title>Demo Server</title>

<link href="[/Styles/style.css](http://localhost:8080/Styles/style.css)" rel="stylesheet" type="text/css">

</head>

<body>

<form action="/reverse.dyalog" method="post" enctype="multipart/form-data">

<br>Enter Text:

*<input type=text size=10 id="Text" name="Text" value="Beethoven">*

<br><br>

*<input type="submit" name="Action" value="Reverse" >*

*<input type="submit" name="Action" value="Clear" >*

</form>

<a href="[/](http://localhost:8080/)">Home</a>

</body>

</HTML>

When rendered by a browser, the page looks like this:



Click on the reverse button, and it looks like this:



#### Getting Information from the Browser

If you enter text into the text field and click the ‘Reverse’ button, the value changes as we might expect it to. The text reverses. If you pay careful attention, you will notice that the page reloaded after you clicked the button. As you may have figured out from looking at the class, the value of the text type input element is passed to the Text field. Then, the Render method is called and those values change the way the HTML is constructed.

Remember that HTMLInput contains functions that generate HTML, a number of which are associated with transferring data between the browser and the server. For all those that generate input elements, the name associated with the public field is the left argument, and the value

'Text' Edit Text

#### Forms: Quick and Dirty

A little more work is in order to get back your data back to the server. You will need to wrap your input elements in a form, which is described in detail in Chapter 6. Right now, know that input elements on a page must be contained by properly formatted <form> tags to be eligible to have their values sent to the server.

HTMLInput.Form wraps HTML with those tags. The easiest method of dealing this need is simply to pass the variable that contains your HTML through HTMLInput.Form right before you pass it into req.Return. Use this syntax:

Html←req('post'Form)CharacterVectorOfYourHTML

Finally, you must trigger a new request that contains the form data, called a form submission. The aptly named ‘submit button’ is a basic way to do that. There are two different submit buttons on the example page, each generated with HTMLInput.Submit, which takes the name attribute as a left argument and the value attribute as the right argument. The value that is passed to the server during submission is the text on the button. Because only the value of the pressed submit button is sent to the server, it is a common practice to give each submit button the same name and use a :Select test to determine which was pressed.

# C H A P T E R 5

Getting Stylish

### 5.1 Styles

Up until now, we have been focused on your MiSite’s functionality, but a website’s presentation can affect its usefulness. You will notice that the HTML we have been writing says very little about the visual representation to be displayed the browser. The <br/> tag creates a line break, but how much space will that be? Moreover, why is the text displayed between the <body> tags a readable size, and not too small or large to be useful? Browsers use properties called ‘styles’ to define how to display HTML elements. Styles control the look of the content of every HTML element.

There are dozens of types of styles, including ones that affect:

* the look of text
* a page’s background
* the thickness of lines in tables
* the way a text is processed by accessibility programs that read web pages to visually impaired users.

Styling your own document allows you to make content appear to a user the way you want it to be displayed. Browsers have a set of default styles to render HTML content sensibly which allows documents that don’t define their own styles to be displayed in a generally readable fashion. However, making sure your website, your product, comes across in a recognizable and consistant way may be important to you.

#### Style Basics

There are three main ways to exercise additional control over the look and feel of your website:

* Using the style attribute within a tag
* Inserting a “cascading style sheet” (CSS) within the <head> tags of an HTML page
* Linking an external CSS document to an HTML page

A style has two parts, a selector, which is a specification of the element you want to style, and one or more declarations, which are property-value pairs. Declarations use a colon to separate the properties and values and always end with a semi-colon.

A few examples of declarations:

* color: green;
* text-align: right;
* text-decoration: underline;

### 5.2 In-Line Styles

While not generally recommended, you can insert styles directly into a tag, via the ‘style’ attribute. This bypasses the need for a selector, as it will affect that tag, and its contents, alone.

Examples:

 'style' 'color:red;' #.HTML.div 'This text will be red'

<div style="color:red;">This text will be red</div>

 'style' 'background-color:blue;' #.HTML.body 'The background within these body tags will be blue'

<body style="background-color:blue;">The background within these body tags will be blue</body>

The use of the in-line styling is frowned upon because it spreads styling over the page and does not have browser compatibility features.

### 5.3 Cascading Style Sheets

Cascading Style Sheets (CSS) are collections of styles that affect an entire page. Each declaration consists of a selector, followed by declarations contained by curly braces.

selector { property: value ; property: value; }

or

selector {

 property: value ;

 property: value;

}

#### Selectors

CSS selectors reference element names or attributes, associating declarations with them. Styles can be applied to a specific element or a group of elements. A selector might reference every <h1> tag, every <h1> tag with the class attribute “underlined,” or the specific <h1> tag with the id attribute “headline.” Each different type of selector requires a particular syntax.

A few examples of selector syntaxes include:

* h1 {} – a tag selector selects every h1 tag.
* .underlined {} – a class selector selects any element with the attribute class=“underlined”
* #headlined {} – an id selector selects the page element with the attribute id= “headline”

Each of these selectors can be combined with other selectors to pinpoint more specific elements.

For example:

* h1.note {}- selects all <h1> tags that have the class “note.”

#### Associating CSS with a web page

There are two ways to associate a style sheet with your web page.

* Internal Style Sheets – Wrapping the entire style sheet within <style type=“text/css”></style> tags contained between a page’s <head></head> tags.
* External Style Sheets – Associating a style sheet contained in a separate file with your page by including a <link/> tag within the <head> tags.

Adding internal style sheets to your page is more powerful and often more convenient than in-line styling. The method collects all the styles in one place and allows for multiple elements to be affected by the same style.

However, changing internal style sheets still means changing the code on the page level. This can get difficult to deal with as your website grows in size.

Styles Formatted for the <head> tag

 <style type="text/css">

 body { color: blue; }

 #Content { opacity:0.7; }

.center { text-align: center; }

 </style>

Example: An HTML page with CSS in the <head> tags, along with its browser representation.

<HTML>

<head>

<style type="text/css">

body { background-color:lightgray; }

.big { font-size:200%; }

</style>

</head>

<body>

Styling documents is <span class="big">easy</span> and <span class="big">fun</span>.

