Overview

This document describes how to configure MiServer 3.0.

MiServer's Configuration Architecture

MiServer uses a 2-tiered configuration system – one at the server level and one at the MiSite[1] level.

The server-level settings in the MiServer installation are comprehensive – every configuration setting that MiServer knows about has a setting.
The server-level settings are installed in MiServer's `/Config/` folder.

The MiSite level configuration settings override their server-level counterparts.
MiSite level settings **must** be installed in the `/Config/` folder of the MiSite.

MiServer seamlessly merges the levels using whatever server-level settings for any that are not defined at the MiSite level. A MiSite can run without any MiSite-level configuration - it just uses the server-level settings.

This gives you a few options:

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

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

Changing the server-level configuration settings is discouraged. Doing so will run the risk of the settings being overridden when you update MiServer, or at a minimum require that you merge the settings of the update with those you have changed. In other words, don't modify the files in MiServer's `/Config/` folder. ☺

[1] A MiSite is a MiServer web site

Configuration Files

There are several XML files in the `/Config/` folder which store the configuration settings.

File	Description
`ContentTypes.xml`	Used internally by MiServer to select the appropriate HTTP content-type header value when composing a response. In general, you will not need to define these.
`Logger.xml`	Settings for the HTTP Request logger extension found in the `/Extensions/` folder
`Resources.xml`	Defines the interdependencies and resources (JavaScript and CSS files) needed by widgets and utility libraries. In general, you will not need to modify this unless you plan build your own widget APIs.
`Server.xml`	Defines all server settings. These are the settings that you will most likely change to suit your needs.
`Virtual.xml`	Defines "virtual" directories (aliases) for flexible management of widget libraries.

Server.xml Settings

Settings that you are most likely to copy to and modify at the MiSite level are **green**.
Any settings that are for time-related items (e.g. timeout settings) may be specified with a number followed by a unit (d – days, h – hours, m – minutes, s – seconds, ms – miliseconds).

Setting	Description
General Settings	
`Name`	The name of the current MiSite Default: MiServer 3.0
`Host`	The domain or IP address for your server. This can be useful if you want to include the address of your MiSite in the text of your MiPage Default: localhost
`Server`	This name that is sent in the `server` HTTP response field when MiServer sends a response to the client. Default: MiServer/3.0

ClassName	The name of the class which implements the server component within MiServer. For a simple MiSite, you're unlikely to change this. But for a MiSite that integrates with some business logic or needs some initialization, you will likely need to write a class derived from the MiServer class (see the Customizing MiServer document). Default: MiServer
LogMessageLevel	Controls what types of diagnostic messages are logged within MiServer. This is sepearate from HTTP request logging, which is controlled by the Logger extension. Message types are 1 – error/important 2 – warning 4 – informational 8 – transactional Message types can be additive – 5 would log error and informational messages. ¯1 – will log all message types The messages are outout via the **Log** method in the MiServer class and by default displayed in the APL session. If you implement a class derived from the MiServer class, this method can be overridden and output directed as you desire. Default: 1 (error/important)
RESTful	Boolean which indicates if the server is running a RESTful web service. Default: 0
Production	Boolean which indicates if the server if running in production mode. When Production is 0, MiServer's debugging framework is active and will halt when an error occurs. Default: 0
IdleTimeout	The MiServer class has an overridable method, **onIdle**, which is called during "idle" server periods (the time since the last request was received by the server). **onIdle** can be useful for performing cleanup tasks during periods of low server activity. IdleTimeout specifies how the minimum time after the last request was received that must pass before **onIdle** is called. A value of 0 indicates not to call **onIdle**. Default: 0

Communications Settings	
Ports	The list of ports that MiServercan listen on. MiServer will listen on the first available port in the list. The list can be a list of integers, or a range. This setting is useful if you run mulitple MiServers, particularly during development. If Ports is empty, MiServer will only attempt to list on the port specified by the Port setting. Default: 8080-8090
Port	The preferred port to use. MiServer will first attempt to listen on this port and failing that will attempt to listen on the remaining ports from the Ports setting. Default: 8080
IPVersion	The IP version that Conga should use. Valid values are IPv4, IPv6, and IP (meaning let Conga decide). Default: IPv4
UseContentEncodings	Boolean indicating whether to use HTTP compression when sending the response from the server to the client. Default: 1
SupportedEncodings	A comma-delimited list class names which implement content encodings. A content encoder is a class built using the ContentEncoder interface. Default: gzip,deflate
Secure	Boolean indicating to use HTTPS. Default: 0
CertFile[2]	The path to the server's public certificate file for secure communications. Default:
KeyFile[2]	The path to the server's private key file for secure communications. Default:
RootCertDir[2]	The path to folder containing the CA root certificates for secure communications. Default:
SSLFlags[2]	The TLS/SSL flags for secure communications. Default: 96 = 32 Accept without validating + 64 Request Client Cert
WaitTimeout	The time that Conga will wait before timing out. Default: 5000ms

[2] Information about these settings can be found in the Conga documentation.

