# Contents

Genero Application Server User Guide................................................................. 7

What's new in Genero Application Server (GAS), v 3.00 (Maintenance Releases) .......................................................... 9

Genero Application Server overview................................................................. 11
  What is the Genero Application Server?............................................................... 11
  Standalone Genero Application Server ............................................................... 14
  Front-ends and Extensions................................................................................. 15

GAS Quick Start Guide....................................................................................... 16
  Explore Genero application server resources.................................................... 16
  Quick start guide for applications with UI......................................................... 17
    Launch first application...................................................................................... 17
    Configure an application.................................................................................... 19
    Run an application............................................................................................. 20
    Deploy an application......................................................................................... 22
  Quick start guide for web services applications................................................ 23
    What is Web Service?.......................................................................................... 23
    Explore Genero Web service (server side)......................................................... 23
    Configure a Web service..................................................................................... 24
    Run a client application on Web service using GWC....................................... 25
    Deploy a Web service......................................................................................... 25
  Genero demo applications................................................................................ 27
    Find the demo applications............................................................................... 27
    Display demo applications with the Genero Web Client................................... 27

GAS Basics........................................................................................................ 29
  Architecture of the Genero Application Server................................................ 29
    Architecture overview....................................................................................... 29
    Reliability inherent in the architecture............................................................. 31
    Development architecture (standalone GAS)................................................... 31
    Deployment (production) architecture.............................................................. 31
    Services Pool (GWS Only).............................................................................. 33
  Components of the Genero Application Server................................................ 36
    What is a dispatcher?.......................................................................................... 36
    What is a proxy?................................................................................................. 37
    What is a DVM?................................................................................................. 38
  What is auto logout?........................................................................................... 38
  What is delegation?............................................................................................. 38
  What is Single sign-on (SSO)?............................................................................. 38
  GAS directories.................................................................................................. 39
  Application environment...................................................................................... 43
  Internationalization............................................................................................. 44
    Encoding Architecture....................................................................................... 44
    Charsets Configuration...................................................................................... 45
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick start: deploying applications</td>
<td>257</td>
</tr>
<tr>
<td>What is a Genero Archive?</td>
<td>256</td>
</tr>
<tr>
<td>Accessing Genero Web Services</td>
<td>253</td>
</tr>
<tr>
<td>Genero Web Client for HTML5 (GWC-HTML5)</td>
<td>252</td>
</tr>
<tr>
<td>Genero Web Client for JavaScript (GWC-JS)</td>
<td>254</td>
</tr>
<tr>
<td>Load balancing</td>
<td>170</td>
</tr>
<tr>
<td>Customizing GWC-JS applications</td>
<td>203</td>
</tr>
<tr>
<td>Migrating from GDC to GWC-JS</td>
<td>245</td>
</tr>
<tr>
<td>Migrate from GWC-HTML5 to GWC-JS</td>
<td>251</td>
</tr>
<tr>
<td>Configure multiple dispatchers</td>
<td>156</td>
</tr>
<tr>
<td>How to implement Single sign-on (SSO)</td>
<td>122</td>
</tr>
<tr>
<td>OpenID Connect SSO</td>
<td>123</td>
</tr>
<tr>
<td>OpenID SSO</td>
<td>128</td>
</tr>
<tr>
<td>SAML SSO</td>
<td>136</td>
</tr>
<tr>
<td>How to implement custom single sign-on</td>
<td>146</td>
</tr>
<tr>
<td>Connect to the application database with SSO</td>
<td>152</td>
</tr>
<tr>
<td>How to implement delegation</td>
<td>113</td>
</tr>
<tr>
<td>Configure GRE in local mode</td>
<td>157</td>
</tr>
<tr>
<td>Configure GRE in distributed mode</td>
<td>158</td>
</tr>
<tr>
<td>Resource deployment overview</td>
<td>257</td>
</tr>
<tr>
<td>Deploying with Genero Archive</td>
<td>256</td>
</tr>
<tr>
<td>What is a Genero Archive?</td>
<td>256</td>
</tr>
<tr>
<td>Quick start: deploying applications</td>
<td>257</td>
</tr>
<tr>
<td>Resource deployment overview</td>
<td>257</td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>The fglxslp command.................................................. 308</td>
<td></td>
</tr>
<tr>
<td>The fglgar command..................................................... 308</td>
<td></td>
</tr>
<tr>
<td>The gasadmin command................................................. 310</td>
<td></td>
</tr>
<tr>
<td>Ghost Client and Testing Tools................................. 316</td>
<td></td>
</tr>
<tr>
<td>Automatic discovery of User Agent (adua.xrd).................. 335</td>
<td></td>
</tr>
<tr>
<td>What is an Output Map?.................................................. 336</td>
<td></td>
</tr>
<tr>
<td>How an Output Map is chosen......................................... 336</td>
<td></td>
</tr>
<tr>
<td>Modify the adua.xrd file to specify custom Output Maps........ 337</td>
<td></td>
</tr>
<tr>
<td>Specify the Output Map in the application URI.................. 337</td>
<td></td>
</tr>
<tr>
<td>ADUA Syntax Diagrams.................................................... 338</td>
<td></td>
</tr>
<tr>
<td>adua.xrd usage example................................................. 339</td>
<td></td>
</tr>
<tr>
<td>GAS Predefined resources.............................................. 340</td>
<td></td>
</tr>
<tr>
<td>GAS predefined resources overview............................... 340</td>
<td></td>
</tr>
<tr>
<td>Common GAS predefined resources................................. 341</td>
<td></td>
</tr>
<tr>
<td>GAS Configuration Reference........................................... 341</td>
<td></td>
</tr>
<tr>
<td>GAS configuration file................................................... 342</td>
<td></td>
</tr>
<tr>
<td>Application configuration files...................................... 342</td>
<td></td>
</tr>
<tr>
<td>Configuration file hierarchies........................................ 343</td>
<td></td>
</tr>
<tr>
<td>Configuration file elements.......................................... 349</td>
<td></td>
</tr>
<tr>
<td>Glossary and Acronyms................................................. 424</td>
<td></td>
</tr>
<tr>
<td>Legal Notices.............................................................. 426</td>
<td></td>
</tr>
</tbody>
</table>
## Overview

- What's New v3.00 (maintenance releases)
- What's New v3.00
- What is the Genero Application Server? on page 11
- Standalone Genero Application Server on page 14
- Front-ends and Extensions on page 15

## Quick Start Guide

- Explore Genero application server resources on page 16
- Quick start guide for applications with UI on page 17
- Quick start guide for web services applications on page 23
- Genero demo applications on page 27

## Basics

- Architecture of the Genero Application Server on page 29
- Components of the Genero Application Server on page 36
- What is auto logout? on page 38
- What is delegation? on page 38
- What is Single sign-on (SSO)? on page 38
- GAS directories on page 39
- Application environment on page 43
- Components of the Genero Application Server on page 36
- Application Web Address on page 51

## Configuration

- System Requirements on page 61
- GAS Configuration Check on page 62
- Licensing on page 65
- ISAPI Extension Installation and Web Server Configuration on page 70
- FastCGI Installation and Web Server Configuration on page 89
- Java Servlet Installation and Web Server Configuration on page 96
- Validating configuration files on page 98
- Configuring applications on GAS on page 100
- How to implement delegation on page 113
- How to implement Single sign-on (SSO) on page 122
- Compression in Genero Application Server on page 154
- Configure development environment on page 156
- Configure multiple dispatchers on page 156

## Administering the Genero Application Server on page 161

## Developing Web applications on page 184

## Deploying with Genero Archive on page 256

## How to implement Single sign-on (SSO) on page 122
<table>
<thead>
<tr>
<th>Reference on page 304</th>
<th>Tools and Commands on page 304</th>
<th>Upgrading on page 273</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Automatic discovery of User Agent (adua.xrd) on page 335</td>
<td>• Dispatcher: httpdispatch on page 304</td>
<td>• New Features of the Genero Application Server on page 273</td>
</tr>
<tr>
<td>• GAS Predefined resources on page 340</td>
<td>• Dispatcher: fastcgidispatch on page 306</td>
<td>• Upgrade Guides for the Genero Application Server on page 287</td>
</tr>
<tr>
<td>• GAS Configuration Reference on page 341</td>
<td>• Dispatcher: java-j2eedispatch on page 307</td>
<td>•</td>
</tr>
</tbody>
</table>
What's new in Genero Application Server (GAS), v 3.00 (Maintenance Releases)

A summary of new features and changes in functionality added for Genero Application Server (GAS) version 3.00 Maintenance Releases (MRs).

**Important:** Please read What's new in Genero Application Server, v 3.00 on page 275, for a list of features that were introduced with Genero 3.00 General Availability release.

**Table 1: Genero Web Client for JavaScript (GWC-JS), Version 3.00 (MRs)**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhancements for the GWC-JS (v1.00.31):</td>
<td>No additional reference.</td>
</tr>
<tr>
<td>• Native mobile browsers are supported on iOS (Mobile Safari) and Android™ (Mobile Chrome).</td>
<td></td>
</tr>
<tr>
<td>Enhancements for the GWC-JS (v1.00.29):</td>
<td>For details about Mozilla® Firefox®, see User agents on page 61.</td>
</tr>
<tr>
<td>• Support for Mozilla® Firefox® added.</td>
<td>For details about the browserMultiPage style, see the UserInterface style attributes topic in the Genero Business Development Language User Guide.</td>
</tr>
<tr>
<td>• The browserMultiPage style can be used to specify whether the RUN and RUN WITHOUT WAITING instructions will be executed in the same browser tab or in a new browser tab.</td>
<td></td>
</tr>
<tr>
<td>Enhancements for the GWC-JS (v1.00.28):</td>
<td>For details about activating the debug mode, see GWC-JS application in debug mode on page 197.</td>
</tr>
<tr>
<td>• Debug mode activation is simplified.</td>
<td>For details about the allowWebSelection presentation style, see Table Style Attributes in the Genero Business Development Language User Guide.</td>
</tr>
<tr>
<td>• The allowWebSelection presentation style enables content selection from a table with a simple click-and-drag of the cursor.</td>
<td></td>
</tr>
<tr>
<td>• The grunt command supports the creation of multiple customizations in the distribution folder.</td>
<td></td>
</tr>
<tr>
<td>• Richtext is supported.</td>
<td></td>
</tr>
<tr>
<td>• Referencing a resource (such as an image) has been simplified. Use a path relative to index.html.</td>
<td></td>
</tr>
<tr>
<td>• Support for applications using right-to-left languages, such as Arabic, is added.</td>
<td></td>
</tr>
<tr>
<td>Enhancements for the GWC-JS (v1.00.16):</td>
<td>See Features and limitations on page 187.</td>
</tr>
<tr>
<td>• Canvas elements are supported.</td>
<td></td>
</tr>
</tbody>
</table>
### Overview

- Front calls `setvar` and `getvar` are supported for session variable management. See the *Genero Business Development Language User Guide* for more details about their usage.

Enhancements for the GWC-JS (v1.00.16):
- The GWC-JS has a log player feature, which allows you to play back a session log in the browser.

### Table 2: Engine and Architecture, Version 3.00 (MRs)

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>GWC_JS_LOOKUP_PATH</code> element (added as a child of <code>INTERFACE_TO_CONNECTOR</code> on page 381) allows you to configure the location of your custom GWC-JS front-end.</td>
<td>See <code>GWC_JS_LOOKUP_PATH</code> on page 376</td>
</tr>
<tr>
<td>A new URI dedicated to the lookup of the GWC-JS directory. The complete format of the URI is <code>ua/w/$(GWC-JS)/&lt;filename&gt;</code>.</td>
<td>See Application URIs on page 51</td>
</tr>
<tr>
<td>Enhancements for the GAS (v3.00.21):</td>
<td></td>
</tr>
<tr>
<td>• A method to add custom HTTP headers for HTTP responses is provided.</td>
<td>See HTTP on page 378, <code>SESSION_COOKIE</code> on page 406</td>
</tr>
<tr>
<td>• The <code>SESSION_COOKIE</code> element can be set so that cookies are created with the &quot;secure&quot; flag.</td>
<td></td>
</tr>
<tr>
<td>The GAS (v3.00.43 onwards) supports a new user agent output feature called <code>UA_OUTPUT_COMPONENT</code> which can be defined in the component for web applications. If set, an application will inherit the specified UA settings.</td>
<td>See <code>UA_OUTPUT_COMPONENT</code> on page 415</td>
</tr>
</tbody>
</table>

### Table 3: Deployment, Version 3.00 (MRs)

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>WEB_COMPONENT_DIRECTORY</code> element allows for multiple paths to be specified.</td>
<td>See <code>WEB_COMPONENT_DIRECTORY</code> on page 422</td>
</tr>
</tbody>
</table>

**Note:** The new features listed in this topic are available in the latest version of the GAS. Contact your support channel for more details.
Genero Application Server overview

The Genero Application Server (GAS) provides you with a server environment to create Genero applications and to deploy and run them on front-end clients through various protocols, proxies and dispatchers.

- What's New v3.00 (maintenance releases)
- What's New v3.00
- What is the Genero Application Server? on page 11
- Standalone Genero Application Server on page 14
- Front-ends and Extensions on page 15

What is the Genero Application Server?

The Genero Application Server (GAS) is an engine that delivers Genero applications for various Genero front-ends in both development and production environments.

Manages Communication between front-end and DVM

The GAS creates relationships between various front-ends and the Dynamic Virtual Machines (DVMs) which run the applications.

Embeds a Web Server

A Web server to handle requests from the Internet is embedded in the GAS. It includes dispatcher and proxy processes. Communication between the Web server and the GAS is handled by dispatchers.

Simplifies Application Deployment

The GAS simplifies the deployment phase by taking care of the connection to the applications. For applications deployed to Web clients no software installation or configuration is needed on the client; only a browser is required to access the program.

Controls Interaction between DVM and front-ends

The GAS supports the development of Genero Business Development Language (BDL) applications on a single source code stream that can be run on both a browser or on a desktop. If the same application is delivered to either the Genero Web Client for JavaScript (GWC-JS) or the Genero Desktop Client (GDC), the GAS handles the communication with the DVM through its proxy and dispatcher components in much the same way. See the examples, GAS Role in GWC-JS Application Delivery on page 12 and GAS Role in GDC Application Delivery on page 12.

Provides Genero Web Services (GWS) for clients

The GAS can also be configured to provide Genero Web Services (GWS) for clients. GWS DVMs are managed in a pool by the GAS to provide resources to clients when requested. See the example GAS Role in GWS on page 13.
GAS Role in GWC-JS Application Delivery
The GWC-JS front-end is provided as part of the GAS installation. This front-end allows users to run applications from their browser. The following describes the processes involved, highlighting the role of the GAS and its components as it delivers an application to the browser.

1. A request is sent to the dispatcher to run the application from the browser.
2. The dispatcher checks the application configuration in the configuration files (xcf, xrd) and routes the request to the required proxy.
3. The proxy starts a DVM using configuration files and runs the application.
4. The DVM returns an Abstract User Interface (AUI) tree describing the objects of the user interface and sends rendering instructions to the proxy.
5. The client browser interprets the DVM instructions and builds the Web interfaces from widget components that are defined by Cascading Style Sheets (CSS), JavaScript, and HTML code to provide the dynamic behavior for the application.

GAS Role in GDC Application Delivery
The GDC front-end allows you to run applications locally using native screens on your Windows™, Linux®, or MAC® OS® for user interaction. The following example highlights the role of the GAS and its components as a GDC application starts up.
Figure 2: GDC application delivery

1. A request is sent to the dispatcher to run the application from the GDC.
2. The dispatcher checks the application configuration in the configuration files (xcf, xrd) and routes the request to the required proxy.
3. The proxy starts a DVM using configuration files and runs the application.
4. The DVM returns an Abstract User Interface (AUI) tree describing the objects of the user interface and sends rendering instructions to the proxy.
5. The GDC interprets the DVM instructions and AUI tree to create the screens natively on your system so as to provide the user interface.

GAS Role in GWS

The GAS allows you to provide Web services to clients. Web Services configured on your GAS installation are started automatically when the GAS starts and services listen for requests from clients. For more information on Web services, see What is Web Service? on page 23.

The following example highlights the role of the GAS and its components in exposing a Web service to a client that requests the functions of its services over the internet via the hypertext transfer protocol (HTTP).
Figure 3: GWS Server/Client

1. An HTTP request to perform a function of the Web service is sent to the dispatcher from the client.
2. The dispatcher checks the application configuration in the configuration files (xcf, xrd) and routes the request to the GWS proxy.
3. The GWSProxy is in charge of the pool of DVMs that will serve the Web service application, and perform the requested functions.
4. The DVM returns a HTTP response with requested data, and response codes to indicate success or failure.
5. The client interprets the HTTP response instructions and processes the returned data, for example, for display.

- Standalone Genero Application Server on page 14
- What is a dispatcher? on page 36
- What is a proxy? on page 37
- Front-ends and Extensions on page 15

**Standalone Genero Application Server**

With the support of the HTTP protocol, the Genero Application Server provides a direct connection for access to applications without using a Web server.

The standalone server (see Dispatcher: httpdispatch on page 304) is provided for the development cycle only, allowing you to remove the Web server from your development architecture. For production environments, a Web server is mandatory.
Front-ends and Extensions

The Genero Application Server can serve applications using various front-ends and extensions.

**Genero Desktop Client (GDC)**

The Genero Desktop Client allows you to run the application through the GAS, yet deliver the application locally using the GDC. For more information about the GDC, refer to the *Genero Desktop Client User Guide*.

**Genero Web Client for JavaScript (GWC-JS)**

The GWC-JS allows you to deliver Genero applications in a Web browser on the client machine. It is a JavaScript client that works with Node technology. The Genero Web Client is provided as part of the Genero Application Server installation. For more information about the GWC-JS, see *What is GWC-JS?* on page 184.

**Genero Web Services (GWS)**

Genero Web Services allows you to implement Web services. Web services are a standard way of communicating between applications over the internet or an intranet. A web service can be a server that exposes services or a client that consumes a service. For more information, see *What is Web Service?* on page 23.

**Genero Web Client (GWC-HTML5)**

The Genero Web Client for HTML5 allows you to deliver Genero applications in a Web browser without having to install any software on the client machine. It uses browser-based themes. The Genero Web Client is provided as part of the Genero Application Server installation. This version of the GWC has been deprecated; new development should use the GWC-JS instead. For more information about the Genero Web Client for HTML5, refer to the *Genero Application Server 2.50 User Guide*. 
To give you an idea of what the GAS does and help you to get started, this section guides you with examples for configuring, running, and deploying basic types of applications on the GAS.

- **Explore Genero application server resources** on page 16
- **Quick start guide for applications with UI** on page 17
- **Quick start guide for web services applications** on page 23
- **Genero demo applications** on page 27

### Explore Genero application server resources

Paths to files and directories of your GAS installation are set by resources in the GAS configuration file, `as.xcf`. Knowing the absolute path of resources, helps you when working with GAS.

See the **GAS configuration file** on page 342.

**Note:** Resources are like variables that identify or name the resource. For example `res.path.as` identifies the GAS installation directory while the value of this resource will contain the absolute path to your GAS installation directory.

```xml
<RESOURCE Id="res.path.as" Source="INTERNAL">C:\4js\gas\2.50.34</RESOURCE>
```

When, for example, you deploy applications you do not need to know where the real resources are actually located in the production environments because you can map to real resources with a reference using this syntax, `$\{RESOURCE Id\}`. Therefore `$\{res.path.as\}` references the GAS installation directory in all hosts where GAS is installed.

### Typical resources and relative path locations

The following are some typical resources and relative path locations to some of the more common installation and application data files which you will need to run or reference:

- `$\{res.path.as\}/etc/as.xcf`
- `$\{res.path.as\}/bin/httpdispatch.exe`
- `$\{res.fgldir\}/fglrun.exe`
- `$\{res.appdata.path\}/app`
- `$\{res.appdata.path\}/deployment`

**Note:** Installation directories may also be identified by environment variables which are set at installation time by script files, see **Table 4: FGLASDIR directories and files** on page 40.

1. To find the absolute path of resources, you will need to first locate your `as.xcf` file where these predefined resources are set.
   The `as.xcf` file is an XML file which contains the default configuration for the GAS. You must search your disk for it in directories where the file is likely to be located.

2. In a text editor or with Genero Studio, open the `as.xcf` file and locate the source of your `$\{res.path.as\}` resource. You should find its source path amongst the `RESOURCE` list elements for your platform (e.g. WNT or UNIX).
   The source path for `$\{res.path.as\}` is platform dependent.
   - On Linux®/UNIX™, it is also represented by the environment variable FGLASDIR.
   - On Windows™ by the environment variable FGLASDIR.
Knowing the source of this resource, you will be able to locate the binary file of, for example the Dispatcher: httpdispatch on page 304, the standalone dispatcher used by the GAS.

3. In the as.xcf file, locate and note the source of your $(res.appdata.path) resource.
   - On Linux®/UNIX™ it is also represented by FGLASDIR/appdata
   - On Windows™, for example, C:/ProgramData/<vendor>/gas/<gas_version>.

**Quick start guide for applications with UI**

Quick start guides to help you configure, run, and deploy an application with user interface (UI)

- Launch first application on page 17
- Configure an application on page 19
- Run an application on page 20
- Deploy an application on page 22

**Launch first application**

This quick start guide provides you with the steps to launch the Genero Application Server and view demo applications delivered by the GAS to both Genero Web Client for JavaScript (GWC-JS) and Genero Desktop Client (GDC).

Before you begin, you must:
- Have the Genero product suite installed locally.
- Have Genero Studio installed (which by default includes the Genero Business Development Language, Genero Application Server, and Genero Desktop Client).

The goal of this quick start is to provide you with some basic experience in using the Genero Application Server to start a demo application. In this example, we use the standalone GAS dispatcher (httpdispatch), which limits this quick start to a completely local install but simplifies the process by bypassing the need for a Web server.

**Note:** The standalone GAS dispatcher is for development and testing only, a Web server is required for a production environment.

1. Start the standalone dispatcher from the command line, for details about starting httpdispatch see Dispatcher: httpdispatch on page 304.
   The application server (the httpdispatch dispatcher) is started.

2. Open the GAS demos page from a browser by entering the address, http://localhost:6394/demos.html.
   **Note:** By default, access to the demos applications is allowed only to localhost (127.0.0.1). If you want to enable it for other client machines / IP addresses, you must define access in the ACCESS_CONTROL on page 351 element. For more information on GAS configuration see GAS configuration file on page 342.

   The Genero Application Server responds, and you should see the Genero Application Server welcome page displayed. This indicates that the GAS dispatcher is working.

3. Examine the demo application delivered by the Genero Application Server to two front-ends; the GWC-JS and the GDC (see Front-ends and Extensions on page 15).
To view the demo application displayed using the Genero Web Client, click on the **demos** link. Alternatively, you can enter the address [http://localhost:6394/ua/r/gwc-demo](http://localhost:6394/ua/r/gwc-demo). (For more information on the application URI see Application URIs on page 51)

The application displays in the browser.

![Demo application launched by the GWC-JS](image)

Figure 4: Demo application launched by the GWC-JS

4. To run the demo application using the Genero Desktop Client for HTTP, complete the following steps:

   Typically applications are run by the GDC monitor by configuring a HTTP shortcut. For more information see the Create a HTTP Connection shortcut page in the Genero Desktop Client User Guide. Follow these steps to download a shortcut via the browser using the da protocol.

   
   b) In the system dialog that opens, you can elect to open or save the **gwc-demo.gdc** shortcut. To open it, select the Genero Desktop Client from the **Open with** list and click **OK**.
Configure an application

Before you run an application you need to configure it so that it can be executed by the Genero Application Server.

The goal of this quick start is to provide you with some basic experience in configuring information needed by the Genero Application Server to start an application. You provide these details in a separate application-specific configuration file, see Create an application configuration file on page 103 (one per application).
For the purposes of this quick start, you can create a custom configuration file for the **HelloWorld** application located in your Genero Studio’s installation GSTDIR/samples directory.

1. Create a new directory (e.g. you can name it "HelloWorld_config") on your disk where you will store the **HelloWorld** application source files.

2. Copy all the files from GSTDIR/samples/HelloWorld directory to your new local directory.

3. Create a minimal configuration file for your **HelloWorld** application. Provide an absolute path to the location of your compiled application files in the **PATH** element, and in the **MODULE** element specify the module required to launch your application.

Use a text editor or if you are using Studio, go to **File > New > Web/AS > Application Configuration (.xcf)**

**Note:** The **Parent** attribute references defaultgwc, which provides default configuration for all GWC applications (see **GAS configuration file** on page 342).

```xml
<?xml version="1.0" encoding="UTF-8" ?>
  <EXECUTION>
    <PATH><path_to_your_local_directory></PATH>
    <MODULE>HelloWorld.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

4. Name your file with the same name as the application (this is not mandatory but it may help you identify the file), e.g. HelloWorld, with the **xcf** extension.

5. Save the configuration file in your $(res.appdata.path)/app directory.

The **default** directory for external application configuration files is the $(res.appdata.path)/app directory, see **Explore Genero application server resources** on page 16. The **default directory resource** is set in the **GAS configuration file** on page 342 file’s **GROUP** element:

```xml
<GROUP Id="_default">$(res.path.app)</GROUP>
```

You have successfully configured an application.

**What to do next**

When you have completed the above steps, your next task is to test your application to see if it is configured correctly as detailed in **Run an application** on page 20.

**Run an application**

After configuring your application, you can test it to see if it is configured correctly by running it on the GAS. There are several ways of running your application.

For the purposes of this quick start, you can run the "HelloWorld" application you have already configured, see **Configure an application** on page 19.

Before you begin, you must:

- Have the Genero product suite installed locally.
- You must have Genero Studio installed (which by default includes the Genero Business Development Language, Genero Application Server, and Genero Desktop Client (GDC)).

Refer to the **Genero Installation Guide** for installation guidance.

The goal of this quick start is to provide you with some basic experience of running applications in GAS using the two front-ends: Genero Web Client for JavaScript (GWC-JS), GDC, and from Genero Studio.

1. Run your application on the GWC-JS front-end.
a) If your standalone dispatcher is not running, start it from the command line, for details about starting httpdispatch see Dispatcher: httpdispatch on page 304

b) To run your HelloWorld application, in a browser enter the application address:

   http://localhost:6394/ua/r/HelloWorld

   For more information on the application URI see Application URIs on page 51.

   You have launched the application in the GAS.

2. To run your application on GDC, complete the following steps:

   Typically applications are run by the GDC monitor by configuring a HTTP shortcut. For more information see the Create a HTTP Connection shortcut page in the Genero Desktop Client User Guide. Follow these steps to download a shortcut via the browser using the da protocol.

   a) Enter the address in a browser tab

   http://localhost:6394/da/r/HelloWorld

   b) In the system dialog that opens, you can elect to open or save the HelloWorld.gdc shortcut file.

   To open the application, select the Genero Desktop Client from the Open with list and click OK.

   The GDC monitor launches and the HelloWorld application opens in a system window.

3. To run your application from within Genero Studio, complete the following sub-steps:

   a) If your standalone dispatcher is running, close your open applications and shut down the dispatcher.

   To shut down the standalone dispatcher, see Dispatcher: httpdispatch on page 304.

   You do not need to start the GAS dispatcher if you are running an application from within Genero Studio.

   b) In Genero Studio, the combobox in the bottom right corner of the main window displays the currently active configuration. Make sure the selected option is <GAS version> Desktop.

   c) Select the HelloWorld application from the Projects panel.

   If the application is not listed in your Projects panel, you must search your disk for HelloWorld.4pw in directories where the file is likely to be located.

   The HelloWorld project opens in the Projects panel.

   d) Run your HelloWorld application.

   Select Debug > Execute.

   The application opens in a GDC window.

4. To run your application from within Genero Studio to the GDC using HTTP, complete the following sub-steps:

   a) In Genero Studio, the combobox in the bottom right corner of the main window displays the currently active configuration. Select the <GAS version> (GWC) configuration option instead of <GAS version> Desktop to have Studio launch the application for GWC client.

   b) Run your HelloWorld application.

   Select Debug > Execute.

   The application opens in a browser.

   You have successfully run your application using available options for both desktop and Web front-ends.

   **What to do next**

   When you have completed the above steps, your next task is to deploy your application as detailed in Deploy an application on page 22.
Deploy an application
When you have your application configured correctly and running on the GAS, you are now ready to package the files required to deploy it as an application.

This topic provides you with steps to configure and deploy an application that you can test on your own machine.

For the purposes of this quick start, you can use the configuration files created for the HelloWorld.xcf application, see Configure an application on page 19.

1. Update the application configuration file <PATH> element as follows:

   `<PATH>$(res.deployment.path)</PATH>`

2. Create a new directory where you will archive the application's source files (you can name it, for example, "helloworld_deploy").

3. Copy all the HelloWorld application source files from $GSTDIR/samples to the archive directory.
   
   **Note:** If you are deploying resources (e.g. images or Web components) with your application, these need to go in dedicated directories in the archive. For details about building an archive with public resources, please see Quick start: Genero Archive on page 259.

4. Copy the updated application configuration file (e.g. HelloWorld.xcf) to the directory with the application source files.

5. In the same directory, create a MANIFEST file (see The MANIFEST file on page 263) and save it with the name "MANIFEST" (without extension).

   `<MANIFEST>
   <DESCRIPTION/></DESCRIPTION>
   <APPLICATION xcf="HelloWorld.xcf"/>
   </MANIFEST>`

6. Create an archive (gar) file to deploy your application by performing the following steps:

   a) From the command line, navigate to the directory that contains the application source files AND the MANIFEST file.

   b) Enter the command: fglgar --gar.

      A Genero archive file (gar) is created in your current directory that has the same name as the directory. See The fglgar command on page 308.

Deploy your application on your machine

**About this task:**

Once you have configured your application for deployment and created an archive for it in the steps above, you can now deploy your application locally on your machine to test it as described in the next steps.

1. Deploy your (gar) file locally on your machine.

   To deploy an archive named HelloWorld_deploy.gar:

   ```
   gasadmin --deploy-archive HelloWorld_deploy
   ```

   A subdirectory is created in your $(res.deployment.path) directory identified by the archive name and the date and time deployed, e.g. HelloWorld_deploy-20150423-130838. All the files contained in archive (gar) are placed in the directory.

2. Enable your deployed application locally on your GAS.

   To list all deployed archives:

   ```
   gasadmin --list-archives
   ```
To enable the archive, reference it by its archive name:

gasadmin --enable-archive HelloWorld_deploy

This enables the application by copying its configuration file (e.g. HelloWorld.xcf) to your
$(res.appdata.path) directory.

**Run the deployed application**

**About this task:**

Once you have deployed your application on your machine in the steps above, you can now run your application locally to test it as described in the next steps.

1. Start the standalone dispatcher from the command line by typing `httpdispatch`.
2. In a browser enter the address of your deployed application, e.g. `http://localhost:6394/ua/r/HelloWorld`
   The Genero Application Server responds, and you should see your application displayed and be able to interact with it. You have successfully deployed an application.

---

**Quick start guide for web services applications**

Quick start guides to help you configure, run, and deploy a web service application

- What is Web Service? on page 23
- Explore Genero Web service (server side) on page 23
- Configure a Web service on page 24
- Run a client application on Web service using GWC on page 25
- Deploy a Web service on page 25

**What is Web Service?**

Web service is an interface where data is exchanged between applications instead of users.

A Web service provides data as a service over the HTTP protocol. Web services allow applications built using different technologies to communicate with each other. Typically, web services use the SOAP or REST protocols to define the communication and structure of messages, while XML or JSON are the formats used for the data exchanged.

Examples of web services that you may be familiar with are those providing weather or news updates that you can use on your site or application; you can see an example of this type of web service in Genero's RSS demo application. See Genero demo applications on page 27. For more information on Web service, see the Web service topics in the Genero Business Development Language User Guide.

**Explore Genero Web service (server side)**

You can explore the Web services that the GAS can deliver by launching demo applications that invoke them.

1. Start the standalone dispatcher from the command line, for details about starting `httpdispatch` see Dispatcher: `httpdispatch` on page 304.
   The application server (the `httpdispatch` dispatcher) is started.
2. To show for example that the Web service called **Calculator** is working, you can retrieve its Web Service Description Location (WSDL)
   Enter the address `http://localhost:6394/ws/r/demo/Calculator?WSDL` in a browser
**Note:** The WSDL provides you with details such as the address location of the service. For more information on WSDL, please see the *Genero Business Development Language User Guide*.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tempuri.org/" name="Calculator"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/"
  xmlns:fjs="http://tempuri.org/
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  ...
  <wsdl:service name="Calculator">
    <wsdl:port name="CalculatorPortType" binding="fjs:CalculatorBinding">
      <soap:address location="http://localhost:6394/ws/r/demo/Calculator"/>
    </wsdl:port>
    <wsdl:port name="CalculatorPortTypeSoap12"
      binding="fjs:CalculatorBindingSoap12">
      <soap12:address location="http://localhost:6394/ws/r/demo/Calculator"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

3. Start an application that uses the **Calculator** Web service by performing the following steps:
   a) Now that the GWS has started the Web service, return to the **Demos** tab of your browser. In the **Topic** tree, navigate to **WebServices** > **Calculator** > **Client.** Double-click on the demo **Calculator Soap 1.2**. The **Web Services URL** dialog appears. From the drop down menu select the URL stating `http://localhost:6394/ws/r/demo/Calculator`.
   b) In the **Web Services URL** dialog, click **OK**. You should see a calculator screen and be able to interact with it.

**Configure a Web service**

Create a configuration file with the details the Genero Application Server needs to deliver the Web service to client applications.

The configuration file for a Web services is very similar to that used for applications.

For the purposes of this quick start, you can create a custom configuration file for the **Calculator** web service located in your `FGLASDIR/demo/WebServices/calculator/` directory.

1. Create a new directory (e.g. you can name it "calculator_config") on your disk where you will store the Calculator web service source files.
2. Copy all the files from `FGLASDIR/demo/WebServices/calculator/server` to your new local directory.
3. Create a minimal configuration file for your Calculator Web service. Provide a path to the location of your compiled application files in the **PATH** element, and in the **MODULE** element specify the module required to launch your Web service.

Use a text editor or if you are using Studio, go to **File >> New >> Web/AS >> Application Configuration (.xcf)**.

**Note:** The **Parent** attribute reference to `ws.default` provides default configuration for all applications of the Web service type (see *GAS configuration file* on page 342).
4. Save your Web service configuration file, e.g. "mycalculator.xcf", in the directory specified in the $(res.path.services) resource.

   The default services directory is set by the res.path.services resource in the GAS configuration file on page 342 file:

   ```xml
   <RESOURCE Id="res.path.services" Source="INTERNAL">$(res.appdata.path)/services</RESOURCE>
   ```

Run a client application on Web service using GWC

After configuring your Web service, you can test it to see if it is configured correctly by starting the Web service and running a client application with it on the standalone GWS. For production, you will use the GAS.

For this quick start, you must have Genero Business Development Language, and Genero Application Server installed. Refer to the Genero Installation Guide for installation guidance.

For the purposes of this quick start, you can run the mycalculator web service you have already configured, see Configure a Web service on page 24 and use the calculator client application available from the GAS demos page, see Display demo applications with the Genero Web Client on page 27, to interact with it.

1. Start the standalone dispatcher from the command line, for details about starting httpdispatch see Dispatcher: httpdispatch on page 304.

2. To check that the mycalculator web service is reachable, you can retrieve its WSDL. To perform this, enter the address of the Web service application, e.g. http://localhost:6394/ws/r/mycalculator?WSDL, in a browser tab.

3. To interact with the mycalculator service and show it is working, open another browser tab and enter the address of the demo applications, http://localhost:6394/ua/r/gwc-demo.


   The Web Services URL dialog appears. From the drop down menu select the option (Customize)
   Click here to add your own URL and provide the URL to your configured web service, e.g. http://localhost:6394/ws/r/mycalculator.

5. In the Web Services URL dialog, click OK.

   You should see a calculator screen and be able to interact with it.

Deploy a Web service

When you have your Web service application configured correctly and running on the GWS, you are now ready to package the files required to deploy it as a Web service on the Genero Application Server.

For the purposes of this quick start, you can use the configuration files created for the Calculator Web service, see Configure a Web service on page 24.

1. Update the Web service configuration file, i.e. Calculator.xcf <PATH> element as follows:

   ```xml
   <PATH>$(res.deployment.path)/server</PATH>
   ```

2. Create a new directory where you will archive the Web service source files (e.g. you can name it "calculator_deploy") and in it create a sub-directory called "server".

3. Copy the application source files to the archive directory as follows:
   a) Copy all the files from FGLASDIR/demo/WebServices/calculator/server to the server subdirectory in your archive directory
b) Copy the updated Web service configuration file, Calculator.xcf, to the server subdirectory.

4. Create a MANIFEST file and save it with the name "MANIFEST" (without extension) to the directory that contains the application source files. For more information see The MANIFEST file on page 263.