</body>

</HTML>



#### External CSS Sheets

The industry standard is to keep the CSS in an external document linked to your page. External CSS files allow you to tweak CSS between page loads and even swap whole CSS files for others, which is great for accessibility concerns. Moreover, you can have a standard set of styles that affects your entire site, creating a consistent look and feel.

Simply gather your styles onto a text file with a .css extension. Then, associate that style sheet with your webpage using the <link> tag within the <head> tags. In the MiSite Demo, DemoServer.Wrap inserts the link below into the <head> tag structure of each MiPage.

<link href="/Styles/style.css" rel="stylesheet" type="text/css">

### 5.4 Inserting Styles

By default, the DemoServer template associates each MiPage with SiteRoot/Styles/style.css. This contains some basic formatting and browser compatibility styles. Feel free to edit that file to familiarize yourself with composing CSS and to develop your template.

 If you have a different style sheet you would like to associate with the site, you can do one of three things.

* Replace style.css with another file of the same name.
* Change the base style sheet path in DemoServer.Wrap.
* Pass the file path of a new style sheet to the Style method of the request object (req.Style). This appends the link for a new style sheet with that path after the style.css link.

It is possible to associate multiple style sheets with a particular page. Note that if there are two different styles affecting the same property on the same element, the style that is furthest down the page will be the value rendered on page load.

# C H A P T E R 6

Under the Covers

### 6.1 Going Deeper

By this point in the manual, you should already know how to construct basic MiSites and MiPages. You can generate basic text pages or pages which communicate with the server. However, there are a number of behind the scenes factors that you should understand, specifically regarding server-browser communication. Also, this chapter will look at another MiServer resource, HTTPRequest, which is responsible for gathering the HTTP request, parsing it into useful chunks and generating the response.

### 6.2 Browser to Server Communication

#### GET and POST HTTP Requests

In general, browsers communicate with web servers by sending HTTP requests for resources. Data must be sent the same way, along with a request. To allow for different types of communication, there are a few different HTTP request structures. Currently, MiServer only supports GET and POST requests.

* GET requests are the most common form of HTTP request. These are generated when an URL is entered into an address bar or a link is clicked. It is possible to encode data along with the URL as name-value pairs. As the data is visible in the URL bar, it is generally considered a less secure.
* POST requests are generally used by “forms, and contain name value-pairs in the body of the request. The data transfer happens in the background and is considered more secure.

#### Sending Data Encoded HTTP Requests

###### Encoded URL Requests

Percent encoded URLs, commonly called encoded URLs, contain data appended to the end of the resource path. A question mark is placed after the path separating is from a number of data pieces, often name-value pairs. Characters that have significance to the browser in the address field (“/” or “:” for example), are represented with a percent and a two digit hexadecimal reference to the UTF-8 character set. A space character, for example, is represented as %20. Each piece of data is separated by an “&”.

A Data Encoded URL:

<http://localhost:8080?Name=Beethoven>

An HTMLInput.Enclose function generating that page link:

 'a href="http://localhost:8080/reverse?Name=Beethoven"' HTMLInput.Enclose 'Reverse Page With Beethoven in the Edit Field'

<a href="http://localhost:8080/reverse?Name=Beethoven">Reverse Page With Beethoven in the Edit Field</a>

#### Forms

A form is another means to send data back to the server. It is a section of an HTML document surrounded by <form> tags, containing input elements and a control to initiate the submission of an HTTP request containing the form data.

The form tags specify the resource requested upon submission (action="/reverse.dyalog"), the type of HTTP request (method="post") and the content encoding type, which needs to be specifiend but you fortunately do not need to understand (enctype="multipart/form-data").

Form from Reverse.dyalog page source

<form action="/reverse.dyalog" method="post"

 enctype="multipart/form-data">

<br>Enter Text:

<input type=text size=10 id="Name" name="Name" value="Beethoven">

<br><br>

<input type="submit" name="Action" value="Reverse" >

<input type="submit" name="Action" value="Clear" >

</form>

###### Input objects

HTML has a number of tags that are recognized as data when within a form during submission. Each of these tags has the attributes name and value, which populate the data portion of the submission. Without input objects, there isn’t any data to send back. Some of these elements are used to allow for users to submit data to the server. Others are used to store data between submissions. These include, but are not limited to, the <textarea> tag, the <select> tag and all variations of the <input> tag.

While describing the complete functionality of these elements is outside the scope of this manual, we will make use of the text and submit types of the <input> tag.

In the case of the <input> tag, the type attribute allows developers to select from a number of different formats of input controls, like text boxes, check boxes or buttons. When type="text" the <input> tag renders as an editable text box. When type="submit", the tag renders as a button.

A text box with the text “Beethoven”

 'Name' HTMLInput.Edit 'Beethoven'

<input type=text size=10 id="Name" name="Name" value="Beethoven" >

 **

A submit button with the text “Reverse”

 'Action' HTMLInput.Submit 'Reverse'

<input type="submit" name="Action" value="Reverse">

 **

#### Submitting a Form

When clicked, a submit button is used to initiate a HTTP request described by the form tags. The browser then gathers the data of all elements that are ‘valid’ for submission. A valid submission has a control name, which is usually its name attribute, is paired with the current value of the element. These are gathered by the browser and sent via the HTTP request.

### 6.3 How MiServer Gets Client Data: HTTPRequest

When MiServer receives a resource request, it creates a new instance of the class HTTPRequest. This class parses the information contained in the request into a number of fields. It is then passed to the MiPage and is used to format the response.

The data encoded in the request is stored in three possible locations, depending on the method of the HTTP request (remember that a reference to req is always passed to your Render function):

* **req.Data -** an N x 2array of the pairs sent within the body of a post request.
* **reg.Arguments** **-** an N x 2array of the pairs encoded in the URL of a get request.
* **Similarly named public fields -** If a MiPage contains a public field with the same name as the first element of a name-value pair in req.Data or req.Arguments, that field will be set to the value of that pair. For example, reverse.dyalog produces an <input> tag with Text is its name attribute. When the server receives the value of that input tag, it sets reverse.dyalog’s public field Text to the element’s value.

HTTPRequest also constructs the HTTP response, a character vector of HTML is passed to req.Return at the end of the Render method, which is stored in req.Response.HTML. A functional reference of the class is available in Appendix II.