Page/Request Settings	
Lang	The lang attribute to use for the HTML pages rendered by MiServer. This may be overridden for individual pages by setting the 'lang' attribute for the page. Default: en
DefaultPage	The name of the MiPage to attempt to load when no page is specified in the request URL. Default: index.mipage
DefaultExtension	The default file extension to use when no extension is specified. Default: mipage[3]
AllowedHTTPCommands	A comma-delimited list of the HTTP commands that MiServer will serve. This is primarily used for MiServers which implement RESTful web services. Default: get,post
HTTPCacheTime	The length of time to cache static content. Default: 60m
Extensions[4]	
SessionHandler	The name of the class which implements session handling. Default: SimpleSessions
SessionTimeout	The length of inactivity before a user session times out. Default: 30m
Authentication	The name of the class which implements HTTP authentication. Default: SimpleAuth
Logger	The name of the class which implements HTTP request logging. Default: Logger

[3] MiServer 3.0 changed from .dyalog to .mipage. The rationale for this was to distinguish APL script files which defined MiPages from other APL script files.
[4] Extensions allow the user to implement their own behavior for various aspects of MiServer.

Error Handling (DrA[5]) Settings	
TrapErrors	Boolean indicating whether (1) to trap and log errors or (0) to crash and stop in the APL session. Default: 0
Debug	Indicates what debugging information to display. 0 – no debugging information 1 – display an HTML page at the client with debugging information 2 – allow the user to edit the page's source code Default: 2
MailMethod	Method to use if sending email reports of errors. Valid values are NONE, SMTP, NET (Windows .NET only). Default: NONE
MailRecipient	The email address to which to send error reports. Default:
SMTP_Gateway	If MailMethod is set to SMTP, this is the gateway address. Default:

Logger.xml Settings

Logger.xml specifies settings for the Logger extension that is included with MiServer.

It has two settings

`<active>` - Boolean indicating whether to log HTTP requests (1) or not (0)

`<directory>` - path to the folder where log files are to be stored

You may use the following replacements in `<directory>`:

```
%ServerRoot% - MiServer root directory
%SiteRoot%   - MiSite site root directory
```

Example:

```
<Logger>
<!-- valid replacements are
     %ServerRoot% - MiServer root directory
     %SiteRoot%   - web site root -->
  <active>0</active>              <!-- 1 for yes, 0 for no -->
  <directory>%SiteRoot%/Logs</directory>
</Logger>
```

[5] DrA is a Dyalog utility to trap and log errors

Virtual.xml

Virtual.xml allows you to specify aliases for folders in MiServer and your MiSite. This is done by associating a name with folder. The aliases defined are then used to Resources.xml for widget resource defintion.

You may use the following replacements:

```
%ServerRoot% - MiServer root directory
%SiteRoot%   - MiSite site root directory
```

Each entry in Virtual.xml has a format similar to these:

```
<directory>
  <alias>Dyalog</alias>
  <path>%ServerRoot%/PlugIns/Dyalog/</path>
</directory>

<directory>
  <alias>JQuery</alias>
  <path>%ServerRoot%/PlugIns/JQuery/</path>
</directory>
```

One advantage of using aliases is that you can easily upgrade widget libraries (and revert if needed!). For instance, to upgrade to new version of Syncfusion, I change

```
<alias>Syncfusion</alias>
<path>%ServerRoot%/PlugIns/Syncfusion-14.2.0.26/</path>
```

to

```
<alias>Syncfusion</alias>
<path>%ServerRoot%/PlugIns/Syncfusion-14.3.0.49/</path>
```

And all of the widgets that use Syncfusion now point to the new version.

Resources.xml

Resources.xml allows you to specify the JavaScript and CSS files that are necessary to be loaded for a widget to function. Each resource has a name one or more elements that specify the scripts, style, and other resources that are necessary for this resource. These elements are:

`<script>` - specifies a JavaScript file
`<style>` - specifies a CSS file.
`<uses>` - specified another resource used by this resource

All of the widget APIs provided with MiServer have predefined resources and those resources are automatically included when you add the widget to your MiPage. In general, you would only need to add resource definitions to your MiSite Resources.xml file for additional widgets you choose to use in your MiSite.

Examples:
Notice that the following resources makes use of the JQuery alias specified in Virtual.xml.

This resource specifies the JavaScript files necessary for JQuery.

```
<resource>
  <name>JQuery</name>
  <script>/JQuery/jquery-1.12.3.min.js</script>
  <script>/JQuery/APL_JavaScript_Utils.js</script>
</resource>
```

This resource specifies the CSS file necessary for the jQueryUI theme.

```
<resource>
  <name>jqTheme</name>
  <style>/JQuery/JQueryUI/Themes/redmond/jquery-ui.min.css</style>
</resource>
```

Note that resources can speficThe JqueryUI library

```
<resource>
  <name>JQueryUI</name>
  <uses>JQuery</uses>
  <uses>jqTheme</uses>
  <script>/JQuery/JQueryUI/jquery-ui.min.js</script>
</resource>
```