```
<MANIFEST>
  <DESCRIPTION>This archive contains one service</DESCRIPTION>
  <SERVICE xcf="Calculator.xcf" />
</MANIFEST>
```

5. Create an archive (gar) file to deploy your application with the following steps:
   a) From the command line, navigate to the archive directory that contains the application source files AND the MANIFEST file.
   b) Enter the command: fglgar --gar.
      A Genero archive (gar) file is created in your current directory that has the same name as the directory.

Test the deployment on your standalone dispatcher.

6. To deploy your (gar) file locally on your GAS, unpack its files in your $(res.deployment.path) deployment directory.

   To deploy an archive named calculator_deploy.gar:

   ```
   gasadmin --deploy-archive calculator_deploy.gar
   ```

   A subdirectory is created in your $(res.deployment.path) directory identified by the archive name and the date and time deployed, e.g. calculator_deploy-20150423-130838. All the files contained in archive (gar) are placed in the directory.

7. To enable your deployed Web service locally on your GAS, perform the following steps:

   To list all deployed archives:

   ```
   gasadmin --list-archives
   ```

   To enable the archive, reference it by its archive name:

   ```
   gasadmin --enable-archive calculator_deploy
   ```

   This enables the Web server by copying the configuration file to the $(res.path.services) directory.

   To test the deployed Web service, start the Calculator server and run a client application that uses the service by performing the following steps:

8. Start the standalone dispatcher from the command line, for details about starting httpdispatch see Dispatcher: httpdispatch on page 304.

9. To start the Calculator web service, in a browser tab enter the address of the web service application:

   ```
   http://localhost:6394/ws/r/Calculator
   ```

10. To interact with the Calculator service and show it is working, open another browser tab and enter the address of the demo applications:

    ```
    http://localhost:6394/ua/r/gwc-demo
    ```

    The Web Services URL dialog appears.

12. From the drop down menu select one of the following options:
    - http://localhost:6394/ws/r/demo/Calculator
• Or select the option **(Customize)** [Click here to add your own URL](#) and provide the URL to your configured web service.

Click **OK** when finished
You should see a calculator screen and be able to interact with it. You have successfully deployed a Web service.

The steps to configure and deploy a web service on your own machine shown here, can be adapted for deployment on GAS installation on other hosts.

---

**Genero demo applications**

A variety of demo applications are provided to demonstrate Genero functionality.

• [Find the demo applications](#) on page 27
• [Display demo applications with the Genero Web Client](#) on page 27

**Find the demo applications**

Demo applications are provided as part of the Genero Business Development Language with Web Services and Genero Studio installation.

**Demos included with Genero Studio**

We have demos that are bundled with Genero Studio. Access the demo applications from the [Tutorials & Samples](#) tab.

From your file system, you can find these demos within **My Genero Files/samples**.

**Demos bundled with Genero BDL**

• `$FGLDIR/demo` provides the Genero Business Development Language demos.
• `$FGLDIR/web_utilities` provides additional materials for delegation services (SAML, SSO, tutorials, and more).

**Display demo applications with the Genero Web Client**

If you are looking for code snippets and examples, you are encouraged to view the demo programs included with the Genero Application Server.

**Accessing the demos page**

The demo application is defined in the GAS configuration file with an `Id` of `gwc-demo`

```xml
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$\{res.path.fgldir.demo\}</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```

**Note:** By default, access to the demos applications is allowed only to localhost (127.0.0.1). If you want to enable it for other client machines / IP addresses, you must define access in the **ACCESS_CONTROL** on page 351 element. For more information on GAS configuration see **GAS configuration file** on page 342.
To access the GAS gwc-demo application, you can use the Genero Web Client user interface, see GWC-JS home page on page 194 or you can enter this URL:

http://localhost:6394/demos.html

From this page, you can click on the Genero demos link to open the demos application. The GAS must be running (standalone) or must be integrated with a Web server and able to start the required proxies and DVMs.

The demos application can also be accessed directly

You can access the demos application directly by entering the following URL:

http://localhost:6394/ua/r/gwc-demo
GAS Basics

These topics provide an architecture overview, highlight the main features of the GAS, and provide an insight into how the GAS delivers applications.

- Architecture of the Genero Application Server on page 29
- Components of the Genero Application Server on page 36
- What is auto logout? on page 38
- What is delegation? on page 38
- What is Single sign-on (SSO)? on page 38
- GAS directories on page 39
- Application environment on page 43
- Internationalization on page 44
- Application Web Address on page 51

Architecture of the Genero Application Server

For an administrator, it is important to understand the different architectures available for the Genero Application Server, and the implications of each architecture choice.

- Architecture overview on page 29
- Reliability inherent in the architecture on page 31
- Development architecture (standalone GAS) on page 31
- Deployment (production) architecture on page 31
- Services Pool (GWS Only) on page 33

Architecture overview

The architecture of the Genero Application Server uses dispatchers and proxies for optimal reliability, performance and integration in web servers.

The role of the dispatcher is to forward each new incoming request to the appropriate proxy (uaproxy or gwsproxy). The dispatcher handles the GAS configuration and keeps a persistent session table of all proxies it has started. In case of failure, the web server restarts the dispatcher, which uses the session table to reconnect to the proxies (and therefore to the applications).
Components

- Web Server
- GAS Dispatchers
- VMProxies
- DVMs
- Database Server

Note: The Genero Application Server and the Genero BDL runtime should be installed on the same machine.

How it works: the high-level overview

1. In order to request an application, the end-user enters a URI that specifies which application to launch (based upon the GAS configuration file and application configuration files). For example, the alias to serve up the GWC demo application via a web server would be `http://mywebserver/gas/ua/r/gwc-demo`. In development environments, it is possible to exclude the Web Server. For more information, see Architecture for Development (Standalone GAS).

2. The Web Server routes the request to the GAS dispatcher. GAS dispatchers refer to the connectors in charge of dispatching a GAS request to the appropriate proxy. There are different GAS dispatchers, each designed for a specific Web Server. For example, the fastcgidispacth.exe is for use with FASTCGI-compliant Web Servers such as Apache, while the isapidispatch.dll is for the Information Internet Services (IIS) Web Server.

3. The GAS Dispatcher starts the VMProxy to handle the request. Each session requesting an application results in a VMProxy starting up; as a result, you will likely see multiple proxies running concurrently. The type of proxy started (uaproxy or gwsproxy) will depend on the application being requested. The dispatcher will route to the correct proxy. The dispatcher tracks the session and proxy information in a persistent session table. The presence of this information in the session table ensures that if a dispatcher is killed or restarted, the information needed to return to the proxy and running application is still present. For more information on the responsibilities of the GAS dispatcher, see GAS Dispatcher responsibilities.
4. The VMProxy then launches the DVM for the requested application. It handles any child DVMs, keeps the DVM connections up, and handles the requests and responses appropriate for the type of proxy. For more information of the VMProxy responsibilities, see VMProxy responsibilities.

5. The DVM interacts with the database server, as needed.

**Reliability inherent in the architecture**

The architecture of the Genero Application Server supports reliability.

- If an application is running, and the dispatcher is killed, the session information is saved. When the dispatcher restarts, the application continues from where it left off.
- With the one-to-one relationship between a VMProxy and a running application, you can stop or kill an application's VMProxy without affecting any other applications that are running concurrently. If two applications are running and you kill the VMProxy for one application, the other application is not affected.
- The architecture provides capabilities for tuning your system.

**Development architecture (standalone GAS)**

Use the standalone GAS when developing your applications. The simplified architecture of the standalone GAS removes the Web server from the environment, allowing the developer to concentrate on the application. The standalone GAS is for development only, provided to simplify your development setup and configuration.

The `httpdispatch` process allows you to connect directly to a GAS dispatcher without involving a Web server. This direct connection is provided to simplify the setup of development environments, and is the typical connection method used during development.

![Architecture for stand-alone GAS](image)

**Figure 8: Architecture for stand-alone GAS**

To use the standalone GAS, simply start the `httpdispatch` process. On Windows® machines, this can be started from the Start menu. On Linux®, you start the process from the command line.

Once the process is started, connect by providing the machine name and port number. These examples assume you are connecting from the local machine and have not changed the default port number):

- `http://localhost:6394/ua/r/gwc-demo` opens the GWC demo application.

**Important:** The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.

**Deployment (production) architecture**

For deployment and production systems, you must include a Web server.

A Web server:

- Enables load balancing.
- Handles HTTPS.
- Handles authentication. You can use the Web server to handle authentication. Alternatively, you can use a single sign-on solution.
GAS deployment example 1: all-in-one machine

You can have the Genero Application Server on the same machine as the Web server.

GAS deployment example 2: Database on separate server

GAS deployment example 3: Behind frontal web server
If you already have a Web server (server A) on the internet (a non-protected area / inside a DMZ) and do not want to have Genero on this server, you will need to add another Web server (shown in Server 1 in the drawing) to manage the Genero Application Server. This second Web server sits in the protected network. The frontal Web server forwards application requests to the internal web server on server 1.

It is recommended that production deployment be supervised by a Web specialist to avoid security issues.

**Services Pool (GWS Only)**

Requests for Web services are processed by the GWS proxy and are managed using the DVM pool. The examples will assist you in configuring the pool element of your Web service.

**Why do we need Services Pool for GWS?**

The main reason is due to the way the Dynamic Virtual Machines (DVM) is unable to support more than one application at a time, see Reliability inherent in the architecture on page 31. The GWS proxy needs to launch new DVM (fglrun process) for each request on the Web server. If traffic to a Web server was to become very high with, for example, hundreds of requests at any period, this could potentially result in slow response time or even overloading of the server.

To avoid unnecessary use of resources, therefore, and to respond as fast as possible to client requests, the GWS proxy manages DVM processes in a pool. From the pool it can release inactive DVMs or launch new ones when required; up to the maximum number of DVM processes specified for the application. These parameters are defined in the `POOL` element of the Application Server configuration file.

**Important:** When sticky mode is used, the `POOL` on page 395 is disregarded. For instance, the `MAXAVAILABLE` setting, limiting the number of DVMs available to the GWS, is no longer taken into account. Therefore, it is recommended to handle the stopping of the sticky Web service in your application code with a dedicated method to be called by the user agent when it needs to close the session.

**Releasing DVMs not Actively Processing Requests**

Genero Web Service DVMs are not shutdown immediately after they stop processing requests. Instead DVM shutdown is optimized by the `gwsproxy` from request statistics and frequency of use that best
determine the use of resources at runtime. The \texttt{gwsproxy} calculates this based on a combination of the following factors:

**Time to Start a New DVM**

The \texttt{gwsproxy} waits at least the time it takes to start a new DVM before deciding if a DVM can be shutdown.

**Three Times the Last Request Execution Time**

If there is only one request, a DVM shutdown takes place after waiting three times the request execution time from when the last request took place.

**Three Times the Average Request Frequency**

If there is more than one request, a DVM is shutdown after waiting three times the average request frequency. For example, the GWS proxy calculates the elapsed time since it received the last new incoming request. If this is three times greater than the average frequency of new requests it has been receiving, it stops one inactive DVM.

The time to shutdown a DVM will therefore vary depending on how great the Web service's request load has been for the previous period but eventually DVMs are released to reach the value specified by \texttt{MIN\_AVAILABLE} because the \texttt{gwsproxy} calculation is also bound by a minimum (one second) and a maximum (ten minutes) limit.

**Minimum (one second)**

If time to shutdown is less than one second, the \texttt{gwsproxy} waits a full one second before shutting down the DVM.

**Maximum (ten minutes)**

If time to shutdown is more than ten minutes, the \texttt{gwsproxy} waits no longer than ten minutes before shutting down a DVM.

**Example 1: One GWSProxy Starts three DVMs**

Assume the following values have been specified for a Web service application:

```xml
<POOL>
  <START>3</START>
  <MIN\_AVAILABLE>2</MIN\_AVAILABLE>
  <MAX\_AVAILABLE>5</MAX\_AVAILABLE>
</POOL>
```

When the Genero Application Server first starts, the \texttt{START} element defines how many DVMs to start for a particular Web service. There is one GWSProxy in charge of the pool of DVMs for the Web service. For our example, this means that one GWSProxy will launch three DVMs.
Example 2: GWSProxy Releases Inactive DVMs

While the `START` element defines the number of DVMs to start initially, DVMs that are not actively processing requests can be released based on GAS statistics. For example, the GWS proxy calculates the elapsed time since it received the last new incoming request. If this is three times greater than the average frequency of new requests it has been receiving, it stops one inactive DVM. See Releasing DVMs not Actively Processing Requests on page 33.

Continuing with our example, if all of the DVMs are not actively processing requests, then one DVM will eventually be released, bringing the total number of DVMs to the `MIN_AVAILABLE` amount of two.

Example 3: GWSProxy Launches new DVMs when Required up to `MAX_AVAILABLE`

As requests come in, the GWS proxy determines whether there is a need to start up new DVMs. For the number of pending requests in the queue, the GWS proxy computes the average request execution time against the time to start a DVM. If dispatching all pending requests over the active DVMs takes less time than starting a new DVM, no new DVM will be started. In other words, a new DVM will only be started if it will help to decrease the waiting time of all pending requests. At most, `MAX_AVAILABLE` DVMs can be started.

Continuing with our example, up to five DVMs can be launched to handle requests.

Example 4: GWSProxy Managing a Connection Queue

What happens when there are `MAX_AVAILABLE` DVMs actively processing requests, and a new request comes in? The new request is placed in a connection queue, waiting for a DVM to become available. The new requests could (in theory) be waiting indefinitely, except:

- There is the option of a timeout in the Web server to handle infinite wait.
Any Web service client can define its own timeout limit. If a client is willing to wait indefinitely for a Web service response, it is permitted.

Continuing with our example, this means that if all five DVMs are actively processing requests, and a sixth requests comes in, that request is placed in the connection queue until a DVM is available to process the request, or a timeout is reached based on settings in either the Web server or the Web service client.

**Components of the Genero Application Server**

UI or Web service applications, and where they are launched from, have specific requirements of the GAS. The GAS performs its function by routing requests to the required process or component so as to deliver applications correctly.

- What is a dispatcher? on page 36
- What is a proxy? on page 37
- What is a DVM? on page 38

**What is a dispatcher?**

Understand what the dispatcher does and identify which dispatcher to use with a specific Web server.

**Types of GAS Dispatchers**

There are different GAS dispatchers, each designed for a specific Web server:

- **httpdispatch**: standalone dispatcher for development only, provided to simplify your development setup and configuration through a direct connection without a Web server. See Dispatcher: httpdispatch on page 304.
- **isapidispatch**: dispatcher for Internet Information Services (IIS). See Dispatcher: isapidispatch.
- **fastcgidispatch**: dispatcher for Fast CGI compliant Web servers like Apache, see Dispatcher: fastcgidispatch on page 306

Each dispatcher performs the same role of forwarding an application request to the appropriate proxy, and the proxy in turn processes the request by launching the DVM.
GAS Dispatcher and VMProxy

The GAS dispatcher starts a VMProxy to handle the application request. Each session requesting an application results in a VMProxy starting up, so several VMProxies may be running concurrently. The dispatcher tracks the session and proxy information in a persistent session table and routes requests to the correct proxy.

Note: The presence of this information in the session table ensures that if a dispatcher is killed or restarted, the information needed to return to the proxy and running application is still present.

GAS Dispatcher responsibilities

The GAS Dispatcher, in summary, is responsible for the following:

• Launching VMProxies
• Handling and validating the application or service configuration
• Providing the application configuration to the VMProxy via environment variables
• Handling a persistent and shared session table that manages the forwarding of application requests to the corresponding VMProxies
• Stopping the VMProxies when the Web Server shuts down
• Handling static file requests

What is a proxy?

Proxies deliver applications on the GAS. Being aware of the different types of proxies there are, assists you in developing and deploying applications accordingly.

Types of VMProxies

There are two types of GAS VMProxies, each designed for a specific type of application:

• uaproxy: a universal proxy for applications using Genero Desktop Client (GDC), Genero Web Client for JavaScript (GWC-JS) interfaces, Genero Mobile for Android (GMA), Genero Mobile for iOS (GMI) see Proxy: uaproxy on page 307
• gwsproxy: proxy for Genero Web Service (GWS) type applications, see Proxy: gwsproxy on page 308

VMProxy responsibilities

Each proxy performs the same role of starting the Dynamic Virtual Machines (DVM) for the application and handling the connection, and application requests and responses.

In general, a VMProxy is responsible for the following:

• Launching the DVM (see What is a DVM? on page 38)
• Handling child DVMs
• Maintaining the DVM connections

Additional responsibilities depend on the VMProxy type:

uaproxy

The uaproxy is responsible for the following:

• Handles HTTP client-side front-end (CSF) requests (see USER_AGENT on page 416)
• Manages sessions when the client is GDC, GWC-JS, GMA or GMI

gwsproxy

The gwsproxy is responsible for the following:

• Handles the GWS DVM pool (see Services Pool (GWS Only) on page 33)
• Handles HTTP Web Services requests
• Forwards HTTP Web Services responses
  (SOAP, REST, XML over HTTP)

What is a DVM?
The Dynamic Virtual Machine (DVM) is the software or runtime system (fglrun) where applications’
business logic is processed. The DVM executes Genero BDL code to retrieve data, it responds to incoming
service requests, and dispatches output to the service.

What is auto logout?
The GAS supports a timeout feature called AUTO_LOGOUT which can be configured to display a log out
page after a specified time of user inactivity on a Genero Web Client (GWC) application is detected.

What is delegation?
With the delegation mechanism, the GAS is able to delegate the start of a Web application or a Web
service to another Genero REST service in order to perform some controls before granting access and
starting the application.

When you configure delegation, it introduces an additional step in the Genero Application Server workflow
in order to, for example, perform some controls such as authentication, authorization, monitoring, or
whatever is required before the requested application or service is started.

  **Note:** Delegation is required if you want to enable Single-sign-on (SSO) authentication for remote
access to applications, see What is Single sign-on (SSO)? on page 38).

What is Single sign-on (SSO)?
Single sign-on allows a user to enter one name and password to access multiple applications. Getting to
know the features of the different options, helps you select an SSO mechanism that is appropriate for your
needs.

For more information on Single Sign-on and how to implement it see How to implement Single sign-on
(SSO) on page 122.

Genero Application Server supports various kinds of Single sign-on.

**OpenID Connect**
OpenID Connect is the latest evolution of the OpenID authentication technology used for Web applications
that handle many users. OpenID Connect is intended for public Web applications. You have to be
registered on one of the trusted identity providers so that users can be authenticated with Single sign-on on
different web sites. You can exchange custom information (attributes) on the identity.

See OpenID Connect SSO on page 123.

**OpenID**
OpenID is used for standard Web applications that handle many users. OpenID is intended for public Web
applications. A user has the same identifier that he can use on different web sites. Information maintained
on the identity of the user is limited.

See OpenID SSO on page 128.
SAML

SAML is used for standard Web applications that handle many users. SAML is intended for private or intranet Web applications. You have to be referenced on one of the trusted identity providers. You can exchange custom information (attributes) on the identity.

See SAML SSO on page 136

GAS directories

GAS is installed in different directories in a Linux®/UNIX™ platform to a Windows® one. To help you manage your GAS installation, there are descriptions of its files and directories, and recommendations for its use.

The two main directories of the GAS are:

**Installation directory**

The installation directory is where the installation files are located. It is defined by the FGLASDIR environment variable and set by the resource $(res.path.as) in the GAS configuration file, see GAS configuration file on page 342.

**Application directory**

The application directory is where the application data files managed by the GAS are located. The "appdata" directory, as it is commonly known, is set by the resource $(res.appdata.path) in the GAS configuration file.

GAS has the following prerequisites for application data:

- The appdata directory and all the search paths for VM server image files defined by the FGLIMAGEPATH environment variable must be located on the same file system. For more details on FGLIMAGEPATH, see the Genero Business Development Language User Guide.
- The DVM uses hard links to avoid file copies. As hard links are direct pointers to data on the disk, they can not span file systems.

**Note:** It is not recommended to change the location of appdata.

If you need to customized the application file directory location due to, for example, disk space constraints, it is recommended that you move the entire directory hierarchy starting from the root directory at appdata and ensure that all appdata files are relocated on the same file system.

**Caution:** If you have two partitions, partition A and B, they are considered two distinct file systems even if stored on the same physical device.

To implement a location change on the GAS for appdata, there are two recommended options:

- You can reset the resource $(res.appdata.path) in the GAS configuration file with an absolute path reference
to the new location. The example shows appdata located in the path of the GAS installation resource, $(res.path.as):

```
<RESOURCE Id="res.appdata.path" Source="INTERNAL">$(res.path.as)/appdata</RESOURCE>
```

- Or alternatively, at the command line you can override the GAS configuration file $(res.appdata.path) resource with dispatcher option `-E`. For more information on using dispatcher options, see the relevant dispatcher page: Dispatcher: httpdispatch on page 304, Dispatcher: fastcgidispatch on page 306.

Table 4: FGLASDIR directories and files

<table>
<thead>
<tr>
<th>Directory/Subdirectories</th>
<th>Files or Details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FGLASDIR</td>
<td>Contains script for setting environment variables:&lt;br&gt;On Linux®/UNIX™:&lt;br&gt;envas&lt;br&gt;On Windows®:&lt;br&gt;envas.bat</td>
<td>GAS installation directory.</td>
</tr>
<tr>
<td>/bin</td>
<td>*.exe</td>
<td>GAS dispatchers and VMProxy executables.</td>
</tr>
<tr>
<td>/etc</td>
<td>as.xcf</td>
<td>Default GAS configuration file.</td>
</tr>
<tr>
<td>/ISAPI</td>
<td>isapidispatch.ini</td>
<td>ISAPI extension configuration file.</td>
</tr>
<tr>
<td>/lib</td>
<td></td>
<td>Library files.</td>
</tr>
<tr>
<td>/license</td>
<td>*.txt</td>
<td>End user license agreement and notice regarding third party products.</td>
</tr>
<tr>
<td>/release</td>
<td>readme.txt</td>
<td>Copyright notice and release notes.</td>
</tr>
<tr>
<td>/shortcut</td>
<td>*.gdc</td>
<td>Shortcuts for applications that can be run by the Genero Desktop Client using /da/r URLs. See Running a desktop application on page 58</td>
</tr>
<tr>
<td>Directory/Subdirectories</td>
<td>Files or Details</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>$FGLASDIR/web</td>
<td>demos.html</td>
<td>Root directory for direct communication to the application server. The demos.html file provides access to Demos applications.</td>
</tr>
<tr>
<td>$FGLASDIR/gwc-js</td>
<td>The _default text file contains the directory name of the GWC-JS client, which is used to start a Web application.</td>
<td>One or more GWC-JS clients are packaged with the GAS. Client directories are in the format of gwc-js-version, for example gwc-js-1.10.01. See GWC-JS application bootstrap on page 202.</td>
</tr>
</tbody>
</table>

Table 5: appdata Directories and Files

<table>
<thead>
<tr>
<th>Directory/Subdirectories</th>
<th>Files or Details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appdata</td>
<td></td>
<td>GAS application data directory.</td>
</tr>
<tr>
<td></td>
<td>On Linux®/UNIX™: $FGLASDIR/appdata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On Windows®: C: \ProgramData \vendor\gas \gas_version</td>
<td></td>
</tr>
<tr>
<td>appdata/app</td>
<td>*.xcf</td>
<td>Default application group; the default location for your application configuration files.</td>
</tr>
<tr>
<td>appdata/deployment</td>
<td>Contains a directory for each application.</td>
<td>Applications deployed with Genero Archive.</td>
</tr>
</tbody>
</table>
| appdata/public          |                 | A public resource path for all applications.  
  **Caution:** Public images should not be placed in the /public root directory as the fglrun does not look for images to be served via the GAS there; searches start in subdirectory paths, i.e. public/common and public/deployment. |
<p>| /common                 |                 | A resource for common images used by applications. It is the default PUBLIC_IMAGEPATH directory. |
| /deployment             |                 | Contains public images deployed with applications by the deployment framework (fglgar), see Resource deployment overview on page 257. |
| appdata/log             |                 | <strong>Note:</strong> Users need write permissions for this directory. Users who installed the Genero Application Server have write |</p>
<table>
<thead>
<tr>
<th>Directory/Subdirectories</th>
<th>Files or Details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>permissions. If users that start the Genero Application Server (for example Apache®) are not in that group, then you need to grant them write permission.</td>
</tr>
<tr>
<td>/dispatcher_name</td>
<td></td>
<td>A directory is created for each dispatcher, for example, httpdispatch (for the standalone GAS). The logs for a given day are stored in a directory named for that date. See Logging on page 167. <strong>Note:</strong> DVM logs are redirected to files when DAILYFILE is set for the log output type, see OUTPUT (under LOG) on page 388.</td>
</tr>
<tr>
<td>/gasadmin</td>
<td></td>
<td>The logs for gasadmin actions for a given day are stored in a directory named for that date. <strong>Note:</strong> Users need write permissions for this directory. Users who installed the Genero Application Server have write permissions. If users that start the Genero Application Server (for example Apache®) are not in that group, then you need to grant them write permission. <strong>Caution:</strong> The session directory should only be used to store session files, which are required to reconnect dispatchers to proxies. It should not contain other files or subdirectories.</td>
</tr>
<tr>
<td>appdata/session</td>
<td></td>
<td>Persistent session table information. <strong>Note:</strong> Users need write permissions for this directory. Users who installed the Genero Application Server have write permissions. If users that start the Genero Application Server (for example Apache®) are not in that group, then you need to grant them write permission. <strong>Caution:</strong> The session directory should only be used to store session files, which are required to reconnect dispatchers to proxies. It should not contain other files or subdirectories.</td>
</tr>
<tr>
<td>/dispatcher_name</td>
<td></td>
<td>A directory is created for each dispatcher.</td>
</tr>
<tr>
<td>/gasadmin</td>
<td></td>
<td>The directory may exist, but it is not used by gasadmin.</td>
</tr>
<tr>
<td>appdata/tmp</td>
<td></td>
<td>Default file transfer directory. <strong>Note:</strong> Users need write permissions for this directory. Users who installed the Genero Application Server have write permissions. If users that start the Genero Application Server (for example Apache®) are not in that group, then you need to grant them write permission.</td>
</tr>
<tr>
<td>/dispatcher_name</td>
<td></td>
<td>A directory is created for each dispatcher.</td>
</tr>
</tbody>
</table>
**Gas Basics**

<table>
<thead>
<tr>
<th>Directory/Subdirectories</th>
<th>Files or Details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/gasadmin</td>
<td></td>
<td>The gasadmin directory stores gasadmin actions, such as exploring the configuration (.xcf) files.</td>
</tr>
<tr>
<td>appdata/services</td>
<td>*.xcf</td>
<td>Default services group; the default directory for your Web services configuration files.</td>
</tr>
</tbody>
</table>

**Application environment**

When the Genero Application Server starts an application process, it sets environment variables from various sources. Understanding how variables are defined by the various front-ends, is helpful to you when configuring applications.

The application process can be started as a DVM (fglrun), as an intermediate script, or whatever is specified in the application configuration file.

**Environment inheritance**

Environment variable settings are inherited by the DVM in the following order:

1. The environment of the dispatcher that starts the proxy.
2. The environment variables defined in the application configuration file (`ENVIRONMENT_VARIABLE` elements).
3. Additionally, some specific environment variables can be defined by the front-end; whether this is Genero Desktop Client, Genero Web Client. See:
   - How GDC and GWC handle environment variables on page 43
   - How GWS handle HTTP request headers on page 43
   - How GWS with delegation handles HTTP request headers on page 44

**How GDC and GWC handle environment variables**

With Genero Desktop Client and Genero Web Client applications, variables can be set in the HTTP request that starts the application. All HTTP request headers will be transformed into environment variables, by adding the `FGL_WEBSERVER_HTTP_` prefix.

**Note:** Dash (or minus) (–) characters are replaced by underscore (_) characters. For example, the header "User-Agent" will define the `FGL_WEBSERVER_HTTP_USER_AGENT` environment variable.