# C H A P T E R 7

Customizing MiServer Behavior

### 7.1 Adapting MiServer to your Needs.

MiServer was originally designed as a server development environment. It is designed to make modifications easy. Storing the components of MiServer in .dyalog scripted files allows for the associated namespaces and classes to be edited within most text editors and for extensions to MiServer to be easily shared between users. Every piece of the MiServer is available for modification.

The avenues to change the behaviour of MiServer include:

* Editing the files that provide core MiServer functionality (MiServer.dyalog, HTTPRequest.dyalog, Boot.dyalog and MiPage.dyalog)
* Creating a site specific MiServer (index.dyalog)
* Associating additional .dyalog Extensions
* Adding third party plugins (JavaScript)

Editing the core files is self explanatory and not recommended right off the bat. You could make changes to the core components of MiServer, it more likely that you will want to use the MiServer core capacities with some site specific behaviour modifications. It would be a daunting proposition if you had to change the MiServer class itself every time you needed a behaviour change, and it would make the hosting of multiple MiSites with unique specifications difficult. A good deal of MiServer customization can be achieved without editing those files, so we are going to skip ahead.

### 7.2 MiServer Skins: Overriding MildServer Behaviour

The MildServer base class, which represents the core functionality of MiServer, has a number of overridable methods. By creating a class derived from this class, you can override one or all of these methods, which modify the behaviour of your server without touching the core code. Because this class changes the look and feel of the MiServer, we call it the MiServer Skin.

The following sections are descriptions of the overridable methods, including the behaviours in the MiServer skin, ‘DemoServer’.

#### Session handling

If session handling is enabled, as described in Chapter 7.3, onSessionStart should perform any processing necessary when a new session is created. Similarly, onSessionEnd should perform any processing when a session ends. The session handler packaged with MiServer, SimpleSessions, calls these functions.

DemoServer.onSessionStart and DemoServer.onSessionEnd produce log entries.

#### HTML Wrapping

Wrap takes the HTTP request object after it has passed through the MildPage and performs final processing on the response. Wrap can be used to implement a consistent look and feel across all pages of your MiSite.

While it is possible to set up a MiSite so that each page generates all HTML necessary to be rendered by a browser, it may be advantageous to use a template to give each page a similar look and feel.

DemoServer.Wrap takes the page HTML which was passed to the request object at the end of the MiPage’s render method and wraps it with the body tags of an HTML template, as well as creates a common, but overridable <head> tag structure that associates all its MiPages with the default style sheet.

#### Error Handling

Error allows for custom, server error trapping behaviours. MiServer’s error trapping methods call Error.

DemoServer.Error logs errors and posts a server-side error message to the browser.

#### Logging

Log captures and organizes log messages. Several methods in the MiServer architecture pass their status messages to Log, which defaults to posting all messages to the session window.

Log is passed both a character string message and an identifying value. There are four message levels:

1 - error/important

2 - warning

4 - informational

8 - transaction (GET/POST)

DemoServer also posts its messages to the session, but has a control that determines what levels will be displayed. Server.xml contains a parameter called LogMessageLevel, which is either set to 0 for no messages, ¯1 for all messages, or the sum of all the message levels to be displayed.

#### Cleanup

MiServer.Cleanup is called in the MiServer class destructor. Depending on the functionality of your website, there may be operations you need to perform as the server shuts down, such as untying files, disengaging from a database or even shutting down programs that were turned on to assist the server.

#### A MiServer Skin: DemoServer

Below is the example server included in SeverRoot/Demo/Code. Notice that it does not override MiServer.Error, deferring to the behaviour in the base class.

\*\*\*\*\*\*\*\*\*DemoServer.dyalog\*\*\*\*\*\*\*\*\*\*\*\*

### 7.2 Session Handling

There is no intrinsic way for a server to recognize the origin of any given HTTP request. In this ‘stateless’ environment, a server treats each request as if it came from a unique client.

Since there is no inherent marker, information must be imbedded in requests and responses that allow for web server extensions to recognize patterns in requests and identify users. These patterns, called sessions, provide a context for data to persist over multiple page loads. Sessions are ideal for sites that require users to identify themselves with a name and password or that have information that must be available during an entire visit, like a shopping cart.

#### Cookies

MiServer includes a basic session handling extension called SimpleSessions. This basic session handler uses a file called a ‘cookie’ to identify a unique session. A cookie is a text file a browser is instructed to make in the ‘Cookies’ header of the HTTP request. The file has four aspects set at this time:

* The name of the cookie
* The value of the cookie
* Which site and site paths the cookie will be sent with
* The amount of time that the file will be allowed to persist, after which it will be deleted

SimpleSessions checks to see if the HTTP request contains a cookie named ‘Session.’ If does not, it commands the browser to create one and with randomly generates an”id” that is associated with a new session. This session will be recognized until it times out (by default, after 10 minutes).

If the extension finds a ‘Session’ cookie, it compares the cookie’s value to the list of values associate with sessions which have not yet timed out. If it finds a match, session’s timeout clock is reset and the session data is copied to the req.Session field.

###### Using Cookies in Your MiPages

Cookies can also be used to maintain a user’s status after the MildServer has been shut down, which is why they are often used for tracking user behaviour. Saving information to a cookie can persist information past a page load or even a server restart.

The request object contains functions that edit the response to include instructions for the browser to create, SetCookie, or delete cookies, as well as a function that returns the Maintaining additional types of state within your MiSite

* req.SetCookie – Adds a command to the HTTP request for the browser to set the name, value and life of a cookie
* req.GetCookie – returns the value of a cookie
* req.DelCookie – tells the browser to delete a cookie by name

#### In-Session Data storage

You can also save data into the session itself by defining a variable in the req.Session.State namespace. Session.State is persisted between requests of the same session using the mechanism described above. However, once the session ends, this data will be lost.

req.Session.State.ProductNumbers← '1023' '0012' '3104'

#### Access Control and Authentication

SimpleAuth.dyalog is a simple access control extension which can restrict access to site resources based on user credentials. The configuration file Access.xml can be found in the SiteRoot\Config\ directory. Each Folder element contains a path element and a group element. The path determines the path of the directory being restricted. The Groups element is a comma separated list of all the user groups allowed access to that directory. “\*\*” is the wildcard for either element.