The following exceptions are supported for backward compatibility:

- `FGL_WEBSERVER_REMOTE_USER` holds the `REMOTE_USER` value provided by the Web Server.
- `FGL_WEBSERVER_REMOTE_ADDR` holds the `REMOTE_ADDR` value provided by the Web Server.

**How GWS handle HTTP request headers**

With Genero Web Services applications, environment variables cannot be used to pass HTTP request headers. Low-level APIs are used to retrieve values from the HTTP request headers. The header name has the following form:

`X-FourJs-Environment-Parameter-name`
Where `name` is the header name. To get the header value, you must use the `com.HTTPRequest.getRequestHeader()` method. For example:

```javascript
LET param = req.getRequestHeader("X-FourJs-Environment-Parameter-MyHeaderValue")
```

The server name can be found in `X-FourJs-Environment-Variable-SERVER_NAME`.

**Note:** For more information on `com.HTTPRequest` methods, please see the [Web services classes](#) chapter in the [Genero Business Development Language User Guide](#).

How GWS with delegation handles HTTP request headers

When delegation is used in Genero Web Services, the `X-FourJs-Environment-Parameter-<parameter_name>` is used by the delegate mechanism to pass any additional parameters defined in the `as.xcf` file of the application to the delegate service. For instance, in the following `xcf` delegation setting example using Security Assertion Markup Language (SAML) service:

```xml
<DELEGATE service="services/SAMLServiceProvider">
  <IDFORMAT>ABC</IDFORMAT>
  <AUTHCONTEXT>123</AUTHCONTEXT>
</DELEGATE>
```

The parameters `IDFORMAT` and `AUTHCONTEXT` are free, and can be passed to the delegate service as HTTP headers in the following form:

- `X-FourJs-Environment-Parameter-IDFORMAT : ABC`
- `X-FourJs-Environment-Parameter-AUTHCONTEXT : 123`

### Internationalization

How the Genero Application Server handles international applications.

- Encoding Architecture on page 44
-Charsets Configuration on page 45
- Translations for GWC-JS on page 49

**Note:** You can customize rendering engine output encoding as well as preferred input encoding. You are also able to use User Agent-preferred encodings.

### Encoding Architecture

International applications use one or more character sets to support different languages.

Character set encodings are used in different areas such as configuration files, templates, operating system interaction and user applications.

This diagram summarizes the GAS character set encoding architecture:
Figure 9: GAS Encoding Architecture

The GAS uses:

• XML File Encoding
• Template Charset
• Operating system locale

The DVM uses:

• Operating system locale

Charsets Configuration

Charsets can be defined in four places.

1. With environment locales when launching a DVM.
2. In HTML charset in template.
3. Inside XML files used by the GAS.
4. With environment locales when launching the GAS.

- DVM Locale on page 45
- HTML charset on page 46
- XML Encoding on page 47
- GAS System Encoding on page 47
- Default Encoding on page 48
- Configure GWC-JS for right-to-left languages on page 48

DVM Locale

If application files (such as .4gl, .per, .4st files) contain characters in a specific encoding, the DVM has to run in this encoding.

Setting a DVM locale is described in the Genero Business Development Language User Guide, chapter "Localization".
On Linux®/UNIX™, the DVM locale is defined by the `LANG` or `LC_ALL` environment variables, which can be specified in the GAS executing environment, or with the `ENVIRONMENT_VARIABLE` on page 369 child of a `WEB_APPLICATION_EXECUTION_COMPONENT` on page 417 element, inside the as.xcf file.

**Example in as.xcf defining a KOI8-R (Russian) charset for the DVM:**

```xml
<?xml version="1.0" encoding="UTF-8"?>
...
<COMPONENT_LIST>
    <WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
        <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
        <ENVIRONMENT_VARIABLE Id="FGLGUI">$(res.fglgui)</ENVIRONMENT_VARIABLE>
        <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
        ...
        <ENVIRONMENT_VARIABLE Id="LC_ALL">ru_RU.KO8-R</ENVIRONMENT_VARIABLE>
        <DVM>$(res.dvm.wa)</DVM>
    </EXECUTION_COMPONENT>
...
</COMPONENT_LIST>
```

Such `WEB_APPLICATION_EXECUTION_COMPONENT` definition can then be referenced by an `APPLICATION` (for an application) on page 354 element using the `<EXECUTION` Using="identifier"> tag.

**Example in as.xcf defining a KOI8-R (Russian) charset for the DVM:**

**Important:** Keep in mind that the character set used by the server at runtime (during program execution) must match the character set used to write programs. So source files must be created/edited in the encoding of the server where `fglcomp` and `fglrun` will be executed.

**HTML charset**

In order to correctly handle application data in the user agent, the HTML page charset needs to be set.

As the Genero Application Server (GAS) generates HTML5 pages from templates, and uses the `FGLASDIR/web/gwc-js/bootstrap.html` file for the Genero Web Client for JavaScript (GWC-JS) front-end, the charset is defined in the prolog or metadata of these files. The charset defined here takes precedence over system locale settings.

**Example**

The code example comes from the `bootstrap.html` using the UTF-8 encoding:

```html
<!DOCTYPE html>
<!--
FOURJS_START_COPYRIGHT(D,2014)
Property of Four Js*
(c) Copyright Four Js 2014, 2019. All Rights Reserved.
* Trademark of Four Js Development Tools Europe Ltd
in the United States and elsewhere

This file can be modified by licensees according to the product manual.
FOURJS_END_COPYRIGHT
-->
<html>
<head>
```

```html
```
```
**XML Encoding**

Configuration for the Genero Application Server involves XML files (with xcf file extension).

These XML files can include international characters and multiple languages so the UNICODE standard UTF-8 is typically used. How to define an encoding in an XML file is described in *Extensible Markup Language - Character Encoding*. The charset defined in XML files takes precedence over system locale settings.

**Example in as.xcf with UTF-8 (UNICODE) character set:**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- comments -->
<CONFIGURATION ...
```

**Example in as.xcf with ISO-8859-6 (Arabic) character set:**

```xml
<?xml version="1.0" encoding="ISO-8859-6" standalone="yes"?>
<!-- comments -->
<CONFIGURATION ...
```

**GAS System Encoding**

System character encoding matters when Genero Application Server (GAS) interacts with the operating system.

For example, when the GAS:

- Writes log files.
- Opens files defined in the GAS configuration file (`as.xcf`).
- Reads arguments from the command line.

In these cases and more, GAS uses the character encoding set by the operating system:

- **Linux®/UNIX™**: encoding is defined via environment variables `LANG` or `LC_ALL`. For more information see the "Localization" topics in *Genero Business Development Language User Guide* or see The Single UNIX - Specification Version 2 - Locale.

  **Note:** There should be no need to set the `LANG` variable, except your application uses a different character set to the Windows system locale.

- **Windows®**: GAS defaults to the system locale as defined in the language and regional settings.

**How GAS does character set conversion**

The GAS software takes care of character set conversions:

- For `xcf` files, it does the conversion based on what the XML prolog specifies as charset to the GAS locale, for example:
• For the front-ends such as Genero Web Client for JavaScript (GWC-JS) that use UTF-8 encoding:
  • For template files, the charset in the metadata of $FGLASDIR/web/gwc-js/bootstrap.html file is used.
  • For the DVM (V3) the conversion is done in the DVM locale, see DVM Locale on page 45.

  Note: Operating system character sets may have different names across operating systems. To unify character set names in the application server environment, the GAS manages a character set encoding name conversion to map the operating system character encoding name to a canonical name:
  • A charset.alias file is provided. This file is located in the $FGLASDIR/etc directory of the GAS.

Default Encoding
By default, GAS uses UTF-8 encoding for handling all Unicode characters.

See Unicode characters for more information.

Configure GWC-JS for right-to-left languages
If your Genero Web Client for JavaScript (GWC-JS) application uses a language such as Arabic, you need to set up your application for right-to-left language, and set the correct locale.

Make sure that the locale you want to use is installed on your system. If the locale is not installed, follow your operating system's requirements to install locales.

For example, for Arabic support on a Linux®/UNIX™ platform, make sure Arabic locales are installed (for example, check that the locale -a command displays the ar_DZ.iso88596 locale).

1. In the application configuration file (application.xcf), include a reference to the LANG environment variable.
   In our example LANG is set for the Arabic language in Algeria:
   On Linux®/UNIX™ systems
   ```xml
   <ENVIRONMENT_VARIABLE Id="LANG">ar_DZ.iso88596</ENVIRONMENT_VARIABLE>
   ```
   On Windows®
   ```xml
   <ENVIRONMENT_VARIABLE Id="LANG">Arabic_Algeria.28596</ENVIRONMENT_VARIABLE>
   ```

2. In your .4st file, set the reverse presentation style attribute to "yes" for the UserInterface node, in order to get a mirrored display of your forms:
   ```xml
   ... 
   <Style name="UserInterface">
     <StyleAttribute name="reverse" value="yes" />
   </Style>
   ...
   ```

For more details, see the Right-to-left languages support topic in the Localization chapter of Genero Business Development Language User Guide
**Translations for GWC-JS**

Add your custom translation texts in the locale file and reference them in the HTML code in the widget template file.

The Genero Web Client for JavaScript (GWC-JS) front-end provides support for language selection based on locale. This topic details how you can use translation texts to provide localization mainly to text in the following GWC-JS front-end components:

- Welcome Page
- Ending Page
- Calendar Widget
- Contextual Menu For Tables
- File Transfer Dialog

You will find the locale files in your `project_dir/src/locales/xx-YY.json`, where "xx-YY" follows the standard localization code used for languages, for example, en-US, de-DE, fr-FR, etc.

**Excerpt from US English Locale File**

Locale files are json files which have a typical json structure of keys and values in quotes separated by colons, ":". A sample from the US English locale file is shown.

```json
{
  "gwc": {
    "lngName": "English",
    "app": {
      "waiting": "Waiting for connection",
      "restart": "Restart the same application",
      "run": "Run",
      ...
    }
  }
}
```

**Retrieving Localized Texts**

The GWC-JS is able to retrieve the required translated text by referencing the HTML language attribute `data-i18n` in the widget template files.

```html
data-i18n="key"
```

Where "key" is the localization key in the locale file.

**In the Widget file**

This sample from the `project_dir/src/js/base/widget/widgets/application/SessionEndWidget.tpl.html` shows default text for the ending page component.

```html
<span class="" data-i18n="gwc.app.restart">Restart the same application</span>
```

GWC-JS will replace the text "Restart the same application" within the `<span>` element with that referenced by the `gwc.app.restart` key in the locale file. If the key is not found, the GWC-JS falls back to the default locale, "en-US". If the entry is still not found, then the translation value will be the key name.

**In the JavaScript**

Internationalization is done by the GWC-JS JavaScript function `i18n.t`. This function takes the localization key as argument and returns the translation text.
For example retrieving the value of the key “gwc.app.restart”:

```javascript
i18n.t('gwc.app.restart')
```

### Selecting a Language

By default, the GWC-JS language defaults to the browser language but the user can change the interface language from within the user interface of an application open in a browser window by performing the following steps:

- Selecting the **Settings** menu from the **developer** toolbar.

![GWC-JS User Interface Menus](image)

**Figure 10: GWC-JS User Interface Menus**

- Choosing the language from the **Interface language** list in the **Stored Settings configuration** window that pops up.
The name of the language, like "English", that you see displayed is defined in the locale file by the entry `gwc.lngName`, for example, as in the `en-US.json`:

```json
{
    "gwc": {
        "lngName" : "English",
        [...]
    }
}
```

**Application Web Address**

To access an application, you specify the necessary information in the browser's address bar by entering in the appropriate application URI.

**Note:** For details on the URI for integrated application servers, see:
- GAS ISAPI Installation / Web Server Configuration
- GAS FastCGI Installation / Web Server Configuration

- URIs acknowledged by the GAS on page 51
- URI Examples on page 57

**URIs acknowledged by the GAS**

The URIs acknowledged by the Genero Application Server fall into two categories: application URIs and file serving URIs.

- Application URIs on page 51
- File serving URIs on page 55

**Application URIs**

To access a Web application, you enter the appropriate application URI in your browser's address bar.

**URI Syntax**

This topic describes the URIs acknowledged by the GAS based on the standard syntax shown below and explained in Table 6: Explanation of URI syntax options on page 52. This topic provides you with URIs
to access your Genero applications. The information is intended to help you tune and monitor your Web server configuration for Genero.

```plaintext
http[s]://
{  
  web-server[:web-server-port]  
    [  
      /directory [...]  
    ]  
  |  
  app-server[:app-server-port]  
}  
/scope  
/action  
/group  
/  
{  
  web-application-id  
}  
[  
  ?  
  parameter=parameter-value  
    [  
      &  
      parameter=parameter-value  
    ]  
  [...]  
]
```

### Table 6: Explanation of URI syntax options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>web-server</td>
<td>Name or IP address of the Web Server</td>
<td>e.g. localhost</td>
</tr>
<tr>
<td>web-server-port</td>
<td>Port on which the Web Server listens</td>
<td>e.g. 1100</td>
</tr>
</tbody>
</table>
| directory     | A directory or virtual directory on the Web Server defined by the $(connector.uri) resource in as.xcf | Typically this directory is called "gas".  
  
  **Note:** If you are using a direct connection to the GAS like the standalone dispatcher, httpdispatch, the resource `connector.uri` is not used, instead the URI syntax will look like this, `http://localhost:port/ua/r/app-name` |
| app-server    | Name or IP address of the Application Server                                | e.g. localhost             |
| app-server-port| Port on which the Application Server listens                                | e.g. 6394                  |
| scope         | Scope or protocol you are working on, e.g. uaproxy or Web service          | ua, ws                     |
### Option Description Valid Values

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Action requested of the Application Server</td>
<td>See Application URIs acknowledged by the GAS</td>
</tr>
<tr>
<td>group</td>
<td>Application group defined in as.xcf</td>
<td>see GROUP (for an application) on page 373</td>
</tr>
<tr>
<td>web-application-id</td>
<td>Web application identifier or name</td>
<td>e.g. gwc-demo</td>
</tr>
<tr>
<td>parameter</td>
<td>Parameter to communicate to the Application Server to start application with arguments</td>
<td>See Starting applications with arguments on page 58</td>
</tr>
<tr>
<td>parameter-value</td>
<td>Parameter value</td>
<td>See Set gwc-js customization with query string parameter on page 58</td>
</tr>
</tbody>
</table>

### Table 7: Application URIs Acknowledged by the GAS

<table>
<thead>
<tr>
<th>URI path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ua/r</td>
<td>Genero Web Client for JavaScript (GWC-JS) application is started with a /ua/r request. See Running a Web application on page 58.</td>
</tr>
<tr>
<td>/ua/sua</td>
<td>Once the startup request has been answered, the DVM uses /ua/sua requests to submit user actions and communicate with the front-end.</td>
</tr>
<tr>
<td>/ua/ft</td>
<td>GWC-JS file transfer</td>
</tr>
<tr>
<td>/ua/interrupt</td>
<td>Front-end client uses the interrupt request to send an interrupt message to the DVM.</td>
</tr>
<tr>
<td>/ua/i</td>
<td>Resource, e.g. a fully-qualified URL pointing to an application's image file, see Table 8: File serving URIs on page 55</td>
</tr>
<tr>
<td>/ua/components</td>
<td>Web component resource, e.g. a fully-qualified URL pointing to an application's web component file, see Table 8: File serving URIs on page 55</td>
</tr>
<tr>
<td>/ua/close</td>
<td>Front-end sends a close request to the DVM to stop an application.</td>
</tr>
<tr>
<td>/ua/newtask</td>
<td>Front-end sends a newtask request to ask the GAS if it has any pending applications that have not been already notified to the client.</td>
</tr>
<tr>
<td>/ua/report/viewer</td>
<td>URL with the report/viewer prefix is used by the Genero Report Engine (GRE) running in local mode to load the HTML report viewer implementation. It is bound to the $(GREDIR)/viewer directory by default, see REPORT_VIEWER_DIRECTORY on page 398.</td>
</tr>
<tr>
<td>/ua/report/public</td>
<td>URL with the report/public prefix is used by the GRE to access public resources or documents shared</td>
</tr>
<tr>
<td>URI path</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>/ua/report/private/$(session.id)/reports</td>
<td>URL with the <code>report/private</code> prefix is used by the GRE to access private resources or documents locally. These resources are not shared between applications or users and are deleted when the session ends.</td>
</tr>
<tr>
<td>/ua/report-r/viewer</td>
<td>URL with the <code>report-r/viewer</code> prefix is used by the GAS running a GRE in distributed mode to load the HTML report viewer implementation. Adding a <code>REPORT_REMOTE_URL_PREFIX</code> configuration entry in the GAS <code>as.xcf</code> configuration file enables distributed mode support. See Configure GRE in distributed mode on page 158.</td>
</tr>
<tr>
<td>/ua/report-r/public</td>
<td>URL with the <code>report-r/public</code> prefix is used by GAS to access public resources or documents on remote server shared between applications, for example, report fonts.</td>
</tr>
<tr>
<td>/ua/report-r/private/$(session.id)/reports</td>
<td>URL with the <code>report-r/private</code> prefix is used by the GAS to access private resources or documents on a remote server. These resources are not shared between applications or users and are deleted when the session ends.</td>
</tr>
<tr>
<td>/private/${sessionid}</td>
<td>The GAS will send an HTTP request to the remote host to notify it that the GRE session has ended. The remote server is responsible for removing recursively private data identified by the session identifier. For example, <code>DELETE /private/ebf19aba4d09e80cd19cfeaa7868a9a2a</code></td>
</tr>
<tr>
<td>/ua/w</td>
<td>URL dedicated to the look up of the GWC-JS directory. The complete format of the URI is <code>ua/w/${GWC-JS}/${filename}</code>, where files served directly by the dispatcher for the customized GWC-JS are found. The GAS searches for the GWC-JS</td>
</tr>
</tbody>
</table>
**URI path** | **Description**
---|---
/ | directory in one of the paths specified as document root in GWC_JS_LOOKUP_PATH on page 376. GWS request to start Web service application. See Running a WSDL Web service application on page 59.
/ws/r | Request to start application using the GDC from URI. See Running a desktop application on page 58.
/da/r | Monitor requests
.monitor | Monitor logs requests
/monitor/log | Returns a list of all groups configured for applications. See Get the groups for all applications on page 59.
/monitor/configuration/application/group | Returns a list of all application names (or only the abstract applications if ?abstract is specified).
/monitor/configuration/application/name?abstract | Returns a list of all DUAs allowed for the application identified by appld
/monitor/configuration/application/dua?appId | Returns a list of all groups configured for services.
/monitor/configuration/service/group | Returns a list of all service names (or only the abstract services if ?abstract is specified).
/monitor/configuration/service/name?abstract | Returns a list of static and dynamic services that have been invalidated by the dispatcher because first request did not succeed in starting a DVM. See Get list of services invalidated by dispatcher on page 59.
/monitor/configuration/service/invalid | **Note:** A URI path is a string without line breaks. Line breaks have been added to the URIs above to support the printed version of our documentation.

**File serving URIs**
This topic explains how GAS uses the URL of a request to determine the file system location from where to serve a file.

This topic provides examples of how the GAS uses configuration elements defined in your GAS configuration file to locate a file resource specified in an application's URL path (e.g. http://localhost:6394/demos.html).

**Table 8: File serving URIs**

<table>
<thead>
<tr>
<th>URI path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>The slash (/) is the path to the document root configured in the Genero Application Server (GAS) configuration file. The DOCUMENT_ROOT element is located at /</td>
</tr>
<tr>
<td>URI path</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>/ua/i/</td>
<td>In this URL the slash (/) following the /ua/i / is the path to the public resource directory, where resources such as images are found, see Paths to application resources on page 258.</td>
</tr>
<tr>
<td></td>
<td>In this example, when accessing the photo.jpeg with the URL <a href="http://localhost:6394/ua/i/photo.jpeg">http://localhost:6394/ua/i/photo.jpeg</a>, you expect to find the photo.jpeg file in the directory specified by the $(res.public.resources), which expands to $(res.appdata.path)/public/common directory in your application server's application data path.</td>
</tr>
<tr>
<td>/ua/components</td>
<td>In this URL the slash (/) following the /ua/components/ is the path to the web component directory configured in the Genero Application Server (GAS) configuration file where web components are found. The WEB_COMPONENT_DIRECTORY element is located at /CONFIGURATION/APPLICATION_SERVER/WEB_APPLICATION_EXECUTION_COMPONENT/WEB_COMPONENT_DIRECTORY, see WEB_COMPONENT_DIRECTORY on page 422.</td>
</tr>
<tr>
<td></td>
<td>For example, this excerpt is from the default GAS configuration file (as.xcf).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|              | In this example, when accessing a web component with the URL http://localhost:6394/ua/components/mywebcomponent, you expect to find the

For example, this excerpt is from the default GAS configuration file (as.xcf).

```xml
...
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT>
    ...
    <RESOURCE Id="res.path.docroot" Source="INTERNAL">
      $(res.path.as) /web</RESOURCE>
    ...
  </PLATFORM_INDEPENDENT>
</RESOURCE_LIST>
...

<INTERFACE_TO_CONNECTOR>
  ...
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  ...
</INTERFACE_TO_CONNECTOR>
...

In this example, when calling the demo page using the URL http://localhost:6394/demos.html, you expect to find the demos.html file in the directory specified by the DOCUMENT_ROOT entry, i.e. the /web directory in your application server path.

In this example, when accessing the photo.jpeg with the URL http://localhost:6394/ua/i/photo.jpeg, you expect to find the photo.jpeg file in the directory specified by the $(res.public.resources), which expands to $(res.appdata.path)/public/common directory in your application server’s application data path.

In this example, when accessing a web component with the URL http://localhost:6394/ua/components/mywebcomponent, you expect to find the
**URI path** | **Description**
--- | ---
file in the directory specified by the `WEB_COMPONENT_DIRECTORY` entry, i.e. the `/webcomponents` directory in your application path.

### URI Examples

Several URI examples with ways to help you launch and monitor applications.

These examples are grouped under the headings connections, launching applications, Web services, and application groups.

- **Connections**
  - Direct connection to standalone GAS on page 57
  - Connection through a Web server on page 57
- **Launching applications**
  - Running a Web application on page 58
  - Running a desktop application on page 58
  - Starting applications with arguments on page 58
  - Set gwc-js customization with query string parameter on page 58
- **Web services**
  - Running a WSDL Web service application on page 59
  - Get list of services invalidated by dispatcher on page 59
- **Application groups**
  - Running an application using a group on page 59
  - Get the groups for all applications on page 59

#### Connections

**Direct connection to standalone GAS**

Calls the "myApp" Web application on the "myApplicationServer" Application Server, listening to port 6394:

```
http://myApplicationServer:6394/ua/r/myApp
```

If the application is in the default group (`_default`), you can use the same URL or you can include the group name:

```
http://myApplicationServer:6394/ua/r/_default/myApp
```

The use of the `_default` group name is optional.

**Connection through a Web server**

Calls the "myApp" application through the "myWebServer" Web Server.

```
http://myWebServer/gas/ua/r/myApp
```

---

**Launching applications**
Running a Web application

Calls the "myApp" application:

http://appserver:6394/ua/r/myApp

Running a desktop application

Using the URL with da protocol launches an application via the Genero Desktop Client monitor.

http://appserver:6394/da/r/appid

Note: Prerequisites:
1. GDC is installed
2. Application extension associations for gdc are set

Starting applications with arguments

Calls the "myApp" application with arguments, through the "myWebServer" Web Server:


Note:
1. A question mark (?) follows the application name.
2. Val1 is the value of the first argument and Val2 is the value of the second argument.
3. Each argument is separated by an ampersand (&).

Set gwc-js customization with query string parameter

To specify the customization to use for the GWC-JS user interface, add the gwc-js query string parameter to the application URL.


You can see:
• A question mark (?) follows the application name.
• gwc-js is the query string parameter.
• <my-custom-gwc-js> is either a text file containing a directory name or a directory name itself, pointing to the customization directory.

The GAS searches for the GWC-JS directory specified in the string in the paths defined by the GWC-JS-LOOKUP_PATH element.

Note: The gwc-js value provided in the query string takes precedence over the configuration for the GWC-JS element in the application configuration file (xcf).
### Running a WSDL Web service application

To get the WSDL for a specified service:

```
http://appserver:6394/ws/r/appid/service?WSDL
```

To access the Web service:

```
http://appserver:6394/ws/r/appid/service
```

If the Web service uses a group:

```
http://appserver:6394/ws/r/groupid/appid/service
```

Access through a Web server:

```
http://webserver/gas/ws/r/appid/service
```

### Get list of services invalidated by dispatcher

To retrieve a list of all services that have been invalidated by the dispatcher, enter the following URL in a browser:

```
http://myApplicationServer:6394/monitor/configuration/service/invalid
```

You see a page displaying a list of invalid configuration files (if any).

**Note:** If the list is formatted via an XSLT style sheet, the name of the `xcf` file, its group (if any), and the absolute path to the external `xcf` (if any) is displayed.

If there are no invalid services, you see a page displaying:

There are no invalid configuration files.

### Application groups

#### Running an application using a group

Calls the "myApp" application defined in group "demo" through the "myWebServer" Web Server.

```
http://myWebServer/gas/ua/r/demo/myApp
```

#### Get the groups for all applications

To retrieve the list of all groups configured for applications on an application server listening to port 6394, enter the following URL in a browser:

```
http://myApplicationServer:6394/monitor/configuration/application/group
```

An XML document, like the following, is outputted:

```xml
<?xml version='1.0' encoding='UTF-8'?><RESPONSE Request=''/monitor/configuration/application/group'>
| ENTRY Value='C:\gas-current\pkg-ebiz-appserver/app' Name='_default'/ > |
| ENTRY Value='C:\gas-current\pkg-ebiz-appserver/demo/app' Name='demo'/ > |
| ENTRY Value='C:\gas-current\common\repository' Name='common'/ > |
| ENTRY Value='C:\gas-current\tests' Name='qa-test'/ > |
</RESPONSE>
Configuring the Genero Application Server

Configure the Genero Application Server to work with your Web server.

- System Requirements on page 61
- GAS Configuration Check on page 62
- Licensing on page 65
- ISAPI Extension Installation and Web Server Configuration on page 70
- FastCGI Installation and Web Server Configuration on page 89
- Java Servlet Installation and Web Server Configuration on page 96
- Validating configuration files on page 98
- Configuring applications on GAS on page 100
- How to implement delegation on page 113
- How to implement Single sign-on (SSO) on page 122
- Compression in Genero Application Server on page 154
- Configure development environment on page 156
- Configure multiple dispatchers on page 156

System Requirements

Failure to meet system requirements can result in unexpected behavior.

- Operating systems on page 61
- Web servers on page 61
- User agents on page 61
- Databases on page 62

Operating systems

Linux®IBM® AIX®, HP-UX®, SUN Solaris and Microsoft® Windows®.

For a detailed list of supported operating systems, refer to the system support matrix available on the Four Js web site documentation or download area, or contact your support center.

Web servers

Web server support depends on the platform hosting the Genero Application Server.

On Windows® platforms, typically Internet Information Services is used. For more information on prerequisites and configuration see ISAPI installation.

On Linux®/UNIX™ platforms, we recommend Apache httpd, lighttpd and nginx. However, any Web server that is compatible with fastcgi can be used. For more information, refer to http://httpd.apache.org. To install and configure the required components, see FastCGI installation.

GAS has a dedicated dispatcher for Java™ Application Servers. For more information, see Java™ Servlet installation.

User agents

A User Agent is a client agent. It can be a Web browser, the Genero Desktop Client, or a Web Services client.

For the Genero Web Client for JavaScript, the supported user agents are agents supporting HTML5.
We support the latest versions of the supported browsers. Users must update their browsers to the latest version.

**Important:** For the most current information about supported browsers, refer to the *Supported platforms and databases* document, available on the Products download page of the Four Js website.

Supported browsers are:

- Microsoft® Internet Explorer
- Microsoft® Edge
- Google Chrome™
- Safari® on MAC® OS
- Mozilla® Firefox®
- Native mobile browsers are supported on iOS (Mobile Safari) and Android™ (Mobile Chrome).

**Databases**

Any database accessible from the DVM or from the ODI is supported.

For a detailed list of supported databases, refer to the system support matrix available on the Four Js website documentation or download area, or contact your support center.

**GAS Configuration Check**

It is recommended that you verify your install and configuration of Genero Application Server prior to working with your own applications.

- Managing Access Rights on page 62
- Validating the Installation with the Genero Web Client on page 62
- Validating the Installation with the GDC on page 63
- Troubleshooting Configuration Issues on page 64

**Managing Access Rights**

When the Genero Application Server starts an application, the *fglrun* process executes in the context of the operating system user running the Genero Application Server. Generally, this is the user running IIS when using the ISAPI connector or the user running Apache when using the FastCGI connector.

You must ensure that the user has the permissions needed to:

- Execute the DVM *fglrun* program
- Access any needed DVM resources (ODI add-ons, configuration files, and so on)
- Access any needed program files and resources (*42r* files, *42m* files, *42f* files, and so on)

**Validating the Installation with the Genero Web Client**

For the Genero Web Client, you can validate with and without a Web server.

**Important:** After you upgrade the Genero Application Server, you must clear the CSS and JavaScript downloaded to the browser cache by clearing the browser cache. For many browsers, you can accomplish this by pressing CTRL + F5.

- Validate the installation for GWC without a Web server on page 62
- Validate the integration for GWC with a Web server on page 63

**Validate the installation for GWC without a Web server**

To validate the installation of the Genero Application Server without involving a Web server, launch the stand-alone Genero Application Server and run a Genero Web Client demo application.

**Before you begin:**
Due to configuration settings in the Genero Application Server configuration file, access to demo applications needs to be configured. Make sure you have provided access to run the demo applications from the localhost, see Example configuring access control for demo applications on page 352.

1. Set the GAS environment by executing the script $FGLASDIR/envas.

2. Launch the GAS Standalone: `httpdispatch`
   For more information on starting the GAS standalone, and the various command options available, see Tools and Commands on page 304.

3. From your web browser, check the connection to the application server by displaying the Demos page, using a URI that provides a direct connection to the standalone GAS:

   ![Image](http://localhost:6394/demos.html)

4. Launch the GWC Demos program using a URI that provides a direct connection to the standalone GAS:

   ![Image](http://localhost:6394/ua/r/gwc-demo)

The Demos application provided with the installation files, is preconfigured and ready to run.

**Important:** When you install the GAS using the .msi setup program, it sets the ADDRESS element (child of INTERFACE_TO_DVM) to “127.0.0.1” within the as.xcf file. When validating your installation, if the Demos application fails to display (and you receive a runtime error), you may have to replace the INTERFACE_TO_DVM's ADDRESS element with the true IP address of the host machine.

**Validate the integration for GWC with a Web server**
To validate the installation of the Genero Application Server using a Web server, specify the Web server as part of the application URI and run a Genero Web Client demo application.

**Before you begin:**
Due to configuration settings in the Genero Application Server configuration file, access to demo applications needs to be configured. Make sure you have provided access to run the demo applications from the localhost, see Example configuring access control for demo applications on page 352.

**Important:** The following instructions assume that “gas” is the virtual directory defined for the GAS (to be part of the URLs accessing the GAS). For information on defining a virtual directory, see either ISAPI installation or FastCGI installation.

1. Before you begin, you should have first validated the installation as a standalone GAS. See Validate the installation for GWC without a Web server on page 62.

2. From your web browser, ensure that your Web server is correctly configured by accessing a static page (such as index.html), or simply `http://localhost`.

3. Display the Demos page:

   ![Image](http://myWebServer/gas/demos.html)

4. Launch the GWC Demos program.

   ![Image](http://myWebServer/gas/ua/r/gwc-demo)

**Validating the Installation with the GDC**
The Genero Application Server can deliver clients to the Genero Desktop Client.

The Genero Desktop Client (GDC) must be installed prior to starting this validation procedure. It can be installed on the same host, or it can be installed on a separate client machine. For instructions on installing the GDC, refer to the Genero Desktop Client User Guide.

- Validate the installation for GDC without a Web server on page 64
Validate the installation for GDC without a Web server

To validate the installation of the Genero Application Server without involving a Web server, launch the stand-alone Genero Application Server and run a Genero Desktop Client demo application.

Before you begin:

Due to configuration settings in the Genero Application Server configuration file, access to demo applications needs to be configured. Make sure you have provided access to run the demo applications from the localhost, see Example configuring access control for demo applications on page 352.

1. Set the GAS environment by executing the script $FGLASDIR/envas.
2. Launch the GAS Standalone: httpdispatch
   For more information on starting the GAS standalone, and the various command options available, see Startup and Command Options.
3. Within the GDC, create a shortcut pointing to the demo application.
   To create the shortcut, you must start the GDC in administrative mode using the --admin or -a option. Refer to the Genero Desktop Client User Guide for more information on creating shortcuts.
   a) On the first page of the New Shortcut wizard, select the HTTP, through a web server option.
   b) On the second page of the New Shortcut wizard - the HTTP connection information page - provide the application URL (http://myApplicationServer:6394/ua/r/gwc-demo) and specify the HTTP Proxy mode as Direct connection.
      On most systems, you can replace the myApplicationServer with localhost for this test: http://localhost:6394/ua/r/gwc-demo.
   c) For the remaining pages of the Shortcut Wizard, you can accept the defaults and click Finish. The shortcut is added to the GDC.
4. To run the application, select the shortcut and click Start!
   **Important:** When you install the GAS using the .msi setup program, it sets the ADDRESS element (child of INTERFACE_TO_DVM) to "127.0.0.1" within the as.xcf file. When validating your installation, if the Demos application fails to display (and you receive a runtime error), you may have to replace the INTERFACE_TO_DVM's ADDRESS element with the true IP address for the host machine.

Troubleshooting Configuration Issues

Troubleshooting tips are provided for the most common issues encountered.

- **Proxy errors on Windows platform** on page 64
- **Cannot find 127.0.0.1 or localhost on Windows** on page 65
- **What if the application doesn't start?** on page 65

**Proxy errors on Windows® platform**

On Windows®, if you find the error "Proxy refuses to start with socket error code 10038" in the proxy log, it means that the proxy refused to start and returned a socket error code of 10038. This can occur due to issues with the drivers provided by some third-party layered service providers (LSPs).

To rectify this situation, you need to run the following from the command line:

```
netsh winsock reset catalog
```

The command resets the Winsock Catalog to a clean state. Be aware that it might affect your installed applications that use the internet. You might need to reconfigure or reinstall such applications, so use the command cautiously. The command will ask to restart Windows™.
Cannot find 127.0.0.1 or localhost on Windows
For users on Windows® 64-bit machines who are using a network proxy, the browser cannot open 127.0.0.1 or localhost unless you modify your Advanced Network settings to avoid going through the proxy for these addresses.

What if the application doesn't start?
If an application does not start, you can debug the problem by manually launching the program from a command shell.

The first thing to check is the configuration information - to ensure that all components are set properly.

1. The Genero Application Server creates separate log files for its dispatchers, proxies, and the DVMs started by those proxies. Examine the logs as they may provide you with some helpful information or error messages. For more information about accessing log files see Logging on page 167.

2. Check your environment variables in $FGLASDIR/etc/as.xcf.

   **Tip:** You can get messages for the environment in the GAS log by setting the CATEGORIES_FILTER on page 361 category filter to CONFIGURATION.

3. You may need to run the application in debug mode using the FGL debugger.

   To run the FGL debugger, the dispatcher must open a DOS command or an xterm window so that you can run the application with the fglrun -d command. For example, on Windows® platform, start the dispatcher with the command to open a DOS window and override some of the settings for res.dvm.wa:

```
httpdispatch -E res.dvm.wa="cmd /K start cmd"
```

Before the application displays in the Web browser, a command window will open with all environment settings for that application. You can then manually run your application in debug mode, for example with fglrun -d progname to enter the command-line debugger (fgldb). The application will then display in the Web browser. See Using the debugger on page 168.

**Note:**
- You can use the graphical debugger in Genero Studio. For more information, see the Genero Studio User Guide.
- The debug facility of the Genero Desktop Client includes logging and the debug console. For more information on using the GDC debug facility, see the Genero Desktop Client User Guide.
- For details about debugging Genero Web Client for JavaScript (GWC-JS) applications, see Configure development environment on page 156.

When you receive the Error: Runtime error. Try again ... page
Simply put, your application cannot start and you must check your application configuration. This error is typically the result of an incorrect path to the program executable.

**Licensing**
The documentation provides an explanation of how licensing works through the use of diagrams. It does not replace the license agreement.

- Licensing - Base Example on page 66
- Licensing - Using the RUN command on page 66
- Licensing - Multiple User Agents on page 67
- Licensing - Summary Case on page 67
- Genero front-ends and License Counting on page 68
- Licensing Tips and Tricks / Troubleshooting on page 69
**Important:** This topic discusses Genero licensing. Some of the demos accessed via the demos.html page use third-party components which are NOT delivered, licensed or supported by your Genero provider. They are just bundled for the purpose of demonstrating the various capabilities of some of the GWC modes. Before using a demo code snippet for your own development, take care that you fulfill the corresponding licenses from all third-party components.

**Licensing - Base Example**
This scenario shows two User Agents connected to two DVMs.

![Diagram of PC #1 and PC #2 connected to Web Server and Application Server]

The connection is made via the Web server, the ISAPI or FastCGI extension, and dispatchers.

In this scenario, two (2) runtime licenses are used.

**Note:** Most browsers now support tabs. It is important to understand that for this discussion, each browser is assumed to be using only one tab. If you open two tabs in a browser, and each tab connects to its own DVM, then it is just as if two browsers were being used, and two (2) runtime licenses are used.

**Licensing - Using the RUN command**
This scenario shows two User Agents connected to an application, which in turn calls other applications using the Genero BDL **RUN** command or the **RUN WITHOUT WAITING** statement.

![Diagram of PC #1 and PC #2 connected to Web Server and Application Server with runners]
In this scenario, two (2) runtime licenses are used.

**Licensing - Multiple User Agents**

This scenario shows four User Agents running on two different PCs and connected to four DVMs.

In this scenario, four (4) runtime licenses are used.

**Licensing - Summary Case**

This scenario shows four User Agents running on two different PCs and connected to four DVMs, some of which are running external DVMs using the Genero BDL `RUN` command.
In this scenario, four (4) runtime licenses are used.

**Genero front-ends and License Counting**

A Genero front-end (GWC, GDC-HTTP) does not require any additional license information.

When a user requests an application, the dispatcher starts a DVM to handle the request. It is the DVM that consumes a license. For example, one license is used when an application is started from a User Agent. If within this application, a RUN or a RUN WITHOUT WAITING is executed, the same license is used, even if the first User Agent opens new User Agents.

If, however, an application is started in another User Agent (without RUN or RUN WITHOUT WAITING), a new license is used.

For GWC applications, one license is consumed per application started (note as stated above no extra licence for any RUN). Thus, size the license number requested to run your applications. Another solution is to use CPU licenses.

For GDC applications, one license is consumed per monitor/console, no matter how many applications are run from this monitor.

When the license is freed depends on how the application is exited. A license is freed when the applications closes, or to be more exact, when the DVM is shut down. If the user exits the application by clicking on the cancel or exit button, the DVM is shut down and the license is immediately freed. If, however, the user does not exit the application but instead closes the User Agent, the DVM continues to run until the application times out (the number of seconds is set for the USER_AGENT timeout). After the timeout period passes, the proxy closes the connection to the DVM, the DVM shuts down, and the license is freed.

To determine the number of licenses used, run "fglWrt -u" followed by "fglWrt -a info users" on the application server where the Genero runtime is installed.
Important: When you do a refresh on a GWC application, if the current page is the first page of the application this will not refresh the application page but start a new application. Thus, consume an extra license.

Licensing Tips and Tricks / Troubleshooting
Identify GAS configuration elements that can impact licensing and identify additional licensing considerations.

- The ADDRESS element on page 69
- The USER_AGENT timeout element on page 69
- The MAX_AVAILABLE element for a Web Service on page 69
- Evaluate licensing when migrating from GDC on page 69

The ADDRESS element
In the Genero Application Server configuration file, within the INTERFACE_TO_DVM/ADDRESS element, it is not recommended to use either localhost or 127.0.0.1, as the license server requires the real address of the machine to check the licenses.

If your machine is not well configured, a bad address is returned to the license server, which will then refuse to start a new DVM.

The USER_AGENT timeout element
With the Genero Web Client, the USER_AGENT timeout element proves to be particularly useful.

As with the other front-ends, when a user properly exits an application, the DVM handling that application is properly shut down and the license that the application consumed is released back into the Genero license application pool.

However, when the user does not properly exit the application, the DVM remains alive and continues to consume a license even though the front-end has stopped. This can occur with the Genero Web Client, when a user closes the browser instead of properly exiting the application; the front-end has no mechanism to tell the Genero Application Server that the user has closed his browser.

To bypass this limitation, you can define the USER_AGENT timeout parameter to count user inactivity. When this timeout occurs, Genero Application Server unilaterally closes the socket to the DVM, which causes the DVM to shut down and the license to be released.

The MAX_AVAILABLE element for a Web Service
The MAX_AVAILABLE element explicitly limits the number of DVMs that can be started for a specific Web service.

No more DVMs are started beyond this limit, once MAX_AVAILABLE DVMs have been reached. You can use this parameter to assist in limiting the number of licenses consumed.

Evaluate licensing when migrating from GDC
Evaluate licensing when migrating from the Genero Desktop Client (GDC).

For Genero Web Client (GWC) applications, we recommend CPU licenses. GWC applications cannot detect if the user has left the application, if the user has simply closed the browser (as opposed to stopping the application by explicitly exiting the application by selecting the appropriate action). If the user closed the browser without exiting the application, the application is still running and continues to consume a license. A timeout can be configured to release the DVM, which in turn releases the license. To minimize the impact, a solution is to use an application launcher: a main application that executes RUNS of sub-applications or a startmenu.
ISAPI Extension Installation and Web Server Configuration

Follow the procedures to configure a Microsoft® Internet Information Services (IIS) Web server with ISAPI extensions for **isapidispatch**.

- The Genero Application Server and IIS on page 70
- Install the ISAPI dispatcher on page 70
- GAS ISAPI Extension configuration file on page 88
- Troubleshooting installation on page 88
- Restarting the ISAPI dispatcher on page 89

The Genero Application Server and IIS

The Internet Information Services (IIS) is configured with the **isapidispatch** and a virtual directory is created for the GAS on the IIS's "Default Web Site". This configuration is shown for different versions of IIS.

The Genero Application Server (GAS) has a special ISAPI extension DLL (**isapidispatch.dll**) that functions as a dispatcher. It is loaded directly by an IIS worker process. The IIS sees this as a script engine or application enabled to handle requests to a virtual directory bound to the GAS.

As the GAS is therefore embedded within the Microsoft® IIS Web server this improves performance and allows administration of the server with IIS tools.

**Note:** The GAS must already be installed in a directory reachable by IIS. The GAS installation directory is referenced as FGLASDIR. See the appropriate installation guide for your installation for more information.

For further details about IIS, see the IIS documentation for your installation. For example, see **IIS 6.0 Operations Guide** or **Learn IIS**.

Install the ISAPI dispatcher

From the procedures to configure the Microsoft® Internet Information Services (IIS) and install the ISAPI dispatcher within its framework, select the configuration that is right for your system's version of IIS.

**Note:** For older version of IIS you install it using the Windows® installer, see **Installing with the Microsoft Installer** on page 70. For latest version, IIS is a built-in component of the operating system, you just need to activate it on your system. If you plan to use IIS, install or activate it before running the GAS setup program so that the installation process can complete the Web Server configuration. To check the installation, or do any additional configuration, select the configuration topic that is right for your system's version of IIS.

- Installing with the Microsoft Installer on page 70
- Manual configuration for IIS 6.0 on page 73
- Manual configuration for IIS 7.x on page 74
- Manual configuration for IIS 8.x and IIS 10.x on page 80
- Finishing the installation on page 87

Installing with the Microsoft® Installer

When you launch the Microsoft® Installer (MSI) for your product, it automatically attempts to install the ISAPI dispatcher within the IIS framework.

Download the appropriate version of the software for your operating system. For Windows®, this means downloading either the 32-bit or 64-bit version. The file is downloaded as an executable (.exe) file.

Once the file is downloaded, click on the file to execute the installation wizard. You will follow the instructions provided by the wizard, to include selecting the install directory, providing the Start Menu Folder name, and specifying the appropriate FGLDIR to use for this GAS installation.
At this point in the wizard, you are asked whether you want to install the ISAPI Extension. If you select NOT to install the ISAPI Extension, then you will either only have a standalone GAS or you will have to install the ISAPI Extension manually at a later time. By default, it will attempt to install the ISAPI Extension.

**Figure 12: Wizard, Product Features page**

If the ISAPI Extension is selected, you must complete three wizard pages. On the first wizard page, you select the Web Site where you want to add your Web application.

**Figure 13: Wizard, ISAPI (1 of 3) - Web Site page**

On the second wizard page, you provide the name for the Web Application. This is the name that will be included as the path to the GAS in the URL for your applications.
On the third wizard page, you specify the web application path. It is within this directory that the isapidispatch.ini file will be created.

At this point, and Install button appears, and the installation takes place.

Once the install is complete, the GAS ISAPI Extension should be ready to use. Open a web browser and enter the URL to the “demos.html” page. The URL should look like:

http://<server>:<port>/<virtual directory>/demos.html

For example, if the server is "localhost", the port is the default port, and the virtual directory is "gas", then the URL should be:

http://localhost/gas/demos.html

If you have an installation failure, or if you opted not to include the ISAPI Extension, the next sections help you to manually configure the ISAPI Extension.
Manual configuration for IIS 6.0
These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 6.0.

Configuring IIS 6.0 application pools
The application pool should be configured according to the kind of application the GAS ISAPI Extension will run. If the GAS ISAPI Extension runs only Web service applications, it is fully compatible with application pool parameters. If it runs Web or Desktop applications, IIS should drive all requests to the same worker process; therefore, a dedicated application pool should be created.

To Create an Application Pool
1. In IIS manager, right-click Application Pools, and then click New and Application Pool...
2. In the Add New Application Pool dialog, enter a name for the application pool ID, for example "GASAppPool".
3. Click OK.

To configure the Application Pool to run Web and Desktop Applications
1. In IIS manager, right-click the application pool, and then click Properties.
2. Click the Recycling tab.
3. Ensure that all checkboxes are unchecked.
4. Click the Performance tab.
5. In the Idle timeout section, ensure that the Shutdown worker processes after being idle for checkbox is unchecked, or that its value is greater than the USER_AGENT timeout value of 4GL applications that the application pool will run.
6. In the Web garden section, ensure that the value of Maximum number of worker process is 1.
7. Click OK.

The GAS ISAPI Extension will be executed on behalf of the user that is registered in the pool's properties. That user must have access to the directory specified by the FGLASDIR environment variable. To change the user identity that runs the GAS:
1. In IIS manager, right-click the application pool, and then click Properties.
2. Click the Identity tab.
3. According to your security policy, either select a built-in account or set a custom account.
4. Click OK.

Note: In order to configure application pools according to the type of applications that will be run, two different virtual directories, each with its own application pool, may be created on the same instance of IIS - one that will run only Web service applications, and another that will run only Web and Desktop applications.

Configuring an IIS 6.0 application
To create a virtual directory:
1. Create a directory on the disk that will be the application root for the application. For further details about IIS-related vocabulary, see Microsoft's documentation for IIS 6.0.
2. In IIS Manager, right-click the web site on which you want to add the virtual directory, for example "Default Web Site", and then click New and Virtual Directory...
3. Click Next on the first sheet of the Virtual Directory Creation wizard.
4. Enter the alias of the virtual directory, for example "gas". This name will be the virtual directory part of the URLs accessing the GAS. Click Next.
5. Enter the path to the directory created in step 1. Click Next.
6. On the Virtual Directory Access Permissions sheet, uncheck the Read checkbox and check Run scripts (such as ASP). Click Next.
7. Click **Finish**.

On IIS 6.0, to bind the virtual directory to the GAS ISAPI extension:

1. In IIS Manager, right-click the virtual directory previously created, and then click **Properties**.
2. Click the **Virtual Directory** tab.
3. In the **Application settings** area, click **Configuration...**, and then click the **Mappings** tab.
4. Optionally, in the **Mappings tab**, remove all predefined application extensions.
5. In the **Wildcard application maps** area, click **Insert...**
6. Enter the path to the GAS ISAPI Extension DLL: `FGLASDIR\bin\isapidispatch.dll`.
7. Uncheck the **Verify that file exists** checkbox.
8. Click **Ok**.
9. In the **Application Configuration** dialog, click **Ok**.
10. In the **Properties** dialog, click **Ok**.

On IIS 6.0, to allow the GAS ISAPI Extension to run:

1. In IIS Manager, click **Web Service Extensions**.
2. Click **Add a new Web service extension...**
3. Enter an extension name, for example "Genero Application Server".
4. Click **Add...**
5. Enter the path to the GAS ISAPI Extension DLL: `FGLASDIR\bin\isapidispatch.dll`.
6. Click **Ok**.
7. Check the **Set extension status to Allowed** checkbox.
8. Click **Ok**.

**Post requisites**

After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see **Finishing the installation** on page 87.

**Manual configuration for IIS 7.x**

These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 7.0.

**Prerequisites**

The installer needs some IIS features to be activated:

- IIS Management Scripts and Tools
- ASP.NET
- .NET Extensibility
- ISAPI Extensions
- ISAPI Filters

**To activate "IIS Management Scripts and Tools" on a Windows® Server 2008:**

- Click on Start -> Control Panel -> Administrative Tools -> Server Manager -> Roles -> Web Server (IIS)
- Right Click -> Add Role Services
- Select "Management Tools"
- Check "IIS Management Scripts and Tools"
- Click the "Install" Button

Do the same for the other features.

Example:

**Figure 16: Role services dialog**
To activate IIS Management Scripts and Tools on Windows® 7

To activate IIS Management Scripts and Tools on Windows® 7 and validate the basic IIS configuration requirements:

1. Verify IIS is installed. In Control Panel > Administrative Tools, you must have an entry called Internet Information Services (IIS) Manager.

2. Verify IIS is well-started. Open a browser and enter the URL: http://localhost. The IIS welcome screen displays.
3. Set the appropriate Windows® Features. Go to Control Panel >> Programs and Features. Click Turn Windows® features on or off.
4. In the *Windows® Features* dialog, select *IIS Management Scripts and Tools*, and verify that the ASP and the ISAPI options are also selected. When checking these options, other options may also be automatically checked. That is normal behavior, as they are combined options.
Configuring the Genero Application Server

Configuring IIS 7.x application pools

The application pool should be configured according to the kind of application the GAS ISAPI Extension will run. If the GAS ISAPI Extension runs only Web service applications, it is fully compatible with application pool parameters. If it runs Web or Desktop applications, IIS should drive all requests to the same worker process; therefore, a dedicated application pool should be created.

To create an application pool:

1. In IIS manager, right-click Application Pools and click Add Application Pool....
2. In the Add Application Pool dialog, enter a name for the application pool, for example "GASAppPool".
3. In the .NET Framework version box, select No Managed Code.
4. In the Managed pipeline mode box, select Classic.
5. Click OK.

To configure the application pool to run Web and Desktop applications:

1. In IIS Manager, right-click the application pool, and click Advanced Settings...
2. In the Advanced Settings dialog, in the Process Model area, set the Idle Time-out (minutes) field to "0" or to a value that is greater than the USER_AGENT timeout value of 4GL applications that the application pool will run.
3. In the Process Model area, set the Maximum Worker Processes field to 1.
4. In the Recycling area, set the Disable Overlapped Recycle flag to True.
5. In the Recycling area, set the Disable Recycling for Configuration Changes flag to True.
6. Click OK.
Configuring the Genero Application Server

The GAS ISAPI Extension will be executed on behalf of the user that is registered in the pool's properties. That user must have access to the FGLASDIR directory. To change the user identity that runs the GAS:

1. In IIS Manager, right-click the application pool, and click Advanced Settings...
2. In the Advanced Settings dialog, in the Process Model area, click the Identity field.
3. Click ... to open the Application Pool Identity dialog.
4. According to your security policy, either select a built-in account or set a custom account.
5. Click OK.
6. In the Advanced Settings dialog, click OK.

Note: In order to configure application pools according to the type of applications that will be run, two different virtual directories, each with its own application pool, may be created on the same instance of IIS - one that will run only Web service applications, and another that will run only Web and Desktop applications.

Configuring an IIS 7.x application

To create an application:

1. Create a directory on the disk that will be the application root for the application, the directory is for example FGLASDIR/ISAPI.
2. In IIS Manager, right-click the web site on which you want to add the application, for example "Default Web Site", and then click Add Application...
3. In the Add Application dialog, enter the alias of the application, for example "gas". This name will be the virtual directory part of the URLs accessing the GAS.
4. Enter the physical path to the directory created in step 1.
5. Click Select...
6. In the Select Application Pool dialog, select the application pool that has been defined previously.
7. Click OK.
8. In the Add Application dialog, click OK.

The authentication configuration depends on your security policy. If all users have access to the application, the identity of the anonymous user should be configured as following:

1. In IIS Manager, click the application on which you want to configure the identity of the anonymous user.
2. In the Features View panel, double-click the Authentication icon.
3. In the Authentication feature, select the Anonymous Authentication line.
4. Ensure that the status is Enabled.
5. In the Actions area, click Edit...
6. In the Edit Anonymous Authentication Credentials dialog, select Application pool identity.
7. Click OK.

To bind the application to the GAS ISAPI Extension:

1. In IIS Manager, click the application on which you want to bind the application to the GAS ISAPI Extension.
2. In the Features View panel, double-click the Handler Mappings icon.
3. In the Handler Mappings feature, in the Actions area, click Add Wildcard Script Map...
4. In the Add Wildcard Script Map dialog, enter the path to the GAS ISAPI Extension DLL: FGLASDIR \bin\isapidispatch.dll.
5. Enter a name for this mapping, for example "GAS ISAPI Extension".
6. Click OK.
7. To the question Do you want to allow this ISAPI extension?, click Yes.
8. In the Handler Mappings feature, in the Actions area, click View Ordered List...
9. Ensure that the GAS ISAPI Extension is at the top of the list.
10. Click View Unordered List...
11. In the Actions area, click Edit Feature Permissions...
12. In the Edit Feature Permissions, ensure that Script is selected and all other are unselected.
13. Click OK.

Although the GAS ISAPI Extension has been allowed automatically when you answered Do you want to allow this ISAPI extension? with Yes, to do it manually:

1. In IIS Manager, click the root node, the one that contains the host name.
2. In the Features View panel, double-click the ISAPI and CGI Restrictions icon.
3. In the ISAPI and CGI Restrictions feature, in the Actions area, click Add...
4. In the Add ISAPI or CGI Restriction dialog, enter the path to the GAS ISAPI Extension DLL: FGLASDIR\bin\isapidispatch.dll.
5. Enter a description, for example “GAS ISAPI Extension”.
6. Ensure that the Allow extension path to execute checkbox is checked.
7. Click OK.

Post requisites
After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see Finishing the installation on page 87.

Manual configuration for IIS 8.x and IIS 10.x
These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 8.0 and IIS 10.0.

Prerequisites
The installer needs some IIS features to be activated:

• IIS Management Scripts and Tools
• ASP.NET
• .NET Extensibility
• ISAPI Extensions
• ISAPI Filters

To activate IIS on a Windows® Server 2012:

• Open Server Manager
• Under the Manage menu, select Add Roles and Features to launch the Add Roles and Features Wizard
• For Installation type, check the option Role-based or Feature-based Installation and click Next
• For Server Selection, select the appropriate server, (the local server is selected by default) and click Next
• For Server Roles, check Web Server (IIS) and click Next
• For Features, accept the default settings and click Next
• For Web Server Role (IIS), accept defaults and click Next
• For Role Services, accept the default settings that have already been selected for you, and verify that the IIS Management Scripts and Tools, IIS Management Console, ASP.NET, .NET Extensibility, ISAPI Extensions, and ISAPI Filters options are selected, and then click Next
• For Confirmation, click the Install button. (When the IIS installation completes, the wizard shows the installation status in the Results screen)
• Click Close to exit the wizard

To activate IIS Management Scripts and Tools on Windows® 8 and 10
To activate IIS Management Scripts and Tools on Windows® 8 and 10 and validate the basic IIS configuration requirements:
1. Verify IIS is installed. In Control Panel > Administrative Tools, you must have an entry called Internet Information Services (IIS) Manager.

![Figure 21: Administrative Tools window with IIS Manager highlighted](image)

2. Verify IIS is started. Open a browser and enter the URL: http://localhost. The IIS welcome screen displays.

![Figure 22: IIS Welcome Screen](image)

3. Set the appropriate Windows® Features. Go to Control Panel > Programs and Features. Click Turn Windows features on or off.
4. In the **Windows Features** dialog, select **IIS Management Scripts and Tools**, and verify that the ASP and the ISAPI options are also selected. When checking these options, other options may also be automatically checked. That is normal behavior, as they are combined options.
Configuring IIS 8.x and IIS 10.x application pools

The application pool should be configured according to the kind of application the GAS ISAPI Extension will run. If the GAS ISAPI Extension runs only Web service applications, it is fully compatible with application pool parameters. If it runs Web or Desktop applications, IIS should drive all requests to the same worker process; therefore, a dedicated application pool should be created.

To create an application pool:

1. In Internet Information Services (IIS) Manager, right-click Application Pools and click Add Application Pool....
2. In the Add Application Pool... dialog, enter a name for the application pool, for example “GASAppPool”.

3. In the .NET CLR version box, select No Managed Code.

4. In the Managed pipeline mode box, select Classic.

5. Click OK.

To configure the application pool to run Web and Desktop applications, perform the following:

1. In Internet Information Services (IIS) Manager, right-click the application pool (GASAppPool in our example), and click Advanced Settings...

2. In the Advanced Settings dialog, in the Process Model area, set the Idle Time-out (minutes) field to “0” or to a value that is greater than the USER_AGENT timeout value of Genero 4GL applications that the application pool will run.

3. In the Process Model area, set the Maximum Worker Processes field to 1.

4. In the Recycling area, set the Disable Overlapped Recycle flag to True.

5. In the Recycling area, set the Disable Recycling for Configuration Changes flag to True.

6. Click OK.

The GAS ISAPI Extension will be executed on behalf of the user that is registered in the pool's properties. That user must have access to the FGLASDIR directory. To change the user identity that runs the GAS:

1. In Internet Information Services (IIS) Manager, right-click the application pool (GASAppPool in our example), and click Advanced Settings...

2. In the Advanced Settings dialog, in the Process Model area, click the Identity field.

3. Click ... to open the Application Pool Identity dialog.

4. According to your security policy, either select a built-in account or set a custom account.

5. Click OK.

6. In the Advanced Settings dialog, click OK.

Note: In order to configure application pools according to the type of applications that will be run, two different virtual directories, each with its own application pool, may be created on the same instance of IIS - one that will run only Web service applications, and another that will run only Web and Desktop applications.
Configuring an IIS 8.x and IIS 10.x application

To create an application:

1. Create a directory on the disk that will be the application root for the application, the directory is for example `FGLASDIR/ISAPI`.
2. In **Internet Information Services (IIS) Manager**, right-click the web site on which you want to add the application, for example “Default Web Site”, and then click Add Application...

![Internet Information Services (IIS) Manager Default Web Site Home screen](image)

3. In the **Add Application** dialog, enter the alias of the application, for example “gas”. This name will be the virtual directory part of the URLs accessing the GAS.
4. Enter the physical path to the directory created in step 1.
5. Click Select...
6. In the **Select Application Pool** dialog, select the application pool that has been defined previously.
7. Click OK.
8. In the **Add Application** dialog, click OK.

The authentication configuration depends on your security policy. If all users have access to the application, the identity of the anonymous user should be configured as following:

1. In **Internet Information Services (IIS) Manager**, click the application on which you want to configure the identity of the anonymous user.
2. In the **Features View** panel, double-click the **Authentication** icon.
3. In the **Authentication** feature, select the Anonymous Authentication line.
4. Ensure that the status is Enabled.
5. In the **Actions** area, click **Edit**.
6. In the **Edit Anonymous Authentication Credentials** dialog, select Application pool identity.
7. Click **OK**.

To bind the application to the GAS ISAPI Extension:

1. In **Internet Information Services (IIS) Manager**, click the application on which you want to bind the application to the GAS ISAPI Extension.
2. In the **Features View** panel, double-click the **Handler Mappings** icon.
3. In the **Handler Mappings** feature, in the **Actions** area, click **Add Wildcard Script Map**.
4. In the **Add Wildcard Script Map** dialog, enter the path to the GAS ISAPI Extension DLL: `FGLASDIR\bin\isapidispatch.dll`.
5. Enter a name for this mapping, for example "GAS ISAPI Extension".
6. Click **OK**.
7. To the question **Do you want to allow this ISAPI extension?**, click Yes.
8. In the **Handler Mappings** feature, in the **Actions** area, click **View Ordered List**.
9. Ensure that the GAS ISAPI Extension is at the top of the list.
10. Click **View Unordered List**.
11. Select the GAS ISAPI Extension in the **Actions** area, click **Edit**.
12. In the **Edit Script Map** dialog, click on **Request Restrictions**.
13. In the **Request Restrictions** dialog's **Mapping** panel, uncheck the **Invoke handler only if requests is mapped to**: option.
14. In the **Request Restrictions** dialog's **Access** panel, select the **Script** option.
15. Click **OK**.

Although the GAS ISAPI Extension has been allowed automatically when you answered **Do you want to allow this ISAPI extension?** with Yes, to do it manually:

1. In **Internet Information Services (IIS) Manager**, click the root node, the one that contains the host name.
2. In the **Features View** panel, double-click the **ISAPI and CGI Restrictions** icon.

![Figure 27: Internet Information Services (IIS) Manager Connections root node screen](image-url)
3. In the ISAPI and CGI Restrictions feature, in the Actions area, click Add...
4. In the Add ISAPI or CGI Restriction dialog, enter the path to the GAS ISAPI Extension DLL: FGLASDIR\bin\isapidispatch.dll.
5. Enter a description, for example “GAS ISAPI Extension”.
6. Ensure that the Allow extension path to execute checkbox is checked.
7. Click OK.

Post requisites
After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see Finishing the installation on page 87.

Finishing the installation
After you have finished ISAPI installation, you need to create a configuration file and verify that your installation and basic configuration was successful.

Prerequisites
It is assumed you have your ISAPI already installed and activated, and that you have created a directory on your disk which will be the application root (i.e. the virtual directory):

To finish the installation
1. Create a file called isapidispatch.ini in the application root directory; a sample file can be found in the FGLASDIR\etc directory.
   • The options section of the file must contain at least the as-directory property.
   • The value of the as-directory must be set to the FGLASDIR environment variable.

   See GAS ISAPI Extension configuration file on page 88 for details about the content of this file.
2. The GAS ISAPI Extension should be ready to use. Open a web browser and enter the URL to the "demos.html" page:

   http://<server>:<port>/<virtual directory>/demos.html

   For example, if the server is "localhost", the port is the default port, and the virtual directory is "gas":

   http://localhost/gas/demos.html

   If everything is correct, the "demos" page should be displayed; otherwise, see Troubleshooting installation.

   Note:
   1. All requests having an URL that begins with the virtual directory name, for example "/gas/", will be served by the GAS ISAPI Extension. Files present in the application root directory will not be served by IIS. There is no way to configure IIS to change this behavior. Therefore, the isapidispatch.ini file should be the only file present in the application root directory.
   2. While IIS allows you to map more than one wildcard application to a virtual directory, the GAS ISAPI Extension should be the last one of the list, as it will not forward URLs to other ones.
   3. As in the standalone GAS, the GAS ISAPI Extension serves application resource files and static files, such as demos.html.
GAS ISAPI Extension configuration file

The GAS ISAPI Extension configuration file is an (ini) file called isapidispatch.ini. It contains several options that the GAS ISAPI Extension uses when starting up.

The isapidispatch.ini file is created during the installation. You must copy this file into the ISAPI Extension root directory. It is read on startup only. If the file is modified, the GAS ISAPI Extension has to be restarted before the changes take effect.

The GAS ISAPI Extension configuration file may contain the following sections and properties:

<table>
<thead>
<tr>
<th>Section</th>
<th>Property</th>
<th>Default value</th>
<th>Required</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>as-directory</td>
<td>N/A</td>
<td>YES</td>
<td>The FGLASDIR directory</td>
</tr>
<tr>
<td>Options</td>
<td>configuration-file</td>
<td>FGLASDIR\etc \as.xcf</td>
<td>NO</td>
<td>The GAS configuration file</td>
</tr>
</tbody>
</table>

Sample isapidispatch.ini

```
# GAS ISAPI Extension configuration file
...
[Options]
# The GAS installation directory.
# This option is required.
as-directory=C:\FourJs\gas

# The GAS main configuration file
# This option is optional
# Default value: <as-directory>\etc\as.xcf
configuration-file=
```

Troubleshooting installation

If the demos page cannot be reached after the ISAPI installation process, this topic may help you understand what went wrong.