A Sample Access.xml

<Access>

 <Folder>

 <Path>/Admin</Path>

 <Groups>admin</Groups>

 </Folder>

 <Folder>

 <Path>/LocationOfTheFellowship</Path>

 <Groups>admin, fellowship</Groups>

 </Folder>

 <Folder>

 <Path>/Mordor</Path>

 <Groups>admin, management, ringbearer</Groups>

 <Folder>

 <Path>\*\*</Path>

 <Groups>\*\*</Groups>

 </Folder>

</Access>

When a browser requests a resource from a restricted directory, the client is prompted to enter a username and password. If the credentials match a user defined in the configuration file, SiteRoot\Config\Users.xml, those user’s credentials are added to the session. Each user is associated with one or more groups in their Groups elements. If the user is associated with one of the groups the associated with the directory, the user is given access. Otherwise, an error page alerts them that they do not have access.

A Sample Users.xml

<Users>

 <User>

 <ID>gandalf</ID>

 <Pass>youshallnotguessmypassword</Pass>

 <Groups>admin, fellowship</Groups>

 </User>

 <User>

 <ID>fbaggins</ID>

 <Pass>goonwithoutmesam</Pass>

 <Groups>ringbearer, fellowship</Groups>

 </User>

 <ID>gimli</ID>

 <Pass>shortiscute</Pass>

 <Group>fellowship</Group>

 </User>

 <User>

 <ID>Gollum</ID>

 <Pass>precious</Pass>

 <Group>ringbearer</Group>

 </User>

 <User>

 <ID>sauron</ID>

 <Pass>allseeingeye</Pass>

 <Group>management</Group>

 </User>

</Users>

# C H A P T E R 8

Using Relational Databases on a MiSite

### 8.1 Using External Data Sources

A data enabled page is one that accesses and presents data. A webpage might display today’s weather forecast or show you the history of the value of holdings in your retirement portfolio. A typical data enabled page will obtain data from some source, possibly manipulate it, format it using HTML and present it to the client.

This data can come from a variety of sources including:

* Text files
* Dyalog Component Files
* Relational Databases
* CSV files
* Excel Spreadsheets

### 8.2 Relational Databases

A data driven website needs someplace to store its data. Often, this is in a relational database, such as MySQL, Microsoft Access, IBM DB2, Microsoft SQL Server or Oracle.

#### Interacting with Relational Databases

Open DataBase Connectivity (ODBC) is a cross platform, language independent interface and is the most widely used standard through which programs interacts with databases. SQAPL, which is a standard component of Dyalog APL under Windows and is available as an option extra on other platforms, is Dyalog’s ODBC interface which provides access from Dyalog APL to any ODBC compliant database. You can incorporate SQAPL in your MiPages to interact with databases.

MiServer contains SQL, a namespace of utility functions that simplifies SQAPL integration.

#### Setting up Datasources.xml

SQL requires an xml configuration file called Datasource.xml, contained in the Siteroot/Config/ directory. The file contains 0 or more datasource elements which are information used by SQL to identify and connect to ODBC compliant databases.

Each datasource element can be defined by five possible elements:

* Name – The name used within MiServer to refer the datasource.
* DriverOptions – SQL driver options
* DSN – Database Source Name
* User – User name for authentication in the database
* Pwd - Password for authentication in the database, although we do not recommend keeping your password in a text file on your computer

In order to use SQL, you need to define one or more datasource. The Name element is required, along with a way to locate the database. The location information can kept either as a Database Source Name (DSN) as defined in your computer’s datasource administrator in the DSN element, or you can specify how connect to the database if it is a DSN-less connection in the DriverOptions Element. If these concepts are unfamiliar to you, please read the SQAPL manual, which you can find at http://docs.dyalog.com. The datasources.xml file included with the Demo server can be found below, including two properly defined datasources:

Datasources.xml

<Datasources>

 <Datasource>

 <Name>ZipCodes</Name>

 <DriverOptions>DRIVER={Microsoft Access Driver (\*.mdb, \*.accdb)}; DBQ=c:\MiServer\dyalog2011\data\zipcodes.accdb;ExtendedAnsiSQL=1;MaxBufferSize=2048;</DriverOptions>

 </Datasource>

 <Datasource>

 <Name>SQRTest</Name>

 <DSN>SQRTest</DSN>

 </Datasource>

</Datasources>

When Boot initializes MiServer, looks for data sources defined in Datasources.xml. If there are, and SQA is not copied into MiServer, it copies in and initializes SQAPL.

#### SQL.ConnectTo and SQL.Do

Once a datasource reference has been established, the SQL namespace makes use of them to bring data to your page.

If you plan to use SQAPL directly and use Datasources.xml, SQL.ConnectTo will initiate the database connection, taking the Name element of a datasource element as its right argument. After the connection has been established, you’re free to use SQAPL as you please.

SQL.Do is a cover function for SQAPL.Do, which executes SQAPL queries. SQL.Do connects to a database named in the configuration file, performs a query and then closes the connection. When Do queries for data, it returns a namespace, which contains the variables Columns, Data and Return Code. Columns are the column names of the query’s result. Data is the data returned as a result of executing the query. The variable ReturnCode will be 0 if the query was successful, all other results being error numbers described in the SQAPL manual.

Refer to the SQL namespace functional reference in Appendix I for syntax.

A sample page using these commands is included below:

:Class SQLdemo : MildPage

 :Include #.HTMLInput

 :Include #.SQL

 :field Public state←''

 :field States

 ∇ Render req;HTML;form;data;chunk

 :Access Public

 HTML←'h2'Enclose'SQAPL/JQuery Demonstration'

 :If 0∊⍴States

 HTML,←BRA'h3'Enclose'ZipCodes database is not available! Sorry...'

 :Else

 chunk←BRA'h3'Enclose'Zip Codes by State'

 form←'Select State: ','state'DropDown States state'autofocus="autofocus" onChange="this.form.submit()"'

 chunk,←'action="#"'('post'Form)form

 :If state≢''

 data←Do'ZipCodes' 'select \* from ZipCodes where StateAbbr = :a<C2: order by Zipcode' state

 :If 0=data.ReturnCode

 chunk,←req #.JQ.TableSorter'tab1'(data.Columns⍪data.Data)'' 1

 :Else

 chunk←'h3'Enclose'Database query failed? RC = ',⍕data.ReturnCode

 :EndIf

 :EndIf

 HTML,←chunk

 :EndIf

 req.Return HTML

 ∇

 ∇ Init;data

 :Implements constructor :base

 :Access public

 States←''

 data←SuperDo'ZipCodes' 'select \* from States order by StateName'

 :If data.ReturnCode=0

 States←'' ''⍪data.Data

 :EndIf

 ∇

:EndClass

### 8.2 Displaying Data in your MiPage

Using data received from the browser is simple, as long as you have given a public field the same name as an input elements that are ‘valid’ for submission. These character vectors can simply be added to your HTML vector:

HTML←’Here’s the data from the “text” field: ’,text

Data in accessed from other APL processes may not produce such clean results.There are a few utilities packaged with MiServer that take APL data and turn it into HTML.

#### Displaying Table Data Using HTMLInput.Table

Let’s face it. A lot of data is rectangular. HTMLInput.Table is used to display a simple matrix of data by enclosing the data within an HTML table tag structure. Because this is done at each page load, it is useful for displaying dynamic data. The first element of the right argument passed to HTMLInput.Table is a matrix of no greater than depth 2.

:Class tablesorter : MildPage

 :Include #.HTMLInput

 ∇ Render req;tabledata;html

 :Access Public

 tabledata←4 2⍴'Names' 'Ages' 'Frodo' 33 'Gollum' 589 'Gandalf' '~2000'

 html←Table tabledata '' '' '' 1

 req.Return html

 ∇

:EndClass

#### JQ.Tablesorter

If you are dealing with large chunks of data, you may benefit from sorting and pagination. The JQ namespace contains page elements backed up by client side scripting, as described in the next chapter. JQ.TableSorter is described in detail in the next chapter and in Appendix I.

# C H A P T E R 9

Improving your UI with JQuery

### 9.1 JQuery and JQueryUI

JavaScript is a popular client side scripting language that is compatible with most browsers. It can create flashy user experiences with widgets, effects and animations and it can support server side business logic by processing or validate data before it is sent back in a request. JQuery is a comprehensive, open source JavaScript library that provides extensive control over the document object model (DOM). The DOM is the tree-like representation of the HTML elements in a web page.

JQuery allows you to search within and manipulate the content of the document object model. For more information about leaning JQuery, visit <http://www.jquery.com>.

JQueryUI is a library which contains a number of plugins based on JQuery. It is a collection of widgets and other tools used to affect the user interface of a webpage. JQueryUI makes it easy to develop sophisticated users interaction. It is a really powerful library and frankly it’s really cool. If you have a chance, check out <http://www.jqueryui.com>.

### 9.2 The JQ namespace

We have provided you with utility functions in the JQ namespace to make it easy to integrate several JQueryUI widgets into your MiPages. These include:

#### JQ.Accordion

This widget has multiple pages that you can click through.

content←2⍴⊂''

content[1]←⊂'Here is the First Page'

content[2]←⊂'Here is the Second Page'

headers←'The First Page' 'The Second Page'

html←(#.HTML.h2'Accordion'),'style' 'width:200px; height:200px;'#.HTML.div req #.JQ.Accordion'myaccordion' headers content 'fillSpace: true'

####

#### JQ.TableSorter

This widget is a table that is sortable by column with optional pagination.

data←4 2⍴'Names' 'Ages' 'Frodo' 33 'Gollum' 589 'Gandalf' '2000'

html←req #.JQ.TableSorter'mytable'(data'' '' '' 1)''



#### JQ.DatePicker

This widget is a text input box that when selected pops up a calendar.

html←(#.HTML.h2'DatePicker')

html,←BRA'Using Multiple Months and yyyy-mm-dd format'

html,←2 BRA'Date: ',req #.JQ.DatePicker'mydate1'('' 20)'numberOfMonths: 3,showButtonPanel: true,dateFormat: "yy-mm-dd"'

html,←BRA'Using Month and Year Menus and DD, dd MM yyyy format'

html,←'Date: ',req #.JQ.DatePicker'mydate2'('' 30)'changeMonth: true,changeYear: true,dateFormat: "DD, d MM yy"'





# A P P E N D I X I

## Utilities Functional References

MiServer includes a number of utility files, found in ServerRoot\Utils\, designed to assist you MiSite. You are encouraged to look through these files and use the utility functions provided in the generation of your own functions

Currently, there are four main utility namespaces:

* HTML – Functions that generate simple HTML tags.
* HTMLInput – Functions that make more
* JQ – Functions that make JQueryUI widgets
* SQL –Functions that make interact with SQAPL

### HTML Namespace Functional Reference

###### Background

This namespace contains functions that generate HTML.

####  Functions

##### HTML.\* Insert HTML Tags

 html←{attrs} HTML.***fn*** innerhtml

|  |  |
| --- | --- |
| attrs | The optional left argument contains any additional attributes for the HTML tags. These can be passed as either an N × 2 matrix of attribute-value pairs, a character vector or as a vector of vectors where each element contains two character vectors representing the name and value and a vector of vectors of depth 2 of alternating name values. |
| ***fn*** | The function which produces a tag of the same name.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | div | h1 | head | ul | p |
| b | font | h2 | html | li | pre |
| body | form | h3 | input | link | span |

 The currently implemented functions are: |
| attributes | Any additional attributes to be placed in the opening tag. (default is '')  |

### HTMLInput Namespace Functional Reference

###### Background

The HTMLInput namespace contains a number of more complex functions that create HTML. Originally, this namespace was designed to make working with the dynamic functionality of the <input> tag more APL-like, but has since expanded to include a number of other functions.