The GAS and Genero BDL must be correctly installed. To check this, try to reach the demos page by using the standalone GAS. If you are using Internet Explorer as the Web browser, ensure that the Show friendly HTTP error messages in the Advanced tab of the internet options dialog is unchecked.

If the page that displayed is like the following, check that the GAS ISAPI Extension configuration file is located in the application root directory and named isapidispatch.ini:

```
Genero Application Server - 2.20.01-41876 - Failed to start!
Started on ............ 2008/09/29 16:25:51
[done]
File System initialization ..................................................
[fail]
The installation directory was not found.
```

The message displayed could help to find out what part of the start up process has failed. See GAS ISAPI Extension configuration file on page 88.

In all other cases, check the installation process.
Restarting the ISAPI dispatcher

The Internet Information Services (IIS) Administration tools provide the mechanism to restart the ISAPI dispatcher.

Please consult your IIS documentation for details.

Restarting the ISAPI dispatcher does not stop all proxies. Sessions are reloaded and applications continue to work. Proxies are only stopped when the USER_AGENT timeout expires.

FastCGI Installation and Web Server Configuration

How to configure the FastCGI extension for various Web Servers.

This section presumes that you have knowledge of fastcgi. This page will only help you configure the fastcgi module to properly work with GAS in a standard way. If you encounter any issues on fastcgi installation or need additional configuration (like fastcgi options), please refer to the fastcgi documentation or contact your system administrator.

**Note:** GAS supports mod_fastcgi but not mod_fcgid.

- Using the FastCGI dispatcher on page 89
- FastCGI GAS configuration on various Web Server on page 89
- Troubleshooting on page 95
- Restarting the FastCGI dispatcher on page 96

Using the FastCGI dispatcher

The GAS supports FastCGI. FastCGI is a protocol for interfacing the Web Server and Application Server, like the Common Gateway Interface (CGI) protocol.

The main advantages of FastCGI include:

- Independence of Web Server used. The Web Server simply needs to have a FastCGI extension.
- Instead of creating a new process for every request (as is done with CGI), FastCGI communicates with the GAS, which handles many requests over its lifetime.
- FastCGI can manage the GAS dispatcher process [Start, Stop, Relaunch on Failure].

FastCGI extension manages the GAS dispatcher process:

- The Web Server and GAS will have the same lifetime: starting the Web Server will start the GAS process, while stopping the Web Server will stop the GAS dispatcher process. If the GAS dispatcher fails, the Web Server restarts a new GAS process.
- The Web Server and GAS must be on the same computer.

GAS FastCGI support is provided by using the `fastcgidispatch` executable, see Dispatcher: `fastcgidispatch` on page 306).

FastCGI GAS configuration on various Web Server

Details are provided for various configurations. Regardless of the configuration, a common URL launches a Genero application using the Genero Application Server.

A typical URL would be:

```plaintext
http://host/gas/ua/r/application
```

**Note:** Assume the Genero Application Server is installed in the following directory (FGLASDIR): `/opt/gas`. Make the appropriate substitution for the FGLASDIR when applying these examples to your own configuration.
Apache: mod_fastcgi

Module mod_fastcgi is the Apache module for FastCGI support.

Note: Assume the Genero Application Server is installed in the following directory (FGLASDIR): /opt/gas. Make the appropriate substitution for the FGLASDIR when applying these examples to your own configuration.

Important: Apache 2.4 does not support mod_fastcgi. If you are using Apache 2.4, use mod_proxy_fcgi instead, see Apache 2.4: mod_proxy_fcgi on page 91.

Note: For more information on the Apache modules, refer to the Apache documentation.

mod_fastcgi installation

Install software package for your system or use these instructions.

mod_fastcgi configuration to manage GAS process

Add these lines to your Apache configuration file:

```apache
LoadModule fastcgi_module /usr/lib/apache2/modules/mod_fastcgi.so

<IfModule mod_fastcgi.c>
  FastCgiServer /opt/gas/bin/wrapper.fcgi -idle-timeout 300
  -initial-env FGLASDIR=/opt/gas
  Alias /gas /opt/gas/bin/wrapper.fcgi
</IfModule>

# set permissions for /gas alias
<Location /gas>
  Order Deny,Allow
  Deny from all
  Allow from mycompany.com
</Location>
```

In the fastcgi configuration example:

- `wrapper.fcgi` is a script delivered with GAS installation that simplifies FastCGI configuration. You can amend this script to add options for `fastcgidispatch`, like `-f` to specify a custom configuration file.
- `/gas` directory is just a virtual directory, no need to create one.
- `/gas` alias permission is set to deny all access to GAS except for clients from mycompany.com. You can modify the alias configuration to your needs. For more details on Apache directives, see Apache documentation.
- `-idle-timeout` must be greater than REQUEST_RESULT timeout in GAS configuration. [By default: `<REQUEST_RESULT> : 240 seconds, mod_fastcgi"-idle-timeout" : 300 seconds]

Using mod_deflate for compression with mod_fastcgi

Normally on Apache web server, compression is enabled through the mod_deflate module. You should be aware that there is a known bug (see [http://osdir.com/ml/ubuntu-bugs/2009-05/msg47891.html](http://osdir.com/ml/ubuntu-bugs/2009-05/msg47891.html) ) resulting in
in incorrect content-length header being returned when pages are loaded as the content-length is not updated after compression. You may begin, therefore, to notice a slow down in your applications as:

- The client (user agent) waits to receive content that doesn't exist while it is unaware that it already has the entire response.
- The client (user agent) waits until eventually a timeout is reached.

Moreover as GAS does compression by default (see Compression in Genero Application Server on page 154), mod_deflate's functionality is really unnecessary. It is therefore recommended that you disable mod_deflate. For more information on disabling mod_deflate, please refer to the fastcgi documentation or contact your system administrator.

**Apache 2.4: mod_proxy_fcgi**

With Apache 2.4, mod_proxy_fcgi is used instead of mod_fastcgi.

**Note:** Assume the Genero Application Server is installed in the following directory (FGLASDIR): /opt/gas. Make the appropriate substitution for the FGLASDIR when applying these examples to your own configuration.

Apache 2.4 does not officially support mod_fastcgi, use mod_proxy_fcgi instead. This module requires Genero Application Server 2.50 (or later). For more information on Apache modules, refer to the Apache documentation.

1. Find the Apache configuration file (for example, httpd.conf). It is likely to be located in the /etc/apache2/ path.
2. Ensure the modules mod_proxy and mod_proxy_fcgi are enabled.
   
   For example, these lines must appear in your Apache configuration.

   ```
   LoadModule proxy_module modules/mod_proxy.so
   LoadModule proxy_fcgi_module modules/mod_proxy_fcgi.so
   ```

3. Add the following lines to the configuration file:

   ```
   Note: Starting with Apache 2.4.11, you can add the enablereuse=on option in the ProxyPass configuration line, in order to recycle connections to the fastcgi dispatcher.

   ... 
   <IfModule mod_proxy_fcgi.c>
   #No PATH_INFO with mod_proxy_fcgi unless this is set
   SetEnvIf Request_URI . proxy-fcgi-pathinfo=1
   ProxyPass /gas/ fcgi://localhost:6394/ enablereuse=on
   Alias /gas /opt/gas/bin/fastcgidispatch
   </IfModule>
   ... 
   ```

   In this excerpt:

   - localhost is where the fastcgidispatch is running
   - fastcgidispatch is listening on port 6394
   - /gas/ directory is just a virtual directory, no need to create one
   - The Alias directive specifies where documents are stored in the local file system

4. Start fastcgidispatch in standalone mode with fastcgidispatch -s. If this dispatcher fails, it must be restarted manually.

**Apache 2.4: Configuring mod_proxy_fcgi for remote server**

Configure Apache 2.4 fastCGI with mod_proxy_fcgi module for a remote GAS server.

If you are configuring mod_proxy_fcgi to connect to a remote GAS server, the requirement for the modules and the configuration details are similar to those described in Apache 2.4: mod_proxy_fcgi on page 91; you just need to configure the Apache configuration file as shown in the examples here.
Add the following lines to the Apache configuration file:

Note: Starting with Apache 2.4.11, you can add the `enablereuse=on` option in the ProxyPass configuration line, in order to recycle connections to the fastcgi dispatcher.

```
...<IfModule mod_proxy_fcgi.c>
  # No PATH_INFO with mod_proxy_fcgi unless this is set
  SetEnvIf Request_URI . proxy-fcgi-pathinfo=1
enablereuse=on
  Alias /gas /opt/gas/bin/fastcgidispatch
</IfModule>
...```

- `<gas-server-ip>` is the address of the remote server where the GAS is running.
- `<gas-server-port>` is the port where fastcgidispatch is listening.
- `/gas/` directory is just a virtual directory, no need to create one
- The `Alias` directive specifies where documents are stored in the local file system

**Rewrite-rule for authorization headers**

If you are using authorization headers, you need to be aware of the following:

- Authorization headers need to be base64 encoded.
- A rewrite rule (see Apache 2.4 documentation) can be created and a `ProxyPassReverse` directive set in order to adjusts or rewrite URLs on the fly before forwarding them on to the client.

For example, an authorization request will cause a redirect to an authentication server and Apache adjusts this URL to the local URL before forwarding the HTTP redirect response to the client. For more details on Apache directives, see Apache 2.4 documentation.

```
...<IfModule mod_proxy_fcgi.c>
  # No PATH_INFO with mod_proxy_fcgi unless this is set
  SetEnvIf Request_URI . proxy-fcgi-pathinfo=1
  # Need to rewrite authorization header
  RewriteEngine on
  RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
enablereuse=on
  Alias /gas /opt/gas/bin/fastcgidispatch
</IfModule>
...```

**Fastcgi for nginx**

Nginx has a fastcgi module. The nginx' fastcgi module is not the same as mod_fastcgi for Apache.

Note: Assume the Genero Application Server is installed in the following directory (FGLASDIR): `/opt/gas`. Make the appropriate substitution for the FGLASDIR when applying these examples to your own configuration.
Edit the Web site configuration file (for example, located in /etc/nginx/sites-enabled/default).

Before the server {...} paragraph, add:

```plaintext
... upstream fcgi_backend {
    server 127.0.0.1:6394;
    keepalive 32;
}
...
```

In this excerpt:
- 127.0.0.1 is the ip where the fastcgidispatch is running.
- 6394 is the port where fastcgidispatch listens to.

Inside the server {...} paragraph, configure the fastcgi module to reuse socket connections for requests:

```plaintext
... location /gas/ {
    fastcgi_keep_conn on;
    fastcgi_pass fcgi_backend;
    include fastcgi_params;
}
...
```

In the fastcgi_params file (for example, located in /etc/nginx/), add:

```plaintext
... fastcgi_split_path_info (/gas) (/?.+)$;
fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;
fastcgi_param PATH_INFO $fastcgi_path_info;
fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;
...
```

In this excerpt:
- /gas is the gas connector alias.

In the fastcgi_params file, find the line that reads:

```plaintext
fastcgi_param SERVER_NAME $server_name;
```

and replace with:

```plaintext
fastcgi_param SERVER_NAME $host;
```

The fastcgidispatch needs to be started in standalone mode: fastcgidispatch -s.

If this dispatcher fails, it must be restarted manually.

**Lighttpd**

The Lighttpd Web server supports natively FastCGI protocol.

**Note:** Assume the Genero Application Server is installed in the following directory (FGLASDIR): /opt/gas. Make the appropriate substitution for the FGLASDIR when applying these examples to your own configuration.

For more information, see the ModFastCGI documentation provided by Apache.
ModFastCGI configuration to manage GAS

Add these lines to your Lighttpd configuration file:

```plaintext
server.modules += ( "mod_fastcgi" )
fastcgi.server = ( "/gas" => 
  ( 
    "host" => "127.0.0.1",
    "port" => <gas-server-port>,
    "check-local" => "disable",
    "bin-path" => "/opt/gas/bin/wrapper.fcgi",
    "bin-environment" => ( 
      "FGLASDIR" => "/opt/gas",
    ),
    "max-procs" => 1
  )
)
```

Note:

- The "server.max-write-idle" global parameter must be greater than request timeouts in the GAS configuration.

Sun Java System Web Server 7.0

Sun Web Server 7 has an integrated FastCGI support.

Note: Assume the Genero Application Server is installed in the following directory (FGLASDIR): /opt/gas. Make the appropriate substitution for the FGLASDIR when applying these examples to your own configuration.

For more information, search for documentation on “FastCGI Plug-in” provided by [http://docs.oracle.com/en/](http://docs.oracle.com/en/).

Enable FastCGI Plug-in

Add this line to your magnus.conf configuration file:

```plaintext
Init fn="load-modules" shlib="libfastcgi.so"
```

FastCGI Plug-in configuration to manage GAS

Add these lines in your obj.conf configuration file:

```plaintext
<Object name="default">
  ...
  NameTrans fn="assign-name" from="/gas/*" name="gas.config"
  ...
</Object>
<Object name="gas.config">
  Service fn="responder-fastcgi" app-env="FGLASDIR=/opt/gas"
  app-path="/opt/gas/bin/wrapper.fcgi" reuse-connection="true"
  resp-timeout="300" restart-interval="0"
</Object>
```

Note:

- The "resp-timeout" must be greater than the request timeouts in the GAS configuration.
Troubleshooting

Troubleshooting tips addressing common issues.

• Why does my application timeout on Apache? on page 95
• Why does my application not work with fastcgi? on page 95
• Invalid installation directory on page 95
• Applications all down at the same time on page 95

Why does my application timeout on Apache?

The Apache module `mod_reqtimeout` controls the request data rate. If the reception of the data is considered too slow, Apache can close an HTTP connection before other configured timeouts have expired.

When a response is written to an HTTP request where Apache has closed the socket, you may get a "SYSTEM ERROR" logged in the dispatcher log. As this is a system error, the message outputted may vary from system to system.

It may also happen that the dispatcher can respond correctly, but that the Web server is unable to forward the answer. In this case, there is no error in the GAS so you must check the Web server logs.

For more information or assistance you can also contact your local Four Js support center for help.

Important:

Starting with GAS version 3.00, the `mod_reqtimeout` module is no longer loaded by the GAS configuration for Apache 2.4.

For more information on working with the Apache timeout, see the "Apache Module mod_reqtimeout" page in the Apache documentation.

Why does my application not work with fastcgi?

Your fastcgi might be misconfigured or does not have the right permissions.

To debug, add these lines in your `$FGLASDIR/bin/wrapper.fcgi`:

```
echo $FGLASDIR >> /work/tmp/log.txt
ls -al $FGLASDIR >> /work/tmp/log.txt
strace -f -F -tt -s 3000 "$FGLASDIR"/bin/fastcgidispatch >> /work/tmp/log.txt 2>&1
```

`log.txt` shows the system calls. Most of the time, you can see "permission denied" on some directories or files.

If you do not find any clues, please contact your local support center with `log.txt` attached.

Invalid installation directory

The fastcgi does not start the Genero Application Server and displays "Invalid installation directory".

Examine two items:

• The FGLASDIR environment variable must be set to the Genero Application Server installation directory.
• The Genero Application server may have been installed by a different user than the one starting the fastcgi (typically, the user owning the web server process). To start the Genero Application Server, the user needs permissions on the Genero Application Server log directories and `$FGLASDIR/tmp`.

Applications all down at the same time

The GAS is probably down. You can find some clues in:

• GAS logs (See Table 5: appdata Directories and Files on page 41 for details on the location of log files)
• Web server logs
One reason could be a cron or a script that periodically shuts down and restarts the Web server. For example, in the Apache log_error file you can find messages like:

```... [notice] SIGUSR1 received. Doing graceful restart
... [error] [client xx] (4) Interrupted system call: FastCGI: comm with server "/opt/gas/bin/wrapper.fcgi" aborted: poll() failed
... [error] [client xx] FastCGI: incomplete headers (0 bytes) received from server "/opt/gas/bin/wrapper.fcgi"
```

Apache received a SIGUSR1 signal. If you search through the error_log, you might have the message daily at the same hour. This is probably due to logrotate (see /etc/logrotate.d).

In order to archive the logs, logrotate sends a request to Apache to restart gracefully. This shuts down the fastcgi, the dispatcher and all Genero applications. You can check if this is really the cause of the issue by running this command (or an equivalent one):

```
/etc/cron.daily/logrotate --force /etc/logrotate.d/apache2
```

To solve, amend your logrotate to process when Apache stops, or at night, or save the logs of the previous day so that you can archive the logs without stopping Apache.

### Restarting the FastCGI dispatcher

To restart fastcgidispatch, use:

```
kills -9
```

Once the web server restarts the dispatcher, the dispatcher uses the session table to reconnect to the various proxies. The applications are still maintained by proxies, are still running, and once the dispatcher is relaunched, the user can continue his or her work.

Ctrl + C or sending SIGTERM will stop the standalone dispatcher, and in both cases the dispatcher will request all proxies to stop. The fastcgi dispatcher will stop sessions on Ctrl-C as well if started in standalone mode - but not on SIGTERM.

With kill -9 the dispatcher process is killed yet the sessions remain alive and untouched. When the dispatcher is restarted, the sessions continue to be active.

### Java™ Servlet Installation and Web Server Configuration

Details around the Java™ Servlet installation and Web Server configuration.

- Using the GAS Java dispatcher on page 96
- Building the Java Web Archive (WAR) on page 97
- Deploying on a Java Web Server on page 98
- Restarting the J2EE dispatcher on page 98

### Using the GAS Java™ dispatcher

A default Java™ dispatcher called java-j2eedispatch is configured by default to use the Genero Application Server.

The Genero Application Server can be deployed on any Java™ server supporting servlets (such as Tomcat, WebSphere®, GlassFish or JBoss) and be able to use the java-j2eedispatch dispatcher.

**Note:** Make sure you are using a Java™ servlet container compatible with J2EE Servlet API v.3.0 or above.

The Java™ servlet dispatcher, java-j2eedispatch, is located in $FGLASDIR/war. It provides the same functionality as the other GAS dispatchers, see GAS Dispatchers on page 304.
URLs used to access the Java™ dispatcher depends on the port number configured for the Java™ server and the servlet name defined in the Java™ servlet deployment file (web.xml), but typically will have the following form:

http://host:port/java-j2eedispatch/ua/r/app_name

For example:

http://localhost:6394/java-j2eedispatch/ua/r/gwc-demo

Typically no configuration is necessary, unless you make changes to the default location of the Genero Application Server installation, for example the configuration file (default as.xcf). Then you will need to reconfigure and redeploy the new Java™ servlet configuration as detailed in the topics in this section:

- Building the Java Web Archive (WAR) on page 97
- Deploying on a Java Web Server on page 98

### Building the Java™ Web Archive (WAR)

Changes to default locations of the GAS installation affect the Java™ servlet, which can be reconfigured by building a new Java™ Web archive using the fglgar tool.

You will only need to perform the tasks described here if one or all of following changes have been made to the default GAS installation:

- Change to the default location for the GAS installation directory (FGLASDIR).
- Change to the default location of the GAS configuration file (default FGLASDIR/etc/as.xcf).
- Change to the default name of the Java servlet and Java dispatcher (default java-j2eedispatch).

For example, the following step describes how to generate a Web archive for the Java™ Servlet and dispatcher with the following GAS configuration:

- The Genero Application Server located at C:/usr/gas/3.00
- The GAS configuration file located at C:/conf/as.xcf
- The servlet and dispatcher named java-j2eedispatch

1. Rename the default Java™ servlet deployment file (default FGLASDIR/war/WebContent/WEB-INF/web.xml) to, for example, web.old
2. Create a copy of the default backup deployment file (default FGLASDIR/war/WebContent/WEB-INF/web.bak) to web.xml

   **Caution:** Do not delete, rename, or make any modifications to web.bak. It may only be used to make a copy.

3. Execute the fglgar command in the FGLASDIR/war/WebContent directory replacing your options for the settings for the GAS installation (--asdir) and configuration (--asxcf) for those shown in the example:

   ```
   fglgar --war --asdir C:/usr/gas/3.00 --asxcf C:/conf/as.xcf --output java-j2eedispatch
   ```

   **Note:** The name of the archive must match the servlet name set in the deployment file, web.xml. For example, if the servlet is called java-j2eedispatch, generate a Web archive called java-j2eedispatch.war. The fglgar tool does this automatically for you.

   A Web archive, e.g. java-j2eedispatch.war, is created.

When you have completed building the Web archive in the above step, your next task is to deploy the new Java Servlet. This is detailed in Deploying on a Java Web Server on page 98.
Deploying on a Java™ Web Server

Deploying a WAR on a Java™ Web server depends on the Java™ server; however it is relatively easy using this procedure.

1. Using your browser, go to the Java™ Server manager web page.
2. In the deployment section, choose the java-j2eedispatch.war file (created previously, see Building the Java Web Archive (WAR) on page 97) to download.
3. Click on the deploy button.
4. Check that java-j2eedispatch is available in the list of web applications, with a status of success.
5. In another browser tab, check that you can load the GAS demos page.

For example:
- http://host:port/java-j2eedispatch/demos.html
- http://host:port/java-j2eedispatch/ua/r/gwc-demo

The application opens, and the Java™ dispatcher is ready to serve GAS requests, in accordance with the as.xcf configuration file set.

Note: In case of failure, refer to your Java™ server war deployment guide.

Restarting the J2EE dispatcher

Restarting the Java™ servlet and Java dispatcher depends on the Java™ server; however it is relatively easy using this procedure.

Restarting the java-j2eedispatch dispatcher depends on the J2EE Server. Typically, there is a simple Web interface to deploy and undeploy the war files. Please consult your J2EE Server manual about how to restart.

Note: When you restart Java servlet and Java dispatcher, the session tables, which are stored in memory, are lost. As a result, it is not possible to recover applications after a restart.

1. Using your browser, go to the Java™ Server manager web page.
2. To stop the java-j2eedispatch dispatcher, select the option to undeploy the war file
3. To restart, select the option to deploy the war file as detailed in Deploying on a Java Web Server on page 98.
4. Check that java-j2eedispatch is available in the list of web applications, with a status of success.
5. In another browser tab, check that you can load the GAS demos page.

For example:
- http://host:port/java-j2eedispatch/demos.html

The application opens in a browser, and the Java™ dispatcher is ready to serve GAS requests.

Note: In case of failure, refer to your Java™ server war deployment guide.

Validating configuration files

The Genero Application Server provides XML Schema Definition (XSD) files, which can be used to validate your Genero Configuration Files (xcf) in Genero Studio as well as any enhanced XML editor.

By default, Genero Studio is configured to support all Genero Application Server configuration grammar.

- What is an XML Schema Definition file? on page 99
- Why specify the XML Schema Definition file? on page 99
- Validating with the gasadmin tool on page 99
What is an XML Schema Definition file?
An XML Schema Definition (XSD) describes the structure of an XML document.

An XML Schema defines the building blocks of the XML document. An XSD describes the elements and attributes that can appear in a document, the data types for elements and attributes, the number of (and order of) child elements, as well as default and fixed values for elements and attributes.

XSD is fully recommended by W3C consortium as a standard for defining an XML document, and has replaced the use of Document Type Definition (DTD) files. For more information on XSD, please refer to the W3C consortium web site at http://www.w3.org.

Why specify the XML Schema Definition file?
XML schema definition (xsd) file are used by XML editors to provide validation and syntax hints when editing the configuration files. They are also used by the dispatcher to validate the application configuration.

When you create a configuration file for the Genero Application Server or for an application, you provide the path to an XML schema definition file (xsd).

- In the GAS configuration file (as.xcf by default), this entry exists:

  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfas.xsd"

- For external application configuration files, this entry should exist:

  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd"

The XML editor either looks on the Web or in its local schemas catalog for the specified XML schema definition file.

While the Genero Application Server validates the configuration files, it does not rely on these entries within the configuration files themselves. The validation is completed by the dispatcher using the schemas provided in $FGLASDIR/etc.

Validating with the gasadmin tool
Describes some options that can prove useful in validating the GAS configuration file with the gasadmin tool.

Syntax

gasadmin [options]

--configuration-check
The configuration-check option validates the GAS configuration file (as.xcf) and exits. Errors are displayed to error output.

--configuration-explode
The configuration-explode option explodes the GAS configuration file into separate files, one for each application, which are then stored in the temporary directory defined by the res.path.tmp resource of the GAS configuration file. Each file lists the entire configuration for an application, expanding the inherited components.

<EXECUTION>
<ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="GREDIR">$(res.gredir)</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="FGLLDPATH">$(res.fglldpath)</ENVIRONMENT_VARIABLE>

<PATH>$(res.path.fgldir.demo.services)/calculator/server</PATH>

...<

</EXECUTION>

--configuration-expand-resources

The configuration-expand-resources option expands the GAS configuration file’s resource elements and replaces them with real values. Can be used in combination with --configuration-explode or --configuration-explode-external options to expand components of the configuration file.

<EXECUTION>

<ENVIRONMENT_VARIABLE Id="FGLDIR">C:\4js\fgl</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="GREDIR">#!GREDIR!#</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="PATH">C:\Windows;C:\4js\gas\2.50.34\bin;#!GREDIR!#
    \bin;C:\4js\fgl\bin;C:\4js\fgl\lib</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="FGLLDPATH">C:\4js\gas\2.50.34\lib;#!GREDIR!#
    \lib;C:\4js\fgl\lib</ENVIRONMENT_VARIABLE>

<PATH>C:\4js\fgl/demo/WebServices/calculator/server</PATH>

...

</EXECUTION>

Selecting an XML editor

With a good XML editor, you can validate your configuration files.

With a good XML editor, you can:

1. Check that your XML file is well-formed.
2. Validate your XML file against the referenced XML Schema Definition file.
3. Add new elements with completion assistance.

Genero Studio can be used to write and edit configuration files for the Genero Application Server, using the XML Schema Definition referenced in the file to validate.

Any XML editor that uses the XML Schema Definition file (xsd) to validate the XML is a valid candidate. Any search for "XML editor" will return a long list of such XML editors. One well-known XML editor is Altova XML Spy. A fuller list of tools can be found on the XML Schema page of the W3C consortium, under Tools (http://www.w3.org/XML/Schema).

Configuring applications on GAS

Understand the options available for configuring and deploying applications on the GAS.

**Note:** For examples of how to get started with configuring, running, and deploying basic types of applications on the GAS, please see GAS Quick Start Guide on page 16.

The topics in this section provide more options for delivering applications on the GAS.

- Application Configuration Overview on page 101
- Create abstract applications on page 101
- Create an application group on page 102
- Create an application configuration file on page 103
- Using External Application Configuration Files on page 105
- Configure DVM environment variables on page 109
- Use a script to set the environment on page 110
- Troubleshooting application Issues on page 111
- Next steps on page 111
Application Configuration Overview

To run an application, configuration details must be provided to the Genero Application Server.

Many of the configuration details for one application end up being the same for a set of applications. Rather than provide the same configuration details in each application's configuration, the concept of *inheritance* allows you to define the common configuration details in a single place, to be inherited and used by multiple applications.

With Genero, you typically define an abstract application to hold the common configuration details. An abstract application is not executable. It is intended to provide the baseline default configuration for other applications to inherit. You can create as many abstract applications as you need. Abstract applications can inherit a base configuration from another abstract application.

You configure your application to inherit the settings of the abstract application. To inherit a configuration of another application, specify the other application as the parent. There is no limit to the levels of inheritance an application can inherit from another application (abstract or not) that inherits from another application, and so on.

To provide the configuration for an executable application, either:

- Provide the configuration details in the GAS configuration file. When you add the application to the GAS configuration file, you must restart the GAS for the application to be recognized.
- Create an application-specific configuration file (one per application). See Create an application configuration file on page 103. When you create an external application configuration file and add the file into a defined GROUP directory, the application is immediately available without having to do a GAS restart. See Create an application group on page 102.

Create abstract applications

To simplify application configuration, an application can specify a parent application to provide a default configuration for the application.

An abstract application is not an executable application. It is only intended to be a parent application, providing configuration defaults for executable applications. You should create your own abstract application and use it as the parent for a set of programs that share common configurations.

**Tip:** If you use this inheritance mechanism efficiently, you can configure new applications with only a few entries in the configuration file.

**Important:** Abstract applications can only be defined in the GAS configuration file. They cannot be defined using an external application configuration file.

Default Genero web client abstract application

The default application for the Genero web client, `defaultgwc`, is found in the Genero Application Server configuration file.

```xml
<!--This is the default application for Genero web client-->
<APPLICATION Id="defaultgwc" Parent="defaultwa" Abstract="TRUE">
  <OUTPUT Rule="UseGWC">
  </OUTPUT>
</APPLICATION>
```

1. This application inherits the configuration of the `defaultwa` abstract application, also defined in the GAS configuration file.
2. To specify an abstract application, set the Abstract attribute to `TRUE`.
3. The OUTPUT Rule identifies which front-end is used to display the application, as set in the `defaultwa` abstract application. In this case the front-end target is the Genero Web Client for JavaScript (GWC-JS).
Example for Web services abstract application

```xml
<Application Id="ws.default" Abstract="TRUE">
  <Execution Using="cpn.ws.execution.local"/>
  <Timeout Using="cpn.ws.timeout.set1"/>
</Application>
```

Create an application group

A *group* defines an alias for a directory where application configuration files can be stored. The alias is used in the application URL, letting the Genero Application Server (GAS) know where to find the application configuration file.

A group consists of an alias (Id) and a directory (path). When a front-end requests an application whose configuration information is stored in an external application configuration file, it provides the group alias, which directs the GAS to the directory where the application configuration file sits. The application name identifies which application configuration file to read (as the application and the configuration file share the same name). A GROUP (for an application) on page 373 element can be added to the APPLICATION_LIST component within the GAS configuration file.

**Syntax**

```xml
<GROUP Id="groupId">path</GROUP>
```

1. *groupId* is the alias
2. *path* is the physical path to the directory

**Usage**

You can use application groups to organize your applications into logical groups or a hierarchy. For example, consider this URL:

http://<server>/gas/ua/r/accounting/app1

In this URL, both a group (accounting) and an application name (app1) are specified. The GAS, on receiving this application request, uses the group alias to identify the directory holding the external application configuration file:

```xml
<GROUP Id="accounting">/path/config/accounting</GROUP>
```

In this directory, the GAS expects to find a file whose name matches the name of the application with an *xcf* suffix. For this example, the GAS looks for a file named *app1.xcf*.

**The default group**

The GAS configuration file provides a default group, defined using the name *_default*. When an application configuration file is added to this group, the application URL can omit using a group name and simply reference the application. For example, consider this URL:

http://server/gas/ua/r/Edit

The application URL does not specify a group, and the Edit application is not defined internally. It must therefore be defined in an external application configuration file, located in the directory defined for the *_default* alias.

```xml
<GROUP Id="_default">$(res.path.app)</GROUP>
```
The resource $(res.path.app)$ resolves to appdata/app, appdata is described in GAS directories on page 39. In this directory, you would expect to find Edit.xcf, the Edit application’s application configuration file.

**Example 1: "myapp" group defined by path to directory**

```
<GROUP Id="_default">$(res.path.app)</GROUP>
<GROUP Id="myapp">$(res.path.app)/myapp</GROUP>
```

**Example 2: "demo" group defined by resource**

```
<GROUP Id="demo">$(res.path.demo.app)</GROUP>
```

This example assigns the alias demo to the directory containing the external application configuration files for demo applications. The path is defined using the resource $(res.path.demo.app). By wisely using a resource, a change to the directory structure only requires a change to a single RESOURCE element in the configuration file.

To access an application that has its configuration file stored in the group directory, enter an application URL that includes the group alias in its path: http://server/gas/ua/r/demo/CardStep1

Based on this URL, the GAS would expect to find the configuration file CardStep1.xcf within the directory specified for the demo group.

**Create an application configuration file**

An application configuration file provides the Genero Application Server (GAS) with the information needed to run an application. It becomes available for use as soon as created and when added to a recognized group directory.

Once you have created an application, you need to configure it so that it can be executed by the GAS. For this you need to create an application configuration file. Typically, the name of the file matches the name of the application and has an xcf suffix. For example, if the application name was "app1", create a configuration file named app1.xcf.