#### Functions

##### HTMLInput.APLToHT ML Insert <br/> Tag After HTML

 html←APLToHTML APL

|  |  |
| --- | --- |
| html | HTML to insert break tags after. |

##### HTMLInput.BRA Insert <br/> Tag After HTML

 html←{n}BRA html

|  |  |
| --- | --- |
| n | the number of tags to insert. (default is1) |
| html | HTML to insert break tags after. |

##### HTMLInput.BR Insert <br/> Tag Before HTML

 html←{n}BR html

|  |  |
| --- | --- |
| n | the number of break tags to insert (default is1) |
| html | HTML to insert break tags before. |

##### HTMLInput.Button HTML Button

r←name Button value {attributes}

|  |  |
| --- | --- |
| name | The value of the name attribute. |
| value | The value of the value attribute, displayed as text on the button. |
| attributes | Any additional attributes to be placed in the tag. (default is '') |

##### HTMLInput.Checkbox HTML Checkbox

HTMLInput.Checkbox wraps a character vector with <input> tags with the type="checkbox" attribute. This renders as a square, selectable box with text defined by the name attribute to its right.

r←name HTMLInput.Checkbox checked {attributes}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| checked | This sets the checked status of the checkbox (default 0). If 1, the checkbox is checked. |
| attributes | Any additional attributes to be placed in the opening tag. (default is '')  |

##### HTMLInput.DropDown HTML Dropdown Menu

r←name DropDown items {value} {attributes} {sort}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| items | An n element vector of the items to be selected from in the dropdown menu. ('Item1' 'Item2') |
| value | A the value to be displayed when the dropdown box is generated. (default 'Item1') |
| attributes | Any additional attributes to be placed in the opening tag. (default is '') |
| sort | 1 or 0 (the default)  If 1, and value matches one element of the items, will place the selected tag within the item and move it to the front of the list. |

##### HTMLInput.Edit HTML Text Field

r←name Edit value {size} {attributes}

|  |  |
| --- | --- |
| name | the value of the name and id attributes |
| value | the text to be displayed in the element (Default is '') |
| size | the maximum character count of the text box (default is 10⌈⍴size) |
| attributes | Any additional attributes to be placed in the opening tag. (default is '') |

##### HTMLInput.Enclose Wrap HTML with Tag

r←attribute Enclose innerHTML

|  |  |
| --- | --- |
| attribute | The tag that will be enclosing the HTML. This can include any number of additional attributes. |
| innerHTML | A character vector |

##### HTMLInput.File HTML File Upload Button

r←name File size {value} {attributes}

|  |  |
| --- | --- |
| name | The value of the name attribute and the id attribute. |
| size | The maximum character count of the file box |
| value | A character string that will be displayed in the text field. |
| attributes | Any additional attributes to be placed in the opening tag. (default is '') |

Note: This tag is not evenly supported by all browsers.

##### HTMLInput.Form Insert HTML Form

r←{atts} (method Form) innerHTML

|  |  |
| --- | --- |
| method | The type of HTTP request to be sent. A form can only use the GET or POST HTTP request types. |
| atts | Any additional attributes to be placed in the opening tag. (default is '') |
| innerHTML | A character string of the HTML to be wrapped by the tag |

Discussion:

When using the “POST” method, HTMLInput.Form adds ‘enctype=“multipart/form-data”’ to the leading form tag.

##### HTMLInput.Hidden HTML Hidden Field

r←name Hidden {value} {attributes}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| value | The value of the value attribute. |
| attributes | Any additional attributes to be placed in the opening tag. (default is '') |

##### HTMLInput.JS Insert JavaScript

r←JS script

|  |  |
| --- | --- |
| script |  A character vector of JavaScript.  |

##### HTMLInput.MultiEdit HTML Multiple Row Text Field

r←name MultiEdit {rows cols} {values} {attributes}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| rows | Rows of text field, in characters. (default is 10) |
| cols | Columns of text field, in characters. (default is 40) |
| value | Text displayed in the text field. (default is '') |
| attributes | Any additional attributes to be placed in the tag. (default is '') |

##### HTMLInput.Password HTML Password Field

r←name Password {size} {value} {attributes}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| size | The maximum character count of the password box |
| value | The text value of the password box (default is '') |
| attributes | Any additional attributes to be placed in the opening tag. (default is '') |

##### HTMLInput.RadioButton HTML Radio Button

r←name RadioButton {checked }{value} {attributes}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| checked | 0, the default, or 1If 1, the button is checked. |
| value | The value of the value attribute |
| attributes | Any additional attributes to be placed in the opening tag. (default is '') |

Note:

Only one radio button can be selected by the user in each form. However, if multiple radio buttons are created with their checked attribute set to one and the user does not make a radio button selection before form submission, multiple radio buttons will pass a checked value.

##### HTMLInput.SP Insert Spaces Before HTML

 html←{n}SP html

|  |  |
| --- | --- |
| n | the number of spaces to insert (default is1) |
| html | HTML to insert spaces before. |

##### HTMLInput.Submit HTML Submit Button

r←name Submit value {attributes}

|  |  |
| --- | --- |
| name | The value of the name attribute. |
| value | The value of the value attribute, displayed as text on the button. Determines button size. (default is 'Push Me!') |
| attributes | Any additional attributes to be placed in the tag. (default is '') |

##### HTMLInput.Table Enclose Array in HTML Table

r←{name}Table {data} {table\_atts }{cell\_attribs} {header\_attribs} {header\_rows}

|  |  |
| --- | --- |
| name | The value of the name and id attributes. |
| data | A matrix with no more than a depth of 2. (default is ‘data’) |
| table\_atts | Attributes to be placed in the leading ‘table’ tag |
| cell\_attribs | Attributes to be placed in the cells of the table outside of the header. (default is '') |
| header\_attribs | Attributes to be placed in the table’s header rows. (default is '') |
| header\_rows | Number of rows that will be marked as the table’s header. (default is ⍬) |

##### HTMLInput.TextToHTML Preserve Text Formatting

 html←TextToHTML html

|  |  |
| --- | --- |
| html | character vector of HTML |

### JQ Namespace Functional Reference

###### Background

JQ is a namespace designed to simplify JQuery integration into your MiPages. It contains cover functions that add JQueryUI widgets to your MiPages. JQ has implemented only a few of the vast menagerie of JQuery plugins and widgets, but can be considered an example of how to integrate JQuery functionality in an APL envornment.

#### Path Variables

For a JQuery widget to be properly rendered, the page it is on must contain:

1. <script> tags that associate it to

the JQuery library

the JQueryUI library

and any necessary plugins

1. <link> tags that associate the appropriate cascading style sheets. These links tell the browser where to find the additional files that need to be loaded and generate HTTP requests for those resources.

The resources listed below are all those necessary to support the cover functions in the JQ namespace. If you additionally develop the namespace, you may need to add more.

|  |  |  |
| --- | --- | --- |
| Field | Value | Description |
| JQuery | 'src="/Scripts/jquery-1.6.2.min.js"'  | JQuery JavaScript |
| JQueryUIcss | '/Styles/redmond/jquery-ui-1.8.16.custom.css'  | JQuery UI styles |
| JQueryUI  | 'src="/Scripts/jquery-ui-1.8.16.custom.min.js"'  | JQuery UI JavaScript |
| JQueryTS  | 'src="/Scripts/jquery.tablesorter.min.js" '  | TableSorter plugin javascript |
| JQueryTScss | '/Styles/blue/style.css'  | TableSorter plugin styles |
| JQueryTSPager | 'src="/Scripts/jquery.tablesorter.pager.js"' | Pager plugin JavaScript |
| JQueryDims | 'src="/Scripts/jquery.dimensions.min.js" | JQuery Dimensions plugin JavaScript |

#### Functions

##### JQ.insertClass Insert a Class Among HTML Attributes

r←class insertClass attrs

|  |  |
| --- | --- |
| class | Thename of the class to be inserted. |
| attrs | A character vector of HTML attributes (default is  |

Discussion:

JQ.insertClass inserts a class attribute into a list of attributes. If the class attribute already exists among that list, the class passed on the left will be inserted into that attribute in addition to others.

##### JQ.IncludeJQuery Insert JQuery Script Link

JQ.IncludeJQuery creates a <link> tag that references the JQuery file within the page <head> tags, if one does not already exist.

r←IncludeJQuery req

|  |  |
| --- | --- |
| req | The HTTPRequest object. |

##### JQ.IncludeJQueryUI Insert JQueryUI Script Link

JQ.IncludeJQueryUI creates a <script> tag that references the JQuery-UI file within the <head> tags, if one does not already exist.

r←IncludeJQueryUI req

|  |  |
| --- | --- |
| req | The HTTPRequest object. |

##### JQ.JQueryfn builds JQuery call

r←JQueryfn {JQueryFunctionName HTMLid JQueryFunctionPars JQueryFunctionChain}

|  |  |
| --- | --- |
| JQueryFunctionName | The name of the JQuery function. (default is '') |
| HTMLid | The id of the HTML element affected by the JQuery widget. (default is '') |
| JQueryFunctionPars |  The parameters passed to the JQuery function. (default is '') |
| JQueryFunctionChain | The code for any chained functions. (default is '') |

#### Cover Functions

The following functions generate calls for JQuery widgets. Each function makes sure the appropriate JQuery files are linked to in the <head> tags.

##### JQ.DatePicker JQuery DatePicker Widget

r←{req} DatePicker (id editpars jqpars)

|  |  |
| --- | --- |
| req | The HTTPRequest object. |
| id | The id for the Datapicker. |
| editpars | The Parameters for the text field (see HTMLInput.Edit) |
| jqpars | The Datepicker JQuery parameters. |

##### JQ.TableSorter JQuery TableSorter Widget

 r←{req}TableSorter {id tablepars jqpars pager}

|  |  |
| --- | --- |
| req | The HTTPRequest object. |
| id | The id attribute of the TableSorter (default is '') |
| tablepars | The parameters for the table (see HTMLInput.Table) (default is ⍬) |
| jqpars | The TableSorter JQuery parameters (default is '') |
| pager | 0, the default, or 1 The use of the Pager plugin, which adds pagination to the table. |

##### JQ.Accordion JQuery Accordion Widget

r←{req} Accordion (id hdrs content jqpars)

|  |  |
| --- | --- |
| req | The HTTPRequest object |
| id | The id attribute for the Accordian. |
| jqpars | The Accordion JQuery parameters. |
| hdrs | An n-element array of header names for each Accordion folder.  |
| content | An n-element array of content for each accordion folder. |

##### JQ.Tabs JQuery Tab Widget

r←{req} Tabs (id tabnames content jqpars)

|  |  |
| --- | --- |
| req | The HTTPRequest object. |
| id | The id attribute of the tabs. |
| content | An n x 2 matrix of [;1] tab titles [;2] tab contents. |
| jqpars | The Tabs JQuery parameters. |

##### JQ.Dialog JQuery Dialog Widget

r←{req} Dialog (id title body jqpars)

|  |  |
| --- | --- |
| req | The HTTPRequest object. |
| id | The id attribute of the Dialog. |
| jqpars | The Dialog JQuery parameters. |
| title | The title for the Dialog window. |
| innerHTML | The HTML displayed in the body of the Dialog window. |

#### What’s Needed to Use JQuery?

To use JQuery on a MiPage, you must include a link to the JQuery JavaScript file, which we have included in the ServerRoot\Plugins\JS\ folder. We have included the most recent version of JQuery, 1.6.2. We recommend that you periodically check for updates on the JQuery website. You should copy this file into the SiteRoot/Scripts folder of your MiSite. You will need to include a <script> tag linking to the JavaScript in the <head> tag of your MiPage.

<script type="text/javascript" src="YourJQueryLibrary.js">
</script>

To use JQueryUI, you also need a link to the JQueryUI JavaScript file and a link to a JQueryUI CSS theme. If you decide to include plugins or widgets that are not implemented by JQueryUI, you will also need links to whatever files they require.

### SQL Namespace Functional Reference

###### Background

SQL is a namespace designed to simply integrate SQAPL, Dyalog’s ODBC compliant database interaction tool, with your MiSite. The following functions require a properly formatted Datasources.xml file, as described in Chapter 8.