Save the file in a defined GROUP directory. By default, the directory where the GAS searches for external application configuration files is defined in the GAS configuration file (default as.xcf) by the tag `<GROUP Id="_default">directory</GROUP>`. You can specify alternate directories; see GROUP (for an application) on page 373 or GROUP (for a service) on page 374.

The configuration file defines an application environment, and starts with the Application element. Within this element, you can define local resources, change the execution environment, the timeout settings, the image, and output settings. You can refer to previously defined components by using the tag attribute Using.

The organization of the elements within the application configuration file depend on the type of application. See Application configuration files on page 342.

These examples show some well-formed external application configuration files.

**Example 1 - A simple application configuration file**

The simplest application configuration file specifies a parent application and the path to the compiled application files. The application inherits the configuration of the parent application. The file is named appname.xcf, where appname is the name of the application.

```
<APPLICATION Parent="defaultgwc">
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd"
```
In this example, the external application configuration file `Edit.xcf` provides a configuration for the Edit application. Things to note:

1. The application name is the name of the configuration file. The GAS uses the configuration file name to identify the application. The `Id` attribute of `<APPLICATION>` element is omitted; even if included, its value is not read.

2. The application configuration file is re-read at each application launch. There is no need to restart the GAS after modifying an application configuration file.

3. In this example, the `Parent` application is `defaultgwc`. A parent application is an abstract application that must be defined in the GAS configuration file. It provides default component configurations, which applications can inherit. See Create abstract applications on page 101.

4. The path to the application executables is defined by the `PATH` component. See PATH (under EXECUTION) on page 390.

5. The `MODULE` element can specify the name of the `.42r` module to run. If the module name is the same as the configuration file name, the `MODULE` element is not necessary, the module name used is the name of the application. See MODULE on page 387.

---

**Example 2 - A Web Client for JavaScript (GWC-JS) Application Configuration**

While an application inherits its base configuration from the parent application, additional configuration elements can be added and existing configuration elements can be overwritten.

This example configuration file is for a GWC-JS Web application.

```
<APPLICATION Parent="defaultgwc">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}/Widgets</PATH>
  </EXECUTION>
  <!-- Override the default gwc-js with gwc-js-custom-->
  <UA_OUTPUT>
    <PROXY>${res.uaproxy.cmd}</PROXY>
    <PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
    <GWC-JS>gwc-js-custom</GWC-JS>
    <TIMEOUT> Using="cpn.wa.timeout"</TIMEOUT>
  </UA_OUTPUT>
</APPLICATION>
```

1. The `UA_OUTPUT` on page 414 element provides configuration details for applications rendered by the UA proxy.

2. The `GWC-JS` on page 374 element provides the directory name `gwc-js-custom` for the customized project, for more information on customization see Configure your environment on page 217.

3. The `PUBLIC_IMAGEPATH` element specifies the public resources directory used for common images used by applications.
Configuring the Genero Application Server

Example 3 - Specifying the use of the HTML5 theme (and other overrides)

This next example is of a hypothetical external application configuration file, tutorialStep1.xcf, for a GWC-HTML5 Web application.

Note: The GWC-HTML5 Web client has been deprecated; new development should use the GWC-JS instead.

```xml
<APPLICATION Parent="demo-tut-abstract">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <!-- Define a resource to the template HTML file -->
  <RESOURCE Id="res.template.tutorial"
    Source="INTERNAL">$(res.path.demo.dem-tut)/web/tutorial/tutorialStep1.html
  </RESOURCE>
  <EXECUTION>
    <PATH>$(res.path.demo.dem-tut)/src</PATH>
    <MODULE>tutStep1.42r</MODULE>
  </EXECUTION>
  <!-- Override default rendering template -->
  <OUTPUT>
    <MAP Id="DUA_HTML5">
      <THEME>
        <TEMPLATE Id="_default">$(res.template.tutorial)</TEMPLATE>
      </THEME>
      <OUTPUT>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

1. The Parent attribute of the APPLICATION element defines the parent application as "demo-tut-abstract". The child application inherits the configuration elements defined by the parent application.
2. The RESOURCE element defines a local resource. This resource maps to a template file. See RESOURCE (for an application) on page 401.
3. The PATH element lists the path to the executable. See PATH (under EXECUTION) on page 390.
4. The MODULE elements provide the path and file name of the program executable. The MODULE element is often excluded, when the executable name matches the application name as provided in the URL. In this example, had the external application configuration file been named "tutStep1.xcf", then the MODULE element could have been excluded. See MODULE on page 387.
5. The Id attribute of the MAP element defines the output map as DUA_HTML5. This means that the application will use the deprecated GWC for HTML5 theme. See MAP on page 384.
6. The TEMPLATE element overrides the default template (Id="_default") with the template defined by the resource "$(res.template.tutorial)". Recall that this resource was defined at the start of this file using a RESOURCE element. See TEMPLATE on page 410.

Using External Application Configuration Files

To configure an application with an external application configuration file, provide the same configuration details that you would for adding the application directly in the Genero Application Server configuration file.
For example, to create an external application configuration file for a program named gwc-demo, you would:

1. Add a file named gwc-demo.xcf. You must use the xcf extension.
2. Place the file in a GROUP directory (see GROUP (for an application) on page 373), as defined in the Genero Application Server configuration file.

**Configure Web client applications**

What do you need to configure a Genero Web Client application?

To add an application for Genero Web Client for JavaScript (GWC-JS), you need to specify:

- An application Id (a unique name for this APPLICATION element)

  **Note:** Applications defined in the GAS configuration file require an **Id** attribute. For external configuration files, if the application and the configuration file share the same name, there is no need to specify the **Id** attribute.

- The parent application from which to inherit configuration details (defaultgwc in this example)
- The path to the compiled application files
- The name of the application to launch
- The access control allowing access (optional)
- The customization project directory to use for the application user interface look and feel (if using a customized project)

**Example: simple application for GWC-JS Web client**

```xml
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

1. The application inherits the configuration settings of its parent ("defaultgwc" in this example).
2. The path used in this example is a RESOURCE; you could also use the absolute path name leading to your application files.
3. The MODULE contains the name of the application to launch.

**Example: gwc-demo-external.xcf**

The main differences between this example and the example shown in Example: simple application for GWC-JS Web client on page 106 are the lack of the **Id** attribute and the reference to the XML schema.

```xml
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>${res.access.control}</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```
1. The **ALLOW_FROM** on page 353 element specifies from what hosts access is allowed, the example here is defined in a **RESOURCE**.

2. The **GWC-JS** on page 374 configuration element, specifies the customization project directory you used to provide the application look-and-feel. See **Customizing GWC-JS applications** on page 203.

**Configure applications for Web service**

Create a separate application configuration (**xcf**) file for each Web services application.

What do you need to configure a Genero Web Service application?

To add an application for a Genero Web Service, you need to specify:

- Your application **Id**
  
  **Note:** Applications defined in the GAS configuration file require an **Id** attribute. For external configuration files, if the application and the configuration file share the same name, there is no need to specify the **Id** attribute.

- The parent application where the main configuration is set (in this example, **ws.default**)

- The path to your compiled files

- The main module to launch

- The access control allowing access (optional)

- The number of DVMs (**fglrun**) to start for this Web Service when the GAS starts, and the minimum and maximum number of DVMs allowed.

**Example: simple Web service application**

In the following example the configuration is for a Web service defined in the GAS configuration file. The **PATH** is a resource. The path can also be an absolute path to your application files. This configures a GWS server that any Web service client can connect to.

```xml
<Application Id="calculator" Parent="ws.default">
  <Execution>
    <Path>${res.path.fgldir.demo}/WebServices/calculator/server</Path>
    <Module>calculatorServer</Module>
  </Execution>
</Application>
```

1. The application inherits the configuration settings of its parent ("ws.default" in this example).

2. The path used in this example references a **RESOURCE** root for demo applications; you could also use the absolute path name leading to your application files.

**Example: Web Service Calculator.xcf**

In the following example, if the file was named "Calculator.xcf, then this configuration file would accomplish the same task as when included in the GAS configuration file as in the example in **Example: simple Web service application** on page 107. The main differences are the lack of the **Id** attribute and the reference to the XML schema.
Note: Because a DVM can have several services defined in it, the Web Service DVM is an application. The services defined inside are still named service. The published functions are named operations.

```xml
<Application Parent="ws.default"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextws.xsd">
  <Execution>
    <Path>$(res.path.fgldir.demo)/WebServices/calculator/server</Path>
    <Module>calculatorServer</Module>
    <Access-Control>
      <Allow_From>$(res.access.control)</Allow_From>
    </Access_Control>
    <Pool>
      <Start>0</Start>
      <Min_Available>0</Min_Available>
      <Max_Available>1</Max_Available>
    </Pool>
  </Execution>
</Application>
```

1. The Allow_From on page 353 element specifies from what hosts access is allowed, the example here is defined in a RESOURCE.
2. The Pool on page 395 element specifies the number of DVMs to start for this Web Service when the GAS starts. In this example zero DVMs at start means the Web service is not set to start with the GAS. And the maximum allowed is one DVM.

This example file can be found in $FGLDIR/demo/WebServices. In conjunction with the Example 2: "demo" group defined by resource on page 103 definition, to access the WSDL of this demo, you can use this kind of URL:

http://appserver:6394/ws/r/demo/calculator?WSDL

See Web services URI information on page 253.

Configure GDC applications
What do you need to configure a Genero Desktop Client (GDC) application?

To add an application for GDC, you need to specify:

- An application Id (a unique name for this APPLICATION element)

  Note: Applications defined in the GAS configuration file require an Id attribute. For external configuration files, if the application and the configuration file share the same name, there is no need to specify the Id attribute.

- The parent application from which to inherit configuration details ("defaultgdc" in this example)
- The path to the compiled application files
- The name of the application to launch
- The access control allowing access (optional)

Example: simple configuration for GDC application
In the following example the configuration is for a GDC application defined in the GAS configuration file.

```xml
<Application Id="my-app" Parent="defaultgdc">
  <Execution>
    <Path>$(res.path.fgldir.demo)/WebServices/calculator/server</Path>
    <Module>demo.42r</Module>
  </Execution>
</Application>
```
1. The application inherits the configuration settings of its parent ("defaultgdc" in this example).
2. The path used in this example is a RESOURCE; you could also use the absolute path name leading to your application files.
3. The MODULE contains the name of the application to launch.

**Example: gdc-demo-external.xcf**

This external configuration file would accomplish the same task as the Example: simple configuration for GDC application on page 108 that is defined in the GAS configuration file. The only differences are the lack of the **Id** attribute and the reference to the XML schema.

```xml
<APPLICATION Parent="defaultgdc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```

The **ALLOW_FROM** on page 353 element specifies from what hosts access is allowed, the example here is defined in a RESOURCE.

### Configure DVM environment variables

The application configuration file can be used to define environment variables for the DVM.

Environment variables are set with **ENVIRONMENT_VARIABLE** elements in the application configuration file.

**Syntax**

```xml
<ENVIRONMENT_VARIABLE Id="env_var">env_value</ENVIRONMENT_VARIABLE>
```

1. **env_var** is the environment variable name.
2. **env_value** is the value used to set the variable name.

**Example**

Example (using Informix® database client):

```xml
<APPLICATION Id="myapp" Parent="defaultgwc">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="DBDATE">DBMY4</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="FGLRESOURCEPATH">/home/myapp/resources</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="INFORMIXDIR">/opt/informix</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="INFORMIXSERVER">ORION</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="LD_LIBRARY_PATH">/opt/informix/lib:...</ENVIRONMENT_VARIABLE>
    <PATH>/home/myapp/bin</PATH>
    <MODULE>app.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

For more details, see **ENVIRONMENT_VARIABLE** on page 369.
Use a script to set the environment
Specify a command script for the application to launch the application.

Alternatively to the `ENVIRONMENT_VARIABLE` elements, you can use the `DVM` element to define a command that will set the environment and launch the application.

On Linux®/UNIX™ platforms, use the `DVM` element to define a shell command that will execute a shell script defined with the `MODULE` element:

```xml
<APPLICATION Id="kiosk" Parent="defaultgwc">
  <EXECUTION>
    <PATH>/home/f4gl/geh/configfiles/officestoredemo</PATH>
    <DVM>/bin/sh</DVM>
    <MODULE>gdc-kiosk.sh</MODULE>
  </EXECUTION>
</APPLICATION>
```

1. **PATH** element defines where the script is stored.
2. **DVM** element defines the command to execute the shell script defined in **MODULE**.
3. **MODULE** element defines the shell script file.

On Windows® platforms, use the `DVM` element to define a .BAT command file, including environment settings and program execution:

```xml
<APPLICATION Id="myprog" Parent="defaultgdc">
  <EXECUTION>
    <PATH>$(res.fgldir)/demo</PATH>
    <DVM>c:\myprog\launch.bat</DVM>
  </EXECUTION>
</APPLICATION>
```

1. **PATH** element defines where the script is stored.
2. **DVM** element defines the BAT command to be executed.

Override configuration resources
Options to override resources defined in the GAS configuration file can be implemented at the command line when starting the dispatcher.

Dispatcher override option (-E)
The dispatcher has an option that allows you to override a resource defined in the configuration file (as.xcf) at the command line. For example:

```
httpdispatch -E res.dvm.wa=$FGLDIR/bin/myrun
```

The value set by the override option (-E) takes precedence over the value set by the configuration file.

Override the res.path.gwcjs.user resource at runtime
Specify your customized Genero Web Client for JavaScript (GWC-JS) with the dispatcher override switch (-E) at runtime.

The `res.path.gwcjs.user` resource is not defined by default in the GAS configuration file (as.xcf). It is intended to be set at runtime.

The resource is included in the `GWC_JS_LOOKUP_PATH` path list. Therefore, the GAS is able to find the GWC-JS it references.

Override the resource with the dispatcher override switch (-E).
For example, start the dispatcher as shown:

```
httpdispatch -E res.path.gwcjs.user=c:\<my_gwc_js_directory>
```
If the specified customization directory is not found, the default GWC-JS directory is used.

**Override the appdata resource at runtime**
If you have changed the default application directory (known as "appdata") location, use this override method to specify the appdata at runtime.

Override the `res.appdata.path` resource with the dispatcher override switch (-E).
For example, at the command line start the dispatcher as shown:

```bash
httpdispatch -E res.appdata.path=/work/tmp/gas/appdata
```

**Important:** If you start the dispatcher with the option (-E) to override the $(res.appdata.path) location, you must also override the resource when using the `gasadmin` command to deploy an archive.

For example, specify the same option with both commands:

- Starting the dispatcher:
  ```bash
  httpdispatch -E res.appdata.path=/work/tmp/gas/appdata
  ```
- Deploying the application:
  ```bash
  gasadmin -E res.appdata.path=/work/tmp/gas/appdata --deploy-archive myapp.gar
  ```

**Next steps**
Other topics provide more details on application configuration and deployment.

See also:
- Developing Web applications on page 184
- Adding a Web Services Application
- Quick start: deploying applications on page 257
- Application List Reference

**Troubleshooting application Issues**
Troubleshooting tips are provided for the most common issues encountered.

- **What if the application doesn't start?** on page 65
- **Get the application session id** on page 112

**What if the application doesn't start?**
If an application does not start, you can debug the problem by manually launching the program from a command shell.

The first thing to check is the configuration information - to ensure that all components are set properly.

1. The Genero Application Server creates separate log files for its dispatchers, proxies, and the DVMs started by those proxies. Examine the logs as they may provide you with some helpful information or error messages. For more information about accessing log files see Logging on page 167.
2. Check your environment variables in `$FGLASDIR/etc/as.xcf`.

   **Tip:** You can get messages for the environment in the GAS log by setting the `CATEGORIES_FILTER` on page 361 category filter to `CONFIGURATION`.
3. You may need to run the application in debug mode using the FGL debugger.

   To run the FGL debugger, the dispatcher must open a DOS command or an xterm window so that you can run the application with the `fglrun -d` command. For example, on Windows® platform, start the dispatcher with the command to open a DOS window and override some of the settings for `res.dvm.wa`:
Before the application displays in the Web browser, a command window will open with all environment settings for that application. You can then manually run your application in debug mode, for example with `fglrun -d progname` to enter the command-line debugger (`fgldb`). The application will then display in the Web browser. See Using the debugger on page 168.

**Note:**

- You can use the graphical debugger in Genero Studio. For more information, see the Genero Studio User Guide.
- The debug facility of the Genero Desktop Client includes logging and the debug console. For more information on using the GDC debug facility, see the Genero Desktop Client User Guide.
- For details about debugging Genero Web Client for JavaScript (GWC-JS) applications, see Configure development environment on page 156.

**When you receive the Error: Runtime error. Try again ... page**

Simply put, your application cannot start and you must check your application configuration. This error is typically the result of an incorrect path to the program executable.

**Get the application session id**

The session identifier represents a GAS session. When you want to find the DVM and proxy logs to troubleshoot an application, locate the session id.

When in development mode, the default ending page for a Genero Web Client for JavaScript (GWC-JS) Web application displays the session id. See GWC-JS application ended page on page 198. You can also find the session id in the application URL of the monitor pages or by viewing the HTML source page.

Example session id in a log entry:

```
/ua/sua/7e26fadb0c9f6939c65702fc9a1ff2a4/0
```

The long string, `7e26fadb0c9f6939c65702fc9a1ff2a4` in the example, is the session id. This uniquely identifies a session, allowing you to locate the relevant proxy and DVM logs for the session.

Once the application ends, the session id is no longer available in the GUI. Or if the application fails to start, there may be no session id.

In these cases, you can only consult the logs. Usually you have a unique session log for the same period as when the session ended. See Logging on page 167.

**Rendering issues**

If an application does not render as expected, these are some things to do to help resolve it.

Graphical widgets that are not rendered properly or displayed with a different style are considered rendering issues. Examples of rendering issues include

- misalignments
- 4st colors not applied
- widgets that are rendered when they should not

When you encounter a rendering issue, first try and identify the source of the rendering issue. Find out where it originates from:

**The browser**

Run your program in different browsers. Not all browsers are equal so verify if the issue occurs in all browsers or just one. If a problem suddenly starts occurring, it may be due to a recent browser update.
Tip: If you have another PC, turn off auto-updates for the browser(s) so that you have an old browser you can use to test your application against.

Other checks to carry out on the browser side include the following:

1. Check the AUI tree to verify that node and properties of the user interface screen you are currently working on are loaded properly and that the widgets attributes are correctly set. See Configure development environment on page 156

2. Use browser developer tools to inspect elements of a GWC-JS application interface to identify the widgets not rendering correctly. Browser developer tools are accessed by pressing the F12 key. Check:
   - that the resource files (.css, .js, and so on) are accessible and not displaying a 404 error message.
   - that the HTML is properly generated. For example, check whether the color for the edit is set or is missing in the generated page.

GWC-JS customization
Run your program without any GWC-JS customization applied, that is run it using the default styling. If the problem does not occur with the default rendering, then the problem most likely is due to your GWC-JS customization.

Your application code
Run the demo application delivered by the Genero Application Server and in the Topic tree of the demo directory, navigate to User Interface. If a particular widget/container renders correctly in the demo but not in your program, then this is likely due to your 4gl code.

How to implement delegation
The GAS is able to delegate the start of a web application or a web service to another Genero REST service in order to perform some controls before granting access and starting the application.

- How delegation works on page 113
- Configure delegation for application or service on page 115
- From the user agent to the REST service on page 116
- From the REST service to the proxy on page 118
- REST service example on page 119
- Delegation use cases on page 122

How delegation works
This section gives details about the delegation process.

Following steps are performed when an delegation occurs:
1. An application or web service start is requested. The type of request is defined by the /ua/r, /wa/r, /ja/r, or /ws/r path segments in the URI.

2. The Genero Application Server dispatcher passes the request to the REST service identified as the delegation service. The delegation service is specified in the application's configuration. The delegation service is written in Genero and managed by the Genero Application Server as a standard REST Web service. The delegation service should reside on the same GAS as the application.

3. The REST service instructs the Genero Application Server to either:
   - Refuses the start of the application or service.
   - Allows the application to start. The delegate service is able to add some environment variables to give additional information to the allowed application.
   - Allows for a service to any request forwarded to it. All Web service requests are in /ws/r so they all go to the delegate service.

The delegation REST service refuses the start of the application or service

In this scenario, the delegation REST service refuses the start of the application or service. The REST service communicates with the user agent in HTTP using the com.HTTPServiceRequest object. For example, it could return an XHTML form asking for a user name and password to grant access to the application or service.

![Diagram](image.png)

Figure 28: REST service communicates with User Agent but does not start the application or service

In this figure:
- (1) Start a new application or service of the form /wa/r, /ja/r or /ws/r.
- (D) Delegate application or service start to the service described in the configuration file (xcf).
- (2) Forward the request to REST service for delegation process
- (A) REST program responds directly to the User Agent (with a login page, for instance).
- (3) (4) Response is sent back to the User Agent.

The delegation REST service allows the start of the application or service

In this scenario, the delegation REST service allows the start the application or service as if launched directly from the user agent.
In this figure:

- (1) Start a new application or service of the form /wa/r, /ja/r, or /ws/r.
- (D) Delegate application or service start to the service described in the configuration file (xcf).
- (2) Forward request to REST service for delegation process
- (B) Genero REST program allows to start the proxy (login and password are correct, for instance).
- (3) (4) Response is sent back to the User Agent.
- (5) REST program response with HTTP code 307 and description string.
- (C) Dispatcher detects REST command and starts the proxy
- (6) Dispatcher forwards response from REST program to proxy
- (7) (8) Any request is forwarded to the proxy without going to the REST service (excepted for GWS that starts a new delegation process).

To allow the application to start, the Genero REST service returns a specific HTTP code and description to the dispatcher using the com.HTTPServiceRequest object. When the dispatcher gets such an HTTP response from the REST service, it starts a new proxy and forwards the request to it, as if no delegation had taken place.

**Configure delegation for application or service**

To delegate the start of an application or service to a Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

The DELEGATE element requires an attribute called service. For the service attribute, specify the Genero REST service that will be in charge of all delegated requests for the application or service. The REST service must be correctly configured in the Genero Application Server.
You can define optional parameters for the REST service to be sent each time a starting request is received. No validation is made for these optional parameters, the REST service must check them and return an error when necessary.

Note: A starting request is a URL with /r. When you see an application URI with /sua, the application has been validated and the delegation REST service is no longer involved. For more information see Application URLs on page 51.

Delegate configuration example

```xml
<EXECUTION>
  <PATH>${res.path}</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="MyGroup/MyDelegateService">
    <AnyParameter>MyFirstParameter</AnyParameter>
    <Other>MySecondParameter</Other>
  </DELEGATE>
</EXECUTION>
```

Parameters defined in the DELEGATE configuration, such as AnyParameter and Other in this example, are transmitted to the REST service. See Passing parameters to the REST service on page 116 for details on receiving the parameters in the REST service.

From the user agent to the REST service

Each request of the form /ua/r, /ja/r and /ws/r coming from the user agent are delegated to the Genero REST service via its entry point.

REST service entry point

When a /ua/r, /ja/r or /ws/r request is delegated to the REST service, the dispatcher appends the string /Delegate to the service URL in order to distinguish a dispatcher delegation from any other standard REST request. In other words, if an application has delegation, the REST service is called with a /Delegate appended in the URL.

```java
IMPORT com
DEFINE req com.HTTPServiceRequest
...
LET req = com.WebServiceEngine.GetHttpServiceRequest(-1)
LET url = req.getUrl()
IF url.getIndexOf("/ws/r/RestGroup/RestService/Delegate",1)>1 THEN
  CALL HandleDelegation(req)
ELSE
  CALL HandleStandardService(req)
END IF
...
```

Passing parameters to the REST service

If parameters are defined in the DELEGATE configuration, they will be transmitted to the Genero REST service at each /wa/r, /ja/r and /ws/r request as HTTP headers.

There is one HTTP header per parameter set in the configuration, and it is of the form X-FourJs-Environment-Parameter-XXX where XXX is the parameter name and the parameter value is the HTTP header value.

REST sample:

```java
IMPORT com
DEFINE req com.HTTPServiceRequest
```
**Configuring the Genero Application Server**

... 

```javascript
LET param1 = req.getRequestHeader("X-FourJs-Environment-Parameter-AnyParameter")
DISPLAY param1 # Displays MyFirstParameter
LET param2 = req.getRequestHeader("X-FourJs-Environment-Parameter-Other")
DISPLAY param2 # Displays MySecondParameter
```

The sample is based on this configuration:

```xml
<EXECUTION>
  <PATH>$(res.path)</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="MyGroup/MyDelegateService">
    <AnyParameter>MyFirstParameter</AnyParameter>
    <Other>MySecondParameter</Other>
  </DELEGATE>
</EXECUTION>
```

**Passing the user agent URL to the REST service**

When a `/wa/r`, `/ja/r` or `/ws/r` request is delegated to a REST service, the original URL is transmitted to the service in the URL query string of the request.

For example, if the user types the following original URL in a browser:

```
http://host:port/ua/r/MyGrp/MyApp?P1=1&P2=2
```

The resulting URL passed to the delegation service will be:

```
```

To process the request in the delegation service, if the service handles only delegation, you can directly extract the original URL with the `readFormEncodedRequest()` method:

```java
IMPORT com
...
DEFINE req com.HTTPServiceRequest
DEFINE original STRING
DEFINE url STRING
DEFINE query STRING
...
LET original = req.readFormEncodedRequest(false)
CALL SplitUrl(original) RETURNING url, query
DISPLAY url # http://host:port/ua/r/MyGrp/MyApp
DISPLAY query # P=1&P=2
...
```

If the service must also handle non-delegated requests, use the `getUrl()` method to retrieve the REST operation to perform. If the REST URL contains the "/Delegate" string, it is a delegation request and you need to extract the original URL after the ? character. You can then check the original URL the user agent wants to access, and extract potential parameters in its query string:

```java
IMPORT com
...
DEFINE req com.HTTPServiceRequest
DEFINE url STRING
DEFINE rest_url STRING
DEFINE original STRING
DEFINE orig_url STRING
```
DEFINE query STRING
...
LET url = req.getURL()
CALL SplitUrl(url) RETURNING rest_url, original
CASE
   WHEN rest_url.getIndexOf("/Delegate",1)
      CALL SplitUrl(original) RETURNING orig_url, query
      DISPLAY orig_url # http://host:port/ua/r/MyGrp/MyApp
      DISPLAY query # P=1&P=2
   WHEN rest_url.getIndexOf("/GetCurstomerInfo",1)
      # Handle regular REST request
      ...
   OTHERWISE
      CALL req.sendTextResponse(400, NULL, "Invalid REST request")
END CASE
...

From the REST service to the proxy
The delegation REST service must notify the dispatcher when it approves the start of an application or service.

Approve the proxy start
To approve the start of an application or a service proxy or the service forwarding (because if it is a web service using delegation, the service may be already started), the Genero REST service must return the following HTTP code and description: The HTTP return code must be 307 and the description must be the string _GENERO_INTERNAL_DELEGATE_.

Returning this HTTP code and description notifies the dispatcher to start the application or service proxy as the response to the current user-agent request.

IMPORT com
DEFINE req com.HTTPServiceRequest
...
CALL req.sendResponse(307,"_GENERO_INTERNAL_DELEGATE_")
...

Note: You can return a body from the REST service that is then transmitted to the proxy if original incoming request was POST or PUT, otherwise body is skipped.

Passing parameters to the proxy
When you need to pass additional parameters to a starting proxy, you can do it via environment variables that are then set in a gwc or gdc proxy environment before the dispatcher starts it. Each parameter must be set in the HTTP header response when you specify the 307 HTTP return code. The HTTP header name must be of the form X-FourJs-Environment-XXX where XXX is the name of the variable to pass to the proxy and as HTTP header value, the value of the environment variable. Then, with a call to the Genero function fgl_getenv(), you can retrieve them in your Genero program.

Passing parameters to a proxy is not possible for Web services requests, as the proxy is already started. The only way to send the environment with a Web services request is through the HTTP header.

REST sample:

IMPORT com
DEFINE req com.HTTPServiceRequest
...
CALL req.setResponseHeader("X-FourJs-Environment-Hello","World")
CALL req.setResponseHeader("X-FourJs-Environment-Name","Georges")
CALL req.sendResponse(307,"_GENERO_INTERNAL_DELEGATE_")
REST service example

In this example, the REST service returns an HTTP error code 404 to the browser until the query string contains the string ByPass.

When the GWC application is started, the parameter ACCESS is also set and can be retrieved in the Genero program with fgl_getenv("ACCESS").

```
IMPORT COM
IMPORT XML

PRIVATE CONSTANT C_BASEURL = "/ws/r/qa-test/delegateService/
PRIVATE CONSTANT C_X_FOURJS_ENVIRONEMENT_ = "X-FourJs-Environment-"
PRIVATE CONSTANT C_X_FOURJS_ENVIRONEMENT_PARAMETER_ = "X-FourJs-Environment-Parameter-"
PRIVATE CONSTANT C_GENERO_INTERNAL_DELEGATE = "_GENERO_INTERNAL_DELEGATE_"

MAIN
DEFINE  req   com.HttpServiceRequest
DEFINE  methd STRING
DEFINE  url   STRING
DEFINE  path  STRING
DEFINE  query STRING
DEFINE  ind   INTEGER
DEFINE  tmp   STRING

CALL com.WebServiceEngine.Start()

WHILE TRUE
TRY
  LET req = com.WebServiceEngine.GetHttpServiceRequest(-1)
  IF req IS NULL THEN
    EXIT WHILE
  ELSE
    LET url = req.getUrl()
    DISPLAY "URL : ",url
    LET methd = req.getMethod()
    CALL SplitUrl(url) RETURNING path, query
    DISPLAY "Incoming request: ",methd," path=" ,path," query=" ,query
    LET ind = path.indexOf(C_BASEURL,1)
    IF ind<1 THEN
      CALL req.sendResponse(400,"Invalid request")
    ELSE
      CALL DispatchService(req,path.subString(ind+C_BASEURL.getLength(),path.getLength()),query)
    END IF
  END IF
CATCH
  EXIT WHILE
END TRY
END WHILE
```
FUNCTION DispatchService(req,path,query)
    DEFINE req com.HttpServiceRequest
    DEFINE path STRING
    DEFINE query STRING
    DEFINE ind INTEGER
    LET ind = path.indexOf("/",1)
    IF ind>0 THEN
        CALL req.sendResponse(400,"invalid path")
    ELSE
        CASE path
            WHEN "Delegate" # Handle a dispatcher delegate service
                CALL DelegateWA(req,query)
            OTHERWISE
                CALL req.sendResponse(400,"unknown service "||path||".");
        END CASE
    END IF
END FUNCTION

# Delegate WA service
# If browser URL doesn't contain 'ByPass' in query string then return a 404 error,
# otherwise start GWC application

FUNCTION DelegateWA(req,query)
    DEFINE req com.HttpServiceRequest
    DEFINE query STRING
    DEFINE url STRING
    DEFINE queryString STRING
    DEFINE parameter STRING
    DEFINE ind INTEGER
    DEFINE ByPass BOOLEAN
    DEFINE q DYNAMIC ARRAY OF RECORD
        qname STRING,
        qvalue STRING
    END RECORD
    # Read requests
    LET query = req.readFormEncodedRequest(false)
    IF query IS NULL THEN
        CALL req.sendResponse(400,"no query string")
    ELSE
        LET ByPass = FALSE
        CALL SplitUrl(query) RETURNING url,queryString
        CALL ParseQueryString(queryString) RETURNING q
        # Check if user-agent query string has a ByPass string ?
        FOR ind=1 TO q.getLength()
            IF q[ind].qname="ByPass" THEN
                LET ByPass = TRUE
            END IF
        END FOR
        IF NOT ByPass THEN
            # return error
            CALL req.sendResponse(404,"ByPass is missing")
        ELSE
            # Set parameter for GWC application via environment variable:
            ACCESS=OK
            CALL req.setResponseHeader(C_X_FOURJS_ENVIRONEMENT_||"ACCESS","OK")
            # Start application with HTTP code 307
            CALL req.sendResponse(307,C_GENERO_INTERNAL_DELEGATE)
        END IF
    END IF
END IF
FUNCTION ParseQueryString(str)
DEFINE str STRING
DEFINE tkz base.StringTokenizer
DEFINE token STRING
DEFINE ind INTEGER
DEFINE ret DYNAMIC ARRAY OF RECORD
  qname STRING,
  qvalue STRING
END RECORD
INITIALIZE ret TO NULL
LET tkz = base.StringTokenizer.create(str,"&")
WHILE (tkz.hasMoreTokens())
  LET token = tkz.nextToken()
  CALL ret.appendElement()
  LET ind = ind + 1
  CALL ExtractKeyValueFromQuery(token) RETURNING
  ret[ind].qname,ret[ind].qvalue
END WHILE
RETURN ret
END FUNCTION

FUNCTION ExtractKeyValueFromQuery(str)
DEFINE str  STRING
DEFINE ind  INTEGER
LET ind = str.getIndexOf("=",1)
IF ind>1 THEN
  RETURN str.substring(1,ind-1),str.substring(ind+1,str.getLength())
ELSE
  RETURN str,NULL
END IF
END FUNCTION

FUNCTION SplitUrl(url)
DEFINE  url   STRING
DEFINE  ind   INTEGER
DEFINE  query STRING
DEFINE  path  STRING
LET ind = url.getIndexOf("?",1)
IF ind>1 THEN
  LET query = url.substring(ind+1,url.getLength())
  LET path = url.substring(1,ind-1)
ELSE
  LET query = NULL
  LET path = url.substring(1,url.getLength())
END IF
RETURN path, query
END FUNCTION
Delegation use cases

Three examples of possible uses for the delegation mechanism.

Simple local authentication / authorization mechanism

You can develop a simple delegation service to authenticate and authorize users to have access to one or several applications on the Genero Application Server.

The delegation service will respond to the request with an HTML form, asking for a user name and password. In this case, HTTPS is required; otherwise the login and password will be sent in clear text.

A request with user name and password as parameters will be processed by the delegation service. The service will check for the user name and password in its database. If the user name and password are correct, a digest authentication will be created, stored in the database, and sent back to the user-agent in a cookie. The delegation service will instruct the user agent to delegate on the same URL (so the user agent will use its newly set cookie).

A request with a cookie will be processed by the delegation service. The cookie will be checked in the database. The corresponding user id, as well as the user role (administrator, user, guest, and so on), will be set as application parameters and the Genero Application Server will be instructed to allow the launch of the application.

A simple local authentication / authorization mechanism is provided in the Genero Application Server tutorials located at FGLDIR/web_utilities/services/simplesso.

Authentication / authorization Single sign-on (SSO) mechanism

You can develop a delegation service to authenticate and authorize users to access one or more applications on the Genero Application Server based on standard SSO services such as OpenID Connect (see OpenID Connect SSO on page 123) or SAML (see SAML SSO on page 136).

The delegation service responds to a simple request for delegation to the SSO service, with reference to the requested application.

A request with a cookie will be processed by the delegation service. The cookie will be checked by the SSO service (by means specific to the SSO protocol). The corresponding user id and user role (as allowed by the SSO protocol) will be set as application parameters and the Genero Application Server will be instructed to allow the launch of the application.

Samples for implementing OpenID Connect and SAML authentication and authorization services are provided in $FGLDIR/web_utilities/services, ready for you to use.

Monitoring or logging requests for a Genero Web service

You can develop a simple delegation service to monitor and log all requests to a given service. Each request can be logged in a dedicated database by the delegation service. The Genero Application Server can then be instructed to pass the request to the GWSProxy. The delegation for Web services is called each time a request is sent to that service.

Note: For applications, logging is only performed at application start up.

How to implement Single sign-on (SSO)

You can add Single sign-on (SSO) to your applications to allow users to enter one name and password in order to access multiple applications. Genero Application Server supports different kinds of Single sign-on.

- OpenID Connect SSO on page 123
- OpenID SSO on page 128
- SAML SSO on page 136
OpenID Connect is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under $FGLDIR/web_utilities/services/openid-connect/.

OpenID Connect providers include Google and Microsoft. To learn more about OpenID Connect, see the OpenID Connect web site.

The Genero OpenID Connect solution is supported on the GAS delegation mechanism. See What is delegation? on page 38.

OpenID Connect implementation creates a circle of trust between the Genero Application Server and an OpenID Connect provider. SSO implementation may vary depending on the IdP, but typically it consists of the following:

- Getting OAuth2 (public and share secret ids), (see Quick start: Set up OpenID Connect in the GAS on page 123)
- Providing the redirect URL of the GAS to the IdP, (see Add OpenID Connect SSO to Genero Web application on page 125)

**Quick start: Set up OpenID Connect in the GAS**

Follow these steps to quickly set up OpenID Connect for your Genero Application Server and Genero Web Client applications.

Before you begin, you must Configure GAS for OpenID Connect SSO on page 124.

In this quick start, you add OpenID Connect Single sign-on (SSO) to a Genero Web Client application, then execute the application with SSO.

1. Add OpenID Connect SSO to a Genero Web Client application requiring SSO.
   a) Add the **DELEGATE** element to all Genero Web Client applications requiring SSO

   The first three parameters are mandatory:
   - **IDP** : the provider of the IdP account (e.g. https://accounts.google.com)
   - **CLIENT_PUBLIC_ID** : the OAuth2 public ID provided by the IdP
   - **CLIENT_SECRET_ID** : the OAuth2 shared secret ID provided by the IdP
   - **SCOPE** : (optional) the OpenID Connect attributes you want to get from the user at time of authentication (e.g. email, phone, address).

   ```xml
   <APPLICATION Parent="defaultgwc">
   <EXECUTION>
   <PATH>$(res.path.mypath)/myapplication</PATH>
   <MODULE> myapp.42r</MODULE>
   <DELEGATE service="services/OpenIDConnectServiceProvider">
   <IDP>https://accounts.google.com</IDP>
   <SCOPE>email</SCOPE>
   ```
2. Execute a Genero Web Client application with SSO.
   a) Start your browser and enter the application URL.
      You are prompted to enter your OpenID Connect credentials.
   b) Click the sign in button.
      Your browser is redirected to the Identity Provider (IdP).
   c) Enter your credentials.
      If your credentials are valid, your browser is redirected to the Genero Web Client application. The application can then get OpenID Connect user information through environment variables such as OIDC_SUB.

   **Note:** The fglrun process is executed in the context of the GAS operating system user.
   For example, when using Apache, the program process will run in the context of the Apache user.

   The next time you start the same application - or any application delivered by the same Genero Application Server - you will not be prompted for your credentials. The application will start and be authenticated by the same OpenID Connect user.

   **Tip:** Read all of the OpenID Connect topics in the Genero Application Server User Guide for details on features provided by OpenID Connect SSO support in the Genero Application Server; including attributes gathering or authorization control.

**Configure GAS for OpenID Connect SSO**

Follow these steps to configure the Genero Application Server for OpenID Connect Single sign-on (SSO).

1. Create an account with an OpenID Connect provider that will provide authentication services for you, e.g. see [Configure OpenID Connect identity on Google](#) on page 124.
   Through this one account, the IdP provides you with an authentication services that identifies to the GAS the users that access your application. Add the public and shared secret ids obtained from the IdP to your application's configuration files, see [Add OpenID Connect SSO to Genero Web application](#) on page 125.

2. If the Genero Application Server is located behind a proxy, configure the proxy in the OpenID Connect fglprofile file in $FGLDIR/web_utilities/services/openid-connect/res.
   Remove the comment and set the correct value for the entry called proxy.http.location and proxy.https.location.

3. Start your dispatcher (if not behind a web server).
   **Note:** Genero OpenId Connect service requires HTTPS communication with the IdP. If needed, you may have to configure SSL and CA authority in the fglprofile file. (see the [Genero Business Development Language User Guide](#) for details).

The Genero Application Server is ready to use OpenID Connect SSO to authenticate end users.

**Configure OpenID Connect identity on Google**

Follow these steps to configure an OpenID Connect Single sign-on (SSO) identity on Google.

1. Go to the Google developer console page [https://console.developers.google.com/](https://console.developers.google.com/)
2. Create a new project (or use an existing one)
3. From the project page select **Credentials** under **APIs and auth**
4. In the **Credentials** page, select **Create new Client ID**
   This will open a new window where you can create a client identity and select a product.
a) Choose Web Application as product
b) In the Authorized JavaScript origins field, specify your JavaScript hostname (e.g. https://host:port/gas)
c) In the Authorized redirect URIs fields, specify the URI redirection where the GAS is listening for the response (i.e. https://host:port/gas/ws/r/services/OpenIDConnectServiceProvider/oauth2callback)

The OAuth2 PUBLIC and SHARED SECRET IDs are displayed

Note: You will need to save these in your web service application configuration file.

You have now setup Google as your IdP for your web services to use OpenID Connect SSO.

Add OpenID Connect SSO to Genero Web application

Follow these steps to add OpenID Connect SSO to a Genero Web application.

This task must be performed in the .xcf application configuration file for the Genera Application Server.

Add <DELEGATE service="services/OpenIDConnectServiceProvider"> to the application configuration (.xcf) file.

Add the DELEGATE tag to all Genero Web Client applications requiring Single sign-on (SSO), plus the following 3 mandatory parameters:

• IDP : the IdP account (e.g. https://accounts.google.com)
• CLIENT_PUBLIC_ID : the OAuth2 public id from the IdP
• CLIENT_SECRET_ID : the OAuth2 shared secret id from the IdP
• SCOPE : (optional) the OpenID Connect attributes you want to get at authentication (e.g. email, phone, address)

<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
<EXECUTION>
  <PATH>$(res.path.qa)/applications/myapp</PATH>
  <MODULE>App.42r</MODULE>
  <DELEGATE service="services/OpenIDConnectServiceProvider" >
    <IDP>https://accounts.google.com</IDP>
    <SCOPE>email</SCOPE>
    <CLIENT_PUBLIC_ID>XXXXXXXX.apps.googleusercontent.com</CLIENT_PUBLIC_ID>
    <CLIENT_SECRET_ID>XXXXXXXX-XXXXXX</CLIENT_SECRET_ID>
  </DELEGATE>
</EXECUTION>
</APPLICATION>

With the above configuration and default GAS configuration, the delegation will point to the $FGLDIR/web_utilities/services/OpenIDConnectServiceProvider.xcf file.

For more information about the DELEGATE configuration element, see DELEGATE on page 365.

The Genero Application Server will handle the OpenID Connect protocol and start the Genero web application only when the user has been authenticated, otherwise an HTML error page is returned.

Retrieve the OpenID Connect user identifier

Follow these steps to retrieve the OpenID Connect Single sign-on (SSO) user identifier in your Genero application.

To retrieve the OpenID Connect user identifier, add this code to your Genero application:
Once the user has been successfully authenticate and before starting the proxy, the OpenID Connect service sets all attributes coming from the IdP with the prefix `OIDC_` (OIDC stands for OpenID Connect).

**Note:** Even if there are no attributes being sent by the IdP (maybe because the user has not allowed the Google console API to send them), the `OIDC_SUB` attribute will always be available. This attribute is an opaque value representing the user subject at IdP.

For example, if you set `email` in the `SCOPE` parameter of your application configuration (see Quick start: Set up OpenID Connect in the GAS on page 123), you will have an attribute called `OIDC_email` set that is then retrievable with the instruction in your application.

```
LET userEmail = fgl_getenv("OIDC_email")
```

### Authorization and OpenID Connect SSO

Authorize whether a user already authenticated by OpenID Connect SSO can access a Genero application.

The JGAS Genero Application Server must be configured for OpenID Connect Single sign-on (SSO). See Configure GAS for OpenID Connect SSO on page 124.

With the Genero OpenID Connect implementation, you can add an external program to determine whether an already authenticated user can access a Genero Web application. For example, you may need to restricted access to certain applications or to certain specific email addresses; otherwise any user authenticated by an OpenID Connect provider can access your Genero Web application. It is therefore recommended that you add an authorization program to filter access to your applications.

This external program can be written in Genero or in another programming language. The application `AccessProgram.4gl` in `$FGLDIR/web_utilities/services/openid-connect` provides an example of an authorization application written in Genero.

**Note:** The external program must be deployed beside the `OpenIDConnectServer.42r` program, because it will be executed by that service program. This is by default under `$FGLDIR/web_utilities/services/openid-connect/bin`.

The authorization program expects two mandatory arguments and the list of OpenID Connect attributes received from the OpenID Connect provider:

```
access-program oidc-userid app-xcf-path [ attribute value [...]]
```

- The first argument is the OpenID Connect identifier (typically an opaque value returned by the IdP)
- The second argument is the application path.
- Next arguments are optional and define OpenID attributes/value pairs.

Example with a Genero authorization program:

```
fglrun AccessProgram 
 "101516043183449889392" \
 "qa-test/application" \
 "fullname" "genero test" \
 "email" "genero@4js.com" \
 "country" "France"
```

The external authorization program is specified in the application configuration element by adding an `AUTHORIZATION` element in the `DELEGATE` element.

1. Add an `AUTHORIZATION` element as a child of the `DELEGATE` element in the application configuration (xcf) file.
2. Within the **AUTHORIZATION** element, specify the command to execute the external authorization program.

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
   <EXECUTION>
     <PATH>$(res.path.qa)/applications/myapp</PATH>
     <MODULE>App.42r</MODULE>
     <DELEGATE service="services/OpenIDConnectServiceProvider">
       <IDP>https://accounts.google.com</IDP>
       <SCOPE>email</SCOPE>
       <CLIENT_PUBLIC_ID>XXXXXXXX.apps.googleusercontent.com</CLIENT_PUBLIC_ID>
       <CLIENT_SECRET_ID>XXXXXX-XXXXXX</CLIENT_SECRET_ID>
       <AUTHORIZATION>fglrn AccessProgram</AUTHORIZATION>
     </DELEGATE>
   </EXECUTION>
</APPLICATION>
```

The authorization program will be called before access to the Web application is granted. If the authorization program exits with an error code of zero (0), then access is granted for the user. Any exit code other than zero indicates access for the user is denied. In the latter case, the end user will be warned with an error page in the web browser, generated by the OpenID Connect service.

**Genero OpenID Connect FGLPROFILE**

Genero OpenID Connect implementation uses its own FGLPROFILE file.

The Genero OpenID Connect Single sign-on (SSO) implementation uses its own FGLPROFILE file in `$FGLDIR/web_utilities/services/openid-connect/res`.

This file can be modified to define the following features:

- ODI database driver definition.
- HTTP and HTTPS proxy configuration.
- X509 and SSL keys for handling HTTPS connection (if needed).

When to modify this file:

- If you want a database engine other than SQLite.
- If your Genero Application Server installation requires proxy configuration to connect to an OpenID Connect provider.

**Genero OpenID Connect log file**

The Genero OpenID Connect implementation produces a log file that helps to identify issues.

The log file of the Genero OpenID Connect Single sign-on (SSO) implementation is called `OIDC.log` and is located in `$(res.appdata.path)/log`. This log file contains all incoming and outgoing requests. It can help to debug OpenID Connect issues.

You can specify the level of detail recorded to the log with the `-debug` category option of the OpenID Connect server program. There are two categories that can be logged individually or together:

- **MSG** - Standard information regarding access and errors. By default, only access and error information are logged.
- **DEBUG** - Traces the entire process of single sign-on (SSO).

To add debugging information to the OIDC.log, modify `OpenIDConnectServiceProvider.xcf` to include the `-debug DEBUG` option as first argument in the command defined by the `MODULE` element:
Configuring the Genero Application Server | 128

Note: Logging is based on the Genero `ERRORLOG()` function. As several instances of the same OpenID Connect server can write into the log file, the PID of the server is written to the log file as well.

OpenID SSO

OpenID is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under `$FGLDIR/web_utilities/services/openid`.

OpenID providers include Google and Yahoo. To learn more about OpenID, see the OpenID web site.

Note: Genero implements only version 2.0 of the OpenID specification and is only intended for Genero Web Client applications.

The Genero OpenID solution is supported on the GAS delegation mechanism. See What is delegation? on page 38.

Important: Genero OpenID service uses an SQLite database to store authentication data. If you do not configure another database, Genero OpenID service will use the `openid.db` SQLite database located in `$FGLDIR/web_utilities/services/openid/bin`. The OpenID REST service will execute in the context of the Genero Application Server. This user must have write permission on the `openid.db` file and the parent `bin` directory, otherwise the service will fail to insert data into the SQLite database.

- Quick start: Set up OpenID in the GAS on page 129
- Configure GAS for OpenID SSO on page 129
- Add OpenID SSO to a Genero Web application on page 130
- Retrieve the OpenID user identifier on page 130
- Retrieve identity attributes with OpenID SSO on page 130
- Authorization and OpenID SSO on page 131
- Execute an application with OpenID SSO on page 132
- Distinct user authentication per application on page 133
- Genero OpenID configuration file on page 133
- Specify a database to store OpenID data on page 134
- Genero OpenID FGLPROFILE on page 135
- Genero OpenID log file on page 135
Quick start: Set up OpenID in the GAS
Follow these steps to quickly set up OpenID Single sign-on (SSO) for your Genero Application Server and Genero Web Client applications.

Before you begin, you must Configure GAS for OpenID SSO on page 129.

In this quick start, you add OpenID Single sign-on (SSO) to a Genero Web Client application, then execute the application with SSO.

1. Add OpenID SSO to a Genero Web Client application.
   a) Add the `DELEGATE` element to all Genero Web Client applications requiring SSO.

```
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>${res.path.mypath}/myapplication</PATH>
    <MODULE>myapp.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider" />
  </EXECUTION>
</APPLICATION>
```

   b) Add a `PROVIDER` element to indicate which identity provider to use for the application.

```
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>${res.path.mypath}/myapplication</PATH>
    <MODULE>myapp.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider">
      <PROVIDER>google.com</PROVIDER>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

If a provider is not defined, a page with the list of available ID providers is displayed.

2. Execute a Genero Web Client application with SSO.
   a) Start your browser and enter the application URL.
       You are prompted to enter your openid.
   b) Click the **signin** button.
       Your browser is redirected to the OpenID provider.
   c) Enter your credentials.

   If your credentials are valid, your browser is redirected to the Genero Web Client application.
   The application can then get OpenID user information through environment variables such as `OPENID_CLAIMED_ID`.

   **Note:** The fglrun process is executed in the context of the GAS operating system user. For example, when using apache, the program process with run in the context of the apache user.

The next time you start the same application - or any application delivered by the same Genero Application Server - you will not be prompted for your credentials. The application will start and get the same OpenID user information.

**Tip:** Read all of the OpenID topics in the Genero Application Server User Guide for details on features provided by OpenID SSO support in the Genero Application Server, to include attributes gathering or authorization control.

Configure GAS for OpenID SSO
Follow these steps to configure the Genero Application Server for OpenID Single sign-on (SSO).

1. Create one or more OpenID users on an OpenID provider.
2. If the Genero Application Server is located behind a proxy, configure the proxy in the OpenID
fglprofile in $FGLDIR/web_utilities/services/openid/res.
   Remove the comment and set the correct value for the entry called proxy.http.location and
proxy.https.location.

3. Start your dispatcher (if not behind a web server).

The Genero Application Server is ready to use OpenID SSO to authenticate end users.

Add OpenID SSO to a Genero Web application

Follow these steps to add OpenID Single sign-on (SSO) to a Genero Web application.

This task must be performed in the .xcf application configuration file for the Genera Application Server.

Add <DELEGATE service="services/OpenIDServiceProvider"/> to the application
configuration (.xcf) file.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmni:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider"/>
  </EXECUTION>
</APPLICATION>
```

With the above configuration and default GAS configuration, the delegation will point to the $FGLDIR/
web_utilities/services/OpenIDServiceProvider.xcf file.

For more information about the DELEGATE configuration element, see DELEGATE on page 365.

The Genero Application Server will handle the OpenID protocol and start the Genero web application only
when the user has been authenticated, otherwise an HTML error page is returned.

Retrieve the OpenID user identifier

Follow these steps to retrieve the OpenID user identifier in your Genero application.

To retrieve your OpenID Single sign-on (SSO) user identifier, add this code to your Genero application:

```
LET id = fgl_getenv("OPENIDCLAIMED_ID")
```

More OpenID user information can be retrieved, for details see Retrieve identity attributes with OpenID
SSO on page 130.

Retrieve identity attributes with OpenID SSO

Follow these steps to retrieve additional attributes about user identity when authenticating to an OpenID
provider.

As a prerequisite, identify which identity attributes are supported by the OpenID provider (i.e. the identity
provider - IDP). The Genero OpenID Single sign-on (SSO) implementation automatically detects which
attribute exchange protocol is supported.

There are two kinds of attribute exchange protocols:

- OpenID Simple Registration Extension (default for Genero)
  - Specifications
  - List of attributes
- OpenID Attribute Exchange (used if default is not available)
  - Specifications
• List of attributes are specific to an OpenID provider and must be mapped to a single name in the configuration file of the Genero OpenID implementation.

Complete this procedure to retrieve additional attributes about your identity when authenticating to an OpenID provider. For example, you can retrieve the email, full name, or country of the user.

1. Add an ATTRIBUTES element as a child of the OpenID DELEGATE element in the application configuration (xcf) file. Provide a comma-separated list of OpenID attributes within the ATTRIBUTES element.

In this example, the email, fullname, and country openid attributes are specified.

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider">
      <ATTRIBUTES>email,fullname,country</ATTRIBUTES>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

2. To retrieve the OpenID attributes in your Genero application, add a fgl_getenv() call for each attribute specified in the XCF file with a prefix of OPENID_.

```plaintext
LET email = fgl_getenv("OPENID_email")
LET fullname = fgl_getenv("OPENID_fullname")
LET country = fgl_getenv("OPENID_country")
```

The Genero application retrieves the requested identity attributes.

**Authorization and OpenID SSO**

Authorize whether an user already authenticated by OpenID SSO can access a Genero application.

The GAS must be configured for OpenID Single sign-on (SSO). See Configure GAS for OpenID SSO on page 129.

With the Genero OpenID implementation, you can add an external program to determine whether an already authenticated user can access a Genero Web application.

This external program can be written in Genero or in another programming language.

The authorization program expects two mandatory arguments and the list of OpenID attributes received from the OpenID provider:

```plaintext
access-program openid-userid app-xcf-path [ attribute value [...] ]
```

• The first argument is the OpenID identifier.
• The second argument is the application path.
• Next arguments are optional and define OpenID attributes/value pairs.

Example with a Genero authorization program:

```plaintext
fglrun AccessProgram
"genero-user.pip.verisignlabs.com" \ "qa-test/application" \ "fullname" "genero test" \ "email" "genero@4js.com" \
```
The application AccessProgram.4gl in $FGLDIR/web_utilities/services/openid provides an example of an authorization application written in Genero.

The external program is specified in the application configuration element by adding a AUTHORIZATION element in the DELEGATE element.

If the AUTHORIZATION element is not defined, any user authenticated by an OpenID provider can access the Genero Web application. It is recommended that you add an authorization program to filter the access to your applications.

**Note:** The external program must be deployed beside the OpenIDServer.42r program, because it will be executed by that service program. This is by default under $FGLDIR/web_utilities/services/openid/bin.

1. Add an AUTHORIZATION element as a child of the OpenID DELEGATE element in the application configuration (xcf) file.
2. Within the AUTHORIZATION element, specify the command to execute the external authorization program.

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider">
      <ATTRIBUTES>email,fullname,country</ATTRIBUTES>
      <AUTHORIZATION>fglrun AccessProgram</AUTHORIZATION>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

The authorization program will be called before access to the Web application is granted. If the authorization program exits with an error code of zero (0), then access is granted for the user. Any exit code other than zero indicates access for the user is denied. In the last case, the end user will be warned with a error page in the web browser, generated by the OpenID service.

**Execute an application with OpenID SSO**
Follow these steps to execute a Genero Web Client application and authenticate user with OpenID Single sign-on (SSO).

1. Open a browser and enter the application URL. You will be prompted to enter your openid in the form xxx.openid.com.
2. Click the signin button. Your browser redirects to the OpenID provider.
3. Enter your credentials.

If the credentials are valid, your browser redirects to your Genero Web Client application that starts and runs as the entered OpenID user.

When you next start the same application, or any application served from the same Genero Application Server, you will not be prompted for your credentials. The Genero Web Client application will start and authenticate with the same OpenID user.
Distinct user authentication per application
Use the realm parameter to define specific domains for a set of applications.

By default, the OpenID Connect Single sign-on (SSO) service is launched with a realm parameter set to auto. All applications using this OpenID service are authenticated by the OpenID provider as coming from the same domain, and thus only require one authentication process for all of them.

The realm parameter can be changed from auto to an exact domain name (such as www.4js.com:6394/gas):

```<EXECUTION>
  ...
  <PATH>${res.path.fgldir.services}/openid/bin</PATH>
  <MODULE>OpenIDServer -realm www.mycompany.com:6394/gas -logPath "${res.appdata.path}"
  </MODULE>
  ...
</EXECUTION>
```

When using an exact domain name, ensure that all URLs for accessing your Genero Web applications have that form, otherwise you will get an OpenID error message.

To force the OpenID authentication login for any application executed by the GAS, remove the realm parameter from the $FGLDIR/web_utilities/services/OpenIDServiceProvider.xcf file. The end user will then be requested for credentials for any single application, even if they use the same OpenID service.

**Genero OpenID configuration file**
Specify OpenID provider constraints and mappings in the configuration file.

The Genero OpenID Single sign-on (SSO) implementation provides a configuration file named configuration.xml in $FGLDIR/web_utilities/services/openid/res.

The OpenID configuration file allows you to:

- Specify OpenID provider constraints about unsupported features (encryption and signature methods)
- Map the OpenID attributes URL to a single name if OpenID Attribute Exchange protocol is required.

**Server constraints**
Some OpenID providers do not support all encryption and signature methods recommended in the specification. To bypass this issue, you can specify the supported method for each individual OpenID provider. By default, Genero OpenID implementation supports the strongest one.

The encryption method defines how the Genero OpenID implementation and the OpenID provider will exchange the signature key. Supported methods are:

- **no-encryption**
  Requires HTTPS as communication channel between the OpenID provider and the Genero implementation. SSL certificate and keys must be specified in the fglprofile of the OpenID service ($FGLDIR/web_utilities/services/openid/res/fglprofile).

- **DH-SHA**
  Default. No specific communication channel required. The signature key is exchanged using a public/private Diffie-Hellman key agreement method fully secured.

The signature method defines how the Genero OpenID implementation validates what comes from the OpenID provider. Supported methods are:
**direct**

Requires HTTPS as communication channel between the OpenID provider and the Genero implementation in order to validate an authentication. Each authentication process requires an additional connection to the OpenID provider.

**HMAC-SHA1**

Default. No specific communication channel required. The Genero OpenID implementation can validate the authentication without any additional request to the OpenID provider.

Each server has a *secured attribute* that ensures that if a combination of encryption and signature method is not fully secured, the authentication process fails with a specific message in the log file.

If you set this attribute to false, the authentication process would not be fully secured as keys are exchanged between the OpenID provider and the Genero implementation on unsecured channels and thus vulnerable to malicious attacks.

**OpenID Attribute Exchange mapping**

If an OpenID provider uses the OpenID Attribute Exchange protocol, the configuration file must define an URL for each kind of attributes the OpenID provider supports.

This example illustrates how this is completed for the Google OpenID provider.

```xml
<Server secured="true" provider="google.com">
  <URL>www.google.com/accounts/o8/ud</URL>
  <Encryption>no-encryption</Encryption>
  <Signature>HMAC-SHA1</Signature>
  <AttributeProfile>http://openid.net/srv/ax/1.0</AttributeProfile>
  <Attribute name="email">http://axschema.org/contact/email</Attribute>
  <Attribute name="country">http://axschema.org/contact/country/home</Attribute>
  <Attribute name="firstname">http://axschema.org/namePerson/first</Attribute>
  <Attribute name="lastname">http://axschema.org/namePerson/last</Attribute>
  <Attribute name="language">http://axschema.org/pref/language</Attribute>
</Server>
```

The identity provider given in the application configuration file must match an identify provider configured in `$FGLDIR/services/openid/res/configuration.xml`.

The response from the ID provider should contain the provider identity ("google.com" in the above example).

The *AttributeProfile* element indicates the method to retrieve the attributes.

**Specify a database to store OpenID data**

Follow these steps to specify a database different from the default for the Genero OpenID implementation.

The implementation of Genero OpenID Single sign-on (SSO) requires a database to store OpenID data related to the protocol. By default, the database engine is SQLite and the database file is `$FGLDIR/web_utilities/services/openid/bin/openid.db`. This database is fully functional after installing the Genero Application Server.

1. Create a new or use an existing database, eventually on a dedicated machine, if several GAS servers are configured for load balancing. There must be a unique database, to centralize all OpenID authentication data.

2. In the file `server.4gl`, modify the functions `BDConnect()` and `DBDisconnect()` to handle and customize the database connection. Recompile the `server.4gl` source.
server.4gl is found in $FGLDIR/web_utilities/services/openid/src

3. Modify the FGLPROFILE file in $FGLDIR/web_utilities/services/openid/res to include the connection information for the database.

4. Create OpenID tables with the CreateDatabase.4gl program. Define the database permissions required to let the Genero Application Server modification the OpenID tables in the new database. CreateDatabase.4gl is found in $FGLDIR/web_utilities/services/openid/src

5. If needed, define the PATH (Windows®) or LD_LIBRARY_PATH (Linux®/UNIX™) environment variables in $FGLDIR/web_utilities/services/OpenIDServiceProvider.xcf with ENVIRONMENT_VARIABLE elements, in order to find the database client libraries required by Genero OpenID service.

   Note: If you use SQLite (by default), you do not need to add the path to the library since it is integrated in the ODI driver on most systems.

The alternate database is now used for the Genero OpenID implementation.

Genero OpenID FGLPROFILE
Genero OpenID implementation uses its own FGLPROFILE file.

The Genero OpenID Single sign-on (SSO) implementation uses its own FGLPROFILE file in $FGLDIR/web_utilities/services/openid/res/fglprofile.

This file can be modified to define the following features:
- ODI database driver definition.
- HTTP and HTTPS proxy configuration.
- X509 and SSL keys for handling a HTTPS connection.

When to modify this file:
- If you want a database engine other than SQLite.
- If your Genero Application Server installation requires proxy configuration to connect to an OpenID provider.
- If you need HTTPS specific settings to communicate with your OpenID provider.

Genero OpenID log file
The Genero OpenID implementation produces a log file that helps to identify issues.

The log file of the Genero OpenID implementation is called OpenID.log and is located in $FGLDIR/web_utilities/services/openid/bin. This log file contains all incoming and outgoing requests. It can help to debug OpenID Single sign-on (SSO) issues.

You can specify the level of detail recorded to the log with the -debug category option of the OpenID server program. There are two categories that can be logged individually or together:
- MSG - Standard information regarding access and errors. By default, only access and error information are logged.
- DEBUG - Traces the entire SSO process.

To add debugging information to the log, modify OpenIDServiceProvider.xcf to include the -debug DEBUG option as first argument in the command defined by the MODULE element:

```xml
<APPLICATION Parent="ws.default"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="FGLPROFILE">
      $(res.path.fgldir.services)/openid/res/fglprofile</ENVIRONMENT_VARIABLE>
    <PATH>$(res.path.fgldir.services)/openid/bin</PATH>
```

SAML SSO

Security Assertion Markup Language (SAML) is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under $FGLDIR/web_utilities/services/saml.

Genero SAML will establish a circle of trust between the service provider (the Genero Application Server) and the SAML identity provider (the entity in charge of managing and authenticating the users).

If you don't configure another database, Genero SAML service will by default use the saml.db SQLite database located in $FGLDIR/web_utilities/services/saml/bin. If Genero is installed with a different user than the user who runs the web server, you must (as a minimum) set write permissions for that user on the openid.db file and its parent bin directory, otherwise the service will fail to insert data into the SQLite database.

Note: Genero implements only version 2.0 of the SAML specification and supports only the HTTP-POST bindings. It is only intended for Genero Web Client applications.

Quick start: Set up SAML in the GAS on page 136
Configure GAS to support SAML SSO on page 138
The ImportIdP program on page 139
Add SAML SSO to a Genero Web application on page 139
Select the SAML server (Identity Provider) on page 140
Define the SAML ID format on page 140
Retrieve the SAML user identifier on page 141
Set the authentication context on page 141
Retrieve identity attributes with SAML on page 142
Authorization and SAML SSO on page 142
Execute an application with SAML SSO on page 143
Genero SAML configuration on page 143
Specify a database to store SAML data on page 145
Genero SAML FGLPROFILE on page 145
Genero SAML log file on page 146

Quick start: Set up SAML in the GAS
Follow these steps to quickly set up SAML for your Genero Application Server and Genero Web Client applications.

In this quick start, you add SAML Single sign-on (SSO) to a Genero Web Client application, then execute the application with SSO.

1. Configure the GAS for SAML SSO:
   a) If your GAS is located behind a proxy, configure the proxy in the SAML fglprofile, located in $FGLDIR/web_utilities/services/saml/res. Uncomment and set values for the entries proxy.http.location and proxy.https.location.
b) SAML requires digital signatures. Create a X509 Certificate and its private key (see the Genero Business Development Language User Guide for details), then modify the SAML configuration file located in $FGLDIR/web_utilities/services/saml/res:

- Uncomment and set values for the entries xml.saml_signature.x509 and xml.saml_signature.key.
- If your Genero Web Client applications must be accessible by HTTP, to be fully secured you must use that key and certificate for XML-Encryption. Uncomment and set the same value for the entries xml.saml_encryption.x509 and xml.saml_encryption.key.

c) Create a circle of trust between the Genero Application Server and a SAML provider.

- Go to $FGLDIR/web_utilities/services/saml.
- Set the SAML environment using the scripts envsaml.bat or envsaml.sh.
- Launch the ImportIdP application with the SAML Provider URL.
  - Example: fglrun ImportIdp http://host:port/openam_954/saml2/jsp/exportmetadata.jsp
  - See SAML provider documentation about how to retrieve the Metadata.
  - If needed, retrieve the SAML provider Certificate and add it as trusted certificate in the SAML configuration file.
    - Uncomment and set values for the entry xml.keystore.calist; see the Genero Business Development Language User Guide for more details.
  - See SAML provider documentation about how to retrieve its X509 certificate.

d) Create a circle of trust between the SAML provider and the Genero Application Server.

- Start the dispatcher (if needed).
- Log into your SAML provider and create a circle of trust based on the Genero Application Server SAML metadata available at this URL: http[s]://host:port/[gas/]ws/r/services/SAMLServiceProvider/Metadata
  - See SAML provider documentation for information on creating the circle of trust.
  - Genero Application Server default SAML identity name is "urn:genero". If needed, you can change the identifier by modifying the saml.entityID entry in the fglprofile file.

2. Add SAML SSO to a Genero Web Client application:

a) Add the DELEGATE tag to all Genero Web Client applications requiring SSO.

```xml
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.mypath)/myapplication</PATH>
    <MODULE>myapp.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider" />
  </EXECUTION>
</APPLICATION>
```

For example:

```xml
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.mypath)/myapplication</PATH>
    <MODULE>myapp.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider" />
  </EXECUTION>
</APPLICATION>
```

3. Execute a Genero Web Client application with SSO:

a) Start your browser and enter the application URL.

You are redirected to the SAML provider and prompted to enter your credentials.

b) Enter your credentials and click the signin button.

If your credentials are valid, your browser is redirected to the Genero Web Client application. The application starts and runs as the entered SAML user.

The next time you start the same application - or any application delivered by the same Genero Application Server - you will not be prompted for your credentials. The application will start (and be authenticated for) the same SAML user.
Configure GAS to support SAML SSO
Follow these steps to setup Genero SAML service.

Before you can use SAML Single sign-on (SSO) with the Genero Application Server, a circle of trust must be established between the service providers (the Genero Application Servers) and one or more SAML identity providers (an entity in charge of managing and authenticating the users). This is established via SAML metadata exchange, where each party imports the metadata from the other party. Each party's metadata defines how to communicate with it.

Note: An X509 certificate authority file can also be exchanged in order to validate SAML signatures. See Certificate authority on page 145.

1. If the Genero Application Server is located behind a proxy, configure the proxy in the SAML FGLPROFILE.
   Uncomment and set correct values for the entries proxy.http.location and proxy.https.location.
2. Create an X509 Certificate and its private key.
   SAML requires digital signatures. See the Genero Business Development Language User Guide for information on creating the certificate and its private key.
3. Modify the SAML configuration file and enter the X509 certificate and private key information.
   The SAML configuration file is located in $FGLDIR/web_utilities/services/saml/res.
   Remove the comment and set correct values for the entries xml.saml_signature.x509 and xml.saml_signature.key.
   If your Genero Web application must be accessible in HTTP, you must also use that key and certificate for XML-Encryption to be fully secure. Uncomment and set the same values for the entries xml.saml_encryption.x509 and xml.saml_encryption.key.
4. Create a circle of trust between the Genero Application Server and a SAML provider. Import the IdP metadata file into the Genero Application Server SAML service provider.
   a) Go to $FGLDIR/web_utilities/services/saml.
   b) Set SAML environment via envsaml.bat or envsaml.sh.
   c) Launch the ImportIdP application using the SAML Provider URL.
      Refer to the IdP documentation for information on generating the metadata file (or the url) from the SAML identity provider.
      $fglrun ImportIdP http://host:port/openam_954/saml2/jsp/exportmetadata.jsp
   d) Retrieve the SAML provider Certificate and add it as a trusted certificate in the SAML configuration file (if needed).
      Uncomment and set the correct values for the entry xml.keystore.calist. Refer to the Genero Business Development Language User Guide for more information.
      Refer to the SAML Identity Provider (IdP) documentation for information about retrieving its X509 certificate.
5. Create a circle of trust between the SAML provider and the Genero Application Server.
   a) Start the dispatcher (if needed).
   b) Log in to your SAML provider and create a circle of trust based on the Genero Application Server SAML metadata.
      Generate the metadata from this URL: http[s]://host:port/[gas/]ws/r/services/SAMLServiceProvider/Metadata
      Refer to the SAML Identity Provider (IdP) documentation for information about importing the Genero Application Server SAML metadata.

The Genero Application Server is ready to support SAML SSO.
The ImportIdP program
Use the ImportIdP program to register a SAML identity provider.

With the Genero ImportIdP program, you can:
- Register a new SAML identity provider (IdP) into the Genero Application Server for SAML Single sign-on (SSO).
- Lists all registered IdPs
- Remove the Idp identified by its URI.

To register a new IdP, you must execute the ImportIdP program with the IdP metadata file or URL. Using a URL can require a proxy configuration in the FGLPROFILE file.

The ImportIdP.4gl source code is provided in $FGLDIR/web_utilities/services/saml/src, and the compiled version is in the bin directory.

Syntax

```
fglrun ImportIdP [options] <url|file>
```

Note:
1. **options** are described in Table 10: ImportIdP options on page 139.
2. **url** is the url of a SAML identity provider.
3. **file** is the metadata file of a SAML identity provider.

Table 10: ImportIdP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-import</td>
<td>Import the IdP specified by the URL or metadata file.</td>
</tr>
<tr>
<td>-list</td>
<td>List all registered IdPs.</td>
</tr>
<tr>
<td>-remove</td>
<td>Remove the registered IdPs.</td>
</tr>
</tbody>
</table>

Add SAML SSO to a Genero Web application

Follow these steps to add SAML SSO to a Genero Web application.

This task must be performed in the .xcf application configuration file for the Genera Application Server.

Add the `<DELEGATE service="services/SAMLServiceProvider"/>` element to the application configuration (.xcf) file.

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfxtwa.xsd">
 <EXECUTION>
  <PATH>$(res.path.qa)/applications/myapp</PATH>
  <MODULE>App.42r</MODULE>
  <DELEGATE service="services/SAMLServiceProvider"/>
 </EXECUTION>
</APPLICATION>
```

With the above configuration and default GAS configuration, the delegation will point to the $FGLDIR/web_utilities/services/SAMLServiceProvider.xcf file.

For more information about the DELEGATE configuration element, see DELEGATE on page 365.
The Genero Application Server will handle the SAML protocol and start the Genero web application only when the user has been authenticated, otherwise an HTML error page is returned.

**Select the SAML server (Identity Provider)**

Follow these steps to specify the SAML server a Genero application must use as its Identity Provider (IdP).

Before you begin, determine the EntityID name for the IdP server you wish to specify. Use the `ImportIdP program` with the `-list` option to identify the EntityID name.

Complete this procedure to specify which SAML server a Genero application must use as its Identity Provider (IdP).

If the `IDP` element is not set in the `DELEGATE` element of the application configuration file, the Genero Application Server will retrieve the unique registered IdP. It will raise an error if more than one IdP is registered.

Add an `IDP` element as a child of the `SAML DELEGATE` element in the application configuration (`xcf`) file. Enter the EntityID name in the `IDP` tag.

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
    <EXECUTION>
        <PATH>$(res.path.qa)/applications/myapp</PATH>
        <MODULE>App.42r</MODULE>
        <DELEGATE service="services/SAMLServiceProvider">
            <IDP>http://idp.4js.com</IDP>
        </DELEGATE>
    </EXECUTION>
</APPLICATION>
```

The Genero Application Server will use the specified IdP as its Single sign-on (SSO) identity provider.

**Define the SAML ID format**

Follow these steps to define the ID format to receive from the SAML IdP.

The SAML Single sign-on (SSO) protocol allows federation of identities. This means that a single user can have different identities on different SAML IdPs. To federate a same user across several IdPs, the notion of ID format was introduced.

The default ID format is transient, meaning that the returned ID is only valid for the current session and has only a meaning for the IdP the GAS is connected to. Other formats exist such as email or persistent, but you must be sure that your IdP supports them, otherwise you will get an error. The IdP decides which format they support. See SAML core specification for more details about the supported ID format.

The ID format allows you to specify how the user is represented to a Service Provider. For Genero Application Server, it defines what piece of data is sent from the IdP to the Genero Application Server to represent the user.

To define the ID format you want to receive from your IdP:

**Add an IDFORMAT element with a valid SAML URN as a child of the SAML DELEGATE element in the application configuration (xcf) file.**

In this example, the IdP will return the email of the authenticated user to the Genero Application Server as `SAML_ID` environment variables.

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
```

```xml
</APPLICATION>
```
When set, the SAML_ID environment variable retrieved in the application program will be in the format specified in the IDFORMAT element of the configuration file.

**Retrieve the SAML user identifier**  
Follow these steps to retrieve the SAML user identifier in your Genero application.

To retrieve the SAML ID returned by the SAML Single sign-on (SSO) Identity Provider (IdP) in your Genero application, add this code:

```plaintext
LET id = fgl_getenv("SAML_ID")
```

**Set the authentication context**  
At the Genero Application Server level, you can specify how the Identity Provider must authenticate a user that wants to access a Genero Web application via a browser.

As a prerequisite, see the SAML core specification for the list of supported URNs. There are several methods -- password protected, X509 certificate, PGP -- but not all work for Web-based Single sign-on (SSO).

**Note:** For most Web Single sign-on, the default authentication method is password protected.

SAML provides a mechanism that allows a service provider (Genero Application Server) to define how a user must be authenticated by the Identity Provider (IdP). The Genero Application Server supports an optional element (AUTHCONTEXT) that allows you to specify which authentication method to use.

If the AUTHCONTEXT element is not defined, the default mechanism set in the IdP is used.

**Important:** Do not specify this tag unless you require a specific authentication method.

Add an AUTHCONTEXT element as a child of the SAML DELEGATE element in the application configuration (xcf) file. Enter a valid authentication method in the text of the AUTHCONTEXT element.

```xml
<?xml version="1.0"?>  
<APPLICATION Parent="defaultgwc"  
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:noNamespaseSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">  
  <EXECUTION>  
    <PATH>$\{res.path.qa\}/applications/myapp</PATH>  
    <MODULE>App.42r</MODULE>  
    <DELEGATE service="services/SAMLServiceProvider">  
      <AUTHCONTEXT>urn:oasis:names:tc:SAML:2.0:ac:classes:X509</AUTHCONTEXT>  
    </DELEGATE>  
  </EXECUTION>  
</APPLICATION>
```

When set, the authentication context method is defined. If the IdP does not support the specified method, or if it uses another mechanism, the Genero Application Server will return an access denied page.
Retrieve identity attributes with SAML
Follow these steps to retrieve attributes about user identity when authenticating to SAML IdP.

As a prerequisite, SAML Single sign-on (SSO) protocol does not provide a mechanism to request specific attributes to be returned when authenticated. You must configure that list at the IdP level. As SAML supports identity federation, it provides a mechanism to map user-specific attributes between different IdPs - an attribute called with one name in one IdP can be called a different name in another IdP. If federation is in use, map them according to other IdPs if needed. Refer to your IdP documentation for more information on how to map and define the list of attributes to pass to the GAS during authentication setup.

To retrieve the SAML attributes returned by the IdP in your Genero application, add a `fgl_getenv()` call for each attribute specified in the XCF file with a prefix of `SAML_`.

```plaintext
LET email = fgl_getenv("SAML_email")
LET fullname = fgl_getenv("SAML_fullname")
LET country = fgl_getenv("SAML_country")
```

The Genero application retrieves the requested identity attributes.

Authorization and SAML SSO
Authorize whether an user already authenticated by SAML Single sign-on (SSO) can access a Genero application.

The GAS must be configured for SAML SSO. See Configure GAS to support SAML SSO on page 138.

With the Genero SAML implementation, you can add an external program to determine whether an already authenticated user can access a Genero Web application.

This external program can be written in Genero or in another programming language.

The authorization program expects two mandatory arguments and the list of SAML attributes received from the Identity Provider (IdP):

```plaintext
access-program saml-userid app-xcf-path [ attribute value [...] ]
```

- The first argument is the SAML identifier. It depends on the ID format specified in the Genero Application Server configuration and by the IdP.
- The second argument is the application path.
- Next arguments are optional and define SAML attributes/value pairs.

Example with a Genero authorization program:

```plaintext
fglrun AccessProgram
 "AZEd3R4" \
 "qa-test/application" \n "fullname" "genero test" \n "email" "genero@4js.com" \n "country" "France"
```

The application `AccessProgram.4gl` in `FGLDIR/web_utilities/services/openid` provides an example of an authorization application written in Genero.

The external program is specified in the application configuration element by adding a `AUTHORIZATION` element in the `DELEGATE` element.

If the `AUTHORIZATION` element is not defined, any user registered in the SAML IdP can access the Genero Web application. It is recommended that you add an authorization program to filter the access to your application.

**Note:** The external program must be deployed beside the `SAMLServer.42r` program, because it will be executed by that service program. This is by default under `FGLDIR/web_utilities/services/saml/bin`.  

1. **Add an AUTHORIZATION element as a child of the SAML DELEGATE element in the application configuration (xcf) file.**

2. **Within the AUTHORIZATION tag, specify the external authorization program.**

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider">
      <AUTHORIZATION>fglrun AccessProgram</AUTHORIZATION>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

The authorization program will be called before access to the Web application is granted. If the authorization program exits with an error code of zero (0), then access is granted for the user. Any exit code other than zero indicates access for the user is denied. In the last case, the end user will be warned with an error page in the web browser, generated by the SAML service.

**Execute an application with SAML SSO**

Follow these steps to execute a Genero Web Client application and authenticate user with SAML Single sign-on (SSO).

1. Open a browser and enter the application URL.
   You are redirected to the SAML provider and prompted to enter your credentials.
2. Enter your credentials.
3. Click the **sign in** button.

If the credentials are valid, your browser redirects to your Genero Web Client application, which starts and runs as the entered SAML user.

When you next start the same application, or any application served from the same Genero Application Server, you will not be prompted for your credentials. The Genero Web Client application will start and authenticate with the same SAML user.

**Genero SAML configuration**

Specify **FGLPROFILE** entries to configure the Genero SAML service provider.

The Genero SAML implementation provides a list of **FGLPROFILE** entries to configure the Genero SAML service provider. The configuration file is located in `$FGLDIR/web_utilities/services/saml/res`.

**Table 11: SAML-related FGLPROFILE entries**

<table>
<thead>
<tr>
<th>FGLPROFILE entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saml.entityID</td>
<td>Defines the SAML entity name for the Genero Application Server, which is how the Genero Application Server is represented to other SAML partners. Mandatory. Default is <code>urn:genero</code>.</td>
</tr>
<tr>
<td>saml.allowUnsecure</td>
<td>Defines whether the GAS accepts unsecured authentication mechanisms. Default is <code>false</code> (recommended). A SAML authentication mechanism is unsecured if communication between the Identity Provider (IdP)</td>
</tr>
<tr>
<td>FGLPROFILE entry</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>saml.wantAssertionsSigned</td>
<td>Defines whether SAML assertions coming from Identity Providers (IdPs) must be signed. Default is true (recommended). It is recommended to have either (or both) saml.wantAssertionsSigned and saml.wantResponseSigned set to true, to ensure the request was not altered. If not signed and entry is set to true, the Genero Application Server returns an access denied HTML page. This entry also adds the wantAssertionsSigned attribute to the SAML metadata describing the SAML needs of the Genero Application Server.</td>
</tr>
<tr>
<td>saml.wantResponseSigned</td>
<td>Defines whether SAML requests coming from the Identity Providers (IdPs) must be signed. Default is false. It is recommended to have either (or both) saml.wantAssertionsSigned and saml.wantResponseSigned set to true, to ensure the request was not altered. You must also take into account the configuration of the Identity Provider (IdP). If not signed and entry is set to true, the Genero Application Server returns an access denied HTML page.</td>
</tr>
</tbody>
</table>

**Assertion encryption**

To support assertion encryption, you must add an X509 certificate and its RAS private key to handle XML-Encryption using the Genero Web Services xml key mapping. There are two entries to be set:

- `xml.saml_encryption.x509`: path to the X509 certificate
- `xml.saml_encryption.key`: path to the RSA private key

You can use the same X509 certificate and RSA private key for signature, encryption and metadata signature.

**Authentication signature**

To sign the authenticate request the Genero Application Server sends to the Identity Provider (IdP), you must add an X509 certificate and its RSA private key to handle XML-Signature using the Genero Web Services xml key mapping. There are two entries to be set:

- `xml.saml_signature.x509`: path to the X509 certificate
- `xml.saml_signature.key`: path to the RSA private key

You can use the same X509 certificate and RSA private key for signature, encryption and metadata signature.
Metadata signature

To sign the generated SAML metadata, add an X509 certificate and its RSA private key in charge of XML-Signature using the Genero Web Services xml key mapping. There are two entries to be set:

- `xml.saml_metadata_signature.x509`: path to the X509 certificate
- `xml.saml_metadata_signature.key`: path to the RSA private key

You can use the same X509 certificate and RSA private key for signature, encryption and metadata signature.

Certificate authority

As XML-Signature and XML-Encryption are in use to secure SAML communication, you must specify the list of trusted certificate authorities. This is done via the Genero Web Services key mapping mechanism, where this entry must be added, containing the list of trusted X509 certificates (coming from the Identity Provider (IdP)).

- `xml.keystore.calist`: path of colon-separated certificate authorities the Genero SAML service provider trusts.

Specify a database to store SAML data

Follow these steps to specify a database different from the default database for the Genero SAML implementation.

The implementation of Genero SAML Single sign-on (SSO) requires a database, to store SAML data related to the protocol. By default, the database engine is SQLite and the database file is `$FGLDIR/web_utilities/services/saml/bin/saml.db`. This database is fully functional after installing the Genero Application Server.

1. Create a new or use an existing database, eventually on a dedicated machine, if several GAS servers are configured for load balancing. There must be a unique database, to centralize all SAML authentication data.

2. In the file `DBase.4gl`, modify the functions `BDConnect()` and `DBDisconnect()` to handle and customize the database connection. Recompile the `DBase.4gl` source.

   `DBase.4gl` is found in `$FGLDIR/web_utilities/services/saml/src`.

3. Modify `fglprofile` in `$FGLDIR/web_utilities/services/saml/res` to include the connection information for the database.

4. Create SAML tables with the `CreateDatabase.4gl` program. Define the database permissions required to let the Genero Application Server modification the SAML tables in the new database.

   `CreateDatabase.4gl` is found in `$FGLDIR/web_utilities/services/saml/src`.

5. If needed, define the PATH (Windows®) or LD_LIBRARY_PATH (Linux®/UNIX™) environment variables in `$FGLDIR/web_utilities/services/SAMLServiceProvider.xcf` with `ENVIRONMENT_VARIABLE` elements, in order to find the database client libraries required by Genero SAML service. Note that if you use SQLite (by default), you do not need to add the path to the library since it is integrated in the ODI driver on most systems.

   The alternate database is now used for the Genero SAML implementation.

Genero SAML FGLPROFILE

Genero SAML Single sign-on (SSO) implementation uses its own FGLPROFILE file.

The file is located in `$FGLDIR/web_utilities/services/saml/res/fglprofile`.

This file can be modified to define the following features:

- ODI database driver definition.
- HTTP and HTTPS proxy configuration. This is needed only when the ImportFGL tool is used.

When to modify this file:
• If you want a database engine other than SQLite.
• If your Genero Application Server installation requires proxy configuration to connect to an SAML provider.

**Genero SAML log file**
The Genero SAML Single sign-on (SSO) implementation produces a log file that helps to identify issues.

The log file of the Genero SAML implementation is called `SAML.log` and is located in `$FGLDIR/web_utilities/saml/bin`. This log file contains all incoming and outgoing requests. It can help to debug SAML issues.

You can specify the level of detail recorded to the log with the `-debug` category option of the SAML server program. There are two categories that can be logged individually or together:

• **MSG** - Standard information regarding access and errors. By default, only access and error information are logged.
• **DEBUG** - Traces the entire process of single sign-on (SSO).

To add debugging information to `SAML.log`, modify `SAMLServiceProvider.xcf` to include the `-debug DEBUG` option in the command defined by the `MODULE` element:

```xml
<APPLICATION Parent="ws.default"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextws.xsd">
   <RESOURCE Id="res.saml.db" Source="INTERNAL"/>
   <EXECUTION>
     <!-- ENVIRONMENT_VARIABLE entries removed for this example -->
     <PATH>$(res.path.fgldir.services)/saml/bin</PATH>
     <MODULE>SAMLServer -logPath $(res.appdata.path) -debug DEBUG -debug MSG</MODULE>
   </EXECUTION>
   <EXECUTION>
     <POOL>
       <START>0</START>
       <MIN_AVAILABLE>0</MIN_AVAILABLE>
       <MAX_AVAILABLE>10</MAX_AVAILABLE>
     </POOL>
     </EXECUTION>
   </APPLICATION>
```

**Note**: Logging is based on the FGL `ERRORLOG()` function. As several instances of the same SAML server can write to a single log file, the PID of the server is written to the log file as well.

**How to implement custom single sign-on**
The aim of this tutorial is to show the basics for delegating the start of a Genero application to another service, in order to handle the authentication via a REST service.

**Tutorial overview**
Genero Application Server Single sign-on (SSO) solution is based on the "delegate" mechanism. For more details, see *How to implement delegation* on page 113.

The purpose of this tutorial is to:

• Prevent direct access to the application and force the end user to enter a login and password via an XHTML page.
• Add a cookie mechanism allowing the password to be kept by the browser, in order to allow future connections without requiring another login.
• Add a simple method for logging out.

This example can be adapted for your own application needs.
**Important:** This tutorial will help you to understand how to handle authentication using a REST service, using a basic authentication mechanism with a username/password and cookie management. On production sites, the security must be improved. Please consider the recommendations mentioned in the last step of this tutorial.

**Workflow**

The end user types the URL of the application in his web browser (step 1 in the workflow). Instead of starting the application program directly, the GAS forwards the HTTP request to the SSOService program. This program checks the user credentials or cookie validity, and returns the appropriate HTTP response. The response is one of the following:

- indicates the user credential are not valid;
- indicates the user credentials are valid, and creates and sets a cookie in the browser; or
- if an existing cookie is received, allows the execution of the application.

![Workflow Diagram](image)

**Figure 30: Single sign-on tutorial workflow**

As shown in the diagram, the following steps are performed:

1. The user enters a URL in the web browser.
2. The Genero REST service receives the HTTP request and checks for a valid cookie.
3. If the cookie is valid, the REST service returns an HTTP response with code 307, and the description `_GENERO_INTERNAL_DELEGATE_`, to notify the GAS dispatcher to start the application.
4. If the cookie is not valid, the REST service returns a login page to the browser, with HTTP code 200.
5. The end user enters a name and password, and submits the HTML form to the GAS.

**Figure 31: Application URL**

- The Genero REST service receives the HTTP request and checks for a valid cookie.
- If the cookie is valid, the REST service returns an HTTP response with code 307, and the description `_GENERO_INTERNAL_DELEGATE_`, to notify the GAS dispatcher to start the application.
- If the cookie is not valid, the REST service returns a login page to the browser, with HTTP code 200.
- The end user enters a name and password, and submits the HTML form to the GAS.
6. The Genero REST service receives the HTTP request and checks the user credentials.
7. If the user is invalid, the REST service returns an HTTP request with an error page, with HTTP code 200.
8. If the user is authenticated, the REST service creates a new cookie, and returns and HTTP response with code 302, to redirect the browser to the application URL.
9. The browser receives the HTTP response, and redirects to the application URL, with the new cookie returned by the REST service. Then return to step 2 (above).

Initial workflow path
The first time the user enters the application URL, it is redirected to the login page.

![Login page](image)

**Figure 32: Login page**

Based on XHTML standards, when the login button is clicked, all data entered in fields of the form, as well the checking of the box “Would you like to keep password?”, are set as parameters in the URL query string.

**Important:** Use HTTPS in production. This example uses a POST method to submit the login form. With the POST method, the username and password are not visible in the browser URL field, however they are passed in clear text in the body of the POST request. You should use HTTPS in order to encrypt and secure on any production site, and avoid clear data being sent through the network.

In step 2 of the workflow, the SSOService program checks the existence of a valid cookie, and analyzes the URL query string parameters in order to find values for the user login and password. Since this is the first start, there is no existing valid cookie (step 3b of the workflow) and the query string is null because the HTML login form was not yet submitted (step 4 on the workflow). As a result, the connection is refused and the user is directed by the service to a login XHTML page.

- GAS configuration for delegation on page 148
- Handle login and password input on page 149
- Cookie handling on page 150
- Disconnect (log off) on page 151
- Production recommendations on page 151

**GAS configuration for delegation**
Delegating the start of an application to a REST service program.

**Configure the application for Simple SSO provider**
Modify your customization to delegate the execution of the application to a REST web service.
Add a `<DELEGATE>` element under the `<EXECUTION>` element in the customization file (xcf), with a reference to the `SimpleSSOServiceProvider` service:

```
<EXECUTION>
  <PATH>${res.path}</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="services/SimpleSSOServiceProvider">
    </DELEGATE>
</EXECUTION>
```

**Configure the SSO service program**

The name `SimpleSSOServiceProvider` maps to the `SimpleSSOServiceProvider.xcf` configuration file, located in `$FGLDIR/web_utilities/services`. This configuration file contains these execution settings:

```
<EXECUTION>
  <PATH>${res.path.fgldir.services}/simplesso/bin</PATH>
  <MODULE>SSOService.42r</MODULE>
</EXECUTION>
```

In this example, the SSO service program is `SSOService.42r`, and the `fgrun` execution context in `$FGLDIR/web_utilities/services/simplesso/bin`.

**Handle login and password input**

Prevent direct access to the application and force the end user to enter a login and password.

**URL handling in the Simple SSO service program**

Once the login page has been submitted (step 4 in the workflow), the incoming URL is split in 2 parts:
- the base URL
- the query string

The query string is parsed by the `SSOService.4gl` code, to retrieve values for the user name, the password, the state of the “keeping password” checkbox, and any potential query string parameters the user may have manually entered in the URL:

```
CALL parseQueryString(query) RETURNING user, pwd, userQueryString, isCookie
```

**Important:** Consider reviewing the functions included in the sample. They are intended to be simple examples. For a production site, you would want to avoid data transmitted in clear.

Once the user and password have been retrieved from the query string, the service program checks whether the user name and password are valid.

If the user name and password are valid, the application can be started by creating the cookie for the user, as explained in Cookie handling on page 150.

If the user could not be authenticated, the Genero REST service returns an HTTP request with an error page, with HTTP code 200:

```
CALL req.setResponseHeader("Content-Type", "text/html")
CALL req.sendTextResponse(200,"Access Forbidden", html-error-page-string)
```
Cookie handling
The cookie mechanism allows the browser to permanently keep the password, eliminating the need to login when the user revisits the application.

How does the cookie mechanism work?
If the user name and password are valid, the REST service program creates a new cookie to be sent to the browser in the HTTP response. An instantaneous redirection is done, this valid cookie is checked by the service (it returns to step 2 of the workflow) and the connection is accepted (step 3a of the workflow).

The next time the user starts the application, the SSO service program will check the validity of this cookie. According to the cookie’s expiration, the application may start automatically without requiring the user to login again or returning a new login page.

The cookie is set via the HTTP header “Set-Cookie” and requires a name-value pair that can be sent to the browser by a request on the form.

CALL req.setResponseHeader("Set-Cookie", myCookie)

Where myCookie is a string containing a name-value pair, with optional parameters, for example:

LET myCookie = COOKIE_NAME, "=",
    CookieEncrypt(user,pwd,expiredDate,key), "; ",
    "Path=", getCookiePath(path)

Note that the following information should also be set:
• A expiration date for the cookie (cookie attributes "max-age" or "expires").
• A path for limiting the validity of the cookie to the current application (cookie attribute “path”).

Cookie expiration date handling
The date of expiration is usually defined by the cookie attribute “max-age”. It represents a value in seconds relative to the current date and time. For instance, max-age=3600 means that the cookie will expire in 1 hour.

The max-age attribute is not supported by Internet Explorer. The service could use the attribute “expires”, however it requires an absolute GMT hour and Genero currently doesn't handle this format.

To solve this problem, the expiration date is directly included (and encrypted) inside the cookie value.

The cookie structure
To ease the understanding of this example, the cookie has a simple structure:

'name=value;Path=path_value'

• The name of the cookie is hard-coded in the SSOUserFunctions.4gl code with the COOKIE_NAME constant.
• value of the cookie is an encrypted combination of username, password and expiration date, separated by a given separator. Once decrypted, value of the cookie can be something like 'userName="myuser";password="mypassword";expirationDate="12/12/2018"'.

    The separator and the name of attributes can be changed by the constants C_USERNAME (for the username attribute), C_PASSWORD (for the password attribute), C_EXPIRATION (for the expiration date attribute) and C_SEP (for the separator) in the SSOUserFunctions.4gl module.
• Path remains the normal “Path” cookie attribute. path_value should correspond to what is after “http://host:port” in the URL.
How the cookie is handled in the Simple SSO service

The cookie needs to be checked initially by the Simple SSO service (step 2 of the workflow) in order to know if the application can be executed directly or if the end user needs to login again.

There are three options.

1. The cookie is valid (step 3a of the workflow).

   The connection is accepted immediately and the application executed. For checking if the cookie is valid, the service needs to retrieve the content of the “Cookie” HTTP header. Once this content has been retrieved, the service decrypts the value of the expiration date. If the expiration date is later than the current date (CURRENT value), the connection is accepted using HTTP code 307 and the description _GENERO_INTERNAL_DELEGATE_.

2. The cookie is no longer valid as the expiration date has expired (step 3b of the workflow).

   A cookie cannot be valid if the date has expired. The cookie is set to a new value (in our example, the user and password values are set to -1 before being encrypted in this new cookie value) and a redirection is done on the same URL. After redirection, the cookie is decrypted and values “-1” are found for login and password. They are considered as invalid and the user is redirected again to the login page. Specifically, in our sample, it’s redirected to an XHTML login page indicating “the session has expired”.

3. The cookie is NULL (step 3b of the workflow).

   For the initial connection, the cookie is NULL. It can be redirected to a simple “Welcome” page rather than the login page.

Expiration date of the cookie

When the cookie is created, it is handled like this:

- If the “keeping password” box has been checked (password kept), the expiration date is set by default to one year later.
- If “keeping password” box has not been checked (password not kept), the expiration date is almost instantaneous (10 seconds by default, so that the cookie remains only valid one time for redirection)

It can be easily changed by the constants C_EXPIRATION_COOKIE_CHECKED (value in years) and C_EXPIRATION_COOKIE_UNCHECKED (value in seconds) of the SSOUserFunctions.4gl module.

Disconnect (log off)

How does the user disconnect after a permanent cookie is set?

If the end user wants to re-connect when a permanent cookie has been set, the disconnect=true parameter must be added to the URL:

http://host:6394/ua/r/myapp?disconnect=true

The user will be automatically redirected to the login page.

When the cookie has expired, the user name and password are set to -1 before being encrypted in the cookie value. In such instances, redirection is also done to the login page.

Production recommendations

It is important to improve security beyond the tutorial.

The tutorial is designed to convey single sign-on basics. Consider these recommendations when preparing for