#### Functions

##### SQL.ConnectTo Connect to a Datasource

r←ConnectTo database

|  |  |
| --- | --- |
| database | The name the datasource, from the name element of one of the datasources described in Datasources.xml |

##### SQL.Do Connect to and Query a Datasource

r←SQUAPLDo database sqlstmt {bindvars}

|  |  |
| --- | --- |
| database | The name the datasource, from the name element of one of the datasources described in Datasources.xml. |
| sqlstmt | The SQL statement to be executed |
| bindvars | Data for bind variables, if any. |

Discussion:

SQL.Do returns a namespace containing two variables:

|  |  |
| --- | --- |
| ReturnCode | 0 if successful. An error code reference can be found in the SQAPL manual. |
| Data | The matrix of data returned |
| Columns | A vector of column names |

Note: SQL.Do, unlike SQA.Do, always fetches all of the data and has no block mode.

##### SQL.CloseAll Close All SQAPL Connections

r←SQL.CloseAll

# A P P E N D I X II

## HTTPRequest Reference

### The Request Object

An instance of the HTTPRequest class, req, is generated at each HTTP request. The request object has a number of fields that contain useful data populated from the request and a number of methods that display and manipulate that data as well as to generate the HTTP response.

### Parsing the HTTP Request

###### Each HTTP request is parsed and distributed among the following fields:

|  |  |
| --- | --- |
| Input | The request line of the HTTP request. This includes the type of request, the resource to be requested and the version of the HTTP being used to format the request. |
| Headers | All the headers of the HTTP request |
| Command | The type of request (post or get) |
| Page | The name of the requested resource |
| Arguments | Any name-value pairs passed within the URL are stored in this field as a 2 × N matrix of name-value pairs |
| PeerCert | When using secure communications, the certificate presented by the client |
| Data | When a post request is encoded with data, the data gets stored in this field as a 2 × N matrix of name-value pairs |
| Cookies | A list of the cookies being used by the server |

### Namespaces

##### HTTPRequest.Session Persists Session Data

###### Notable Content

|  |  |
| --- | --- |
| State | A namespace that persists between page loads in a session. Store session specific variables here. |

##### HTTPRequest.Server Stores Server Settings

###### Notable Content

|  |  |
| --- | --- |
| Config | A namespace of variables generated from the elements of SiteRoot/Config/server.xml. |

##### HTTPRequest.Response Stores the HTTP Response

###### Notable Content

|  |  |
| --- | --- |
| HTML | A variable that contains the all HTML, save that found in the <head> tag structure. |
| HTMLHead | A variable containing the HTML between the <head> tags. |
| Status | The HTTP status code to be returned to the browser. |
| StatusText | The HTTP status message to be returned to the browser. |

### Functions

##### HTTPRequest.Return Passes MiPage HTML to Server

r←{hdrs} Return html

|  |  |
| --- | --- |
| hdrs | A character vector to be added between the <head> tags. This is concatenated to Response.Headers |
| html | The character vector of HTML |

##### HTTPRequest.GetCookie Retrieve Cookie Value

r←SetCookie name

|  |  |
| --- | --- |
| name | The name of the cookie. |

##### HTTPRequest.SetCookie Set a Cookie

r←SetCookie {name value path keep}

|  |  |
| --- | --- |
| name | The name of the cookie. (default is 'CookieName')  |
| value | The value that will be passed with the cookie. (default is 'CookieValue')  |
| path | The path with which the cookie will be associated. (default is '/')  |

##### HTTPRequest.DelCookie Delete A Cookie

r←DelCookie ctl

|  |  |
| --- | --- |
| name | The name of the cookie. (default is 'CookieName')  |
| path | The path with which the cookie is associated. (default is '/') |

##### HTTPRequest.Title Add a Page Title

r←Title x

|  |  |
| --- | --- |
| x | The name to be displayed at the top of the browser. |

##### HTTPRequest.Script Insert Script in <head> Tags

r←{tags} Script x

|  |  |
| --- | --- |
| tags | The Attributes of the opening tag. If a type attribute is not included in the list of tags, 'type="text/javascript"' is inserted. |
| x | The character vector of the script. |

##### HTTPRequest.JSPlugIn Insert JQuery Plugin Link

r←JSPlugIn file

|  |  |
| --- | --- |
| file | The character vector file path of a JQuery plugin |

##### HTTPRequest.Style Insert CSS Link

r←Style file

|  |  |
| --- | --- |
| file | The file path of a cascading style sheet |

# A P P E N D I X IV

## The Future of MiServer

#### The MiServer Project

The MiServer Project is an open source project to promote the development of MiServer and serve as a community building exercise for the APL community. We hope that it becomes a focal point for the considerable talents we see every day among our fellow APLers.

We want people to use, talk about, modify, experiment with, and extend MiServer.

There will be two avenues for this:

* The MiServer page at APLWiki at <http://www.APLWiki.com>/MiServer will be a repository for community contributed content, as well as where we will distribute the ‘official’ release of MiServer
* We will also publish MiServer on the Dyalog Library

We are excited to see what we can build together.

#### Directions for Development

We are committed to working with the APL community to extend MiServer and will continue to develop it on our own. Currently, there are a number of things on our plate.

Making MiServer easier to deploy and administer:

* Being able to run as a service, either as a Windows Service, or a daemon under \*nix operating systems
* Remote management and configuration, such as using a second TCP/IP as a separate path for administration
* Extending the usage and error logging capabilities of MiServer

Functionality features visible to the developer:

* Continue to enhance JQuery support
	+ Add cover functions for more widgets and plug-ins
	+ Client side form validation
	+ Provide a means to return data manipulated by JQuery back to the server
* Use XMLHTTP requests to provide AJAX-like functionality
* Extend Datasources.xml to accommodate other types in addition to ODBC
* Implement Digest Access Authentication for better security than Basic Authentication

Improve Performance:

* Look at other content encoding schemes, gzip being the most likely
* Profile MiServer performance and address hot spots
* Analyze the trade-off between compression and transmission.  For small requests, the time to compress may exceed the time to transmit

1. MiServer is implemented using the object oriented feature of Dyalog APL. You do not need to know much about OO to develop web sites using MiServer, but if you’re unfamiliar with the terms “class” and “instance”, you can learn more about them in the publication “Introduction to Object Oriented Programming for APL Programmers” which is included in the documentation provided with Dyalog APL. [↑](#footnote-ref-1)
2. Please note: This is the default configuration and will be used in all subsequent examples. [↑](#footnote-ref-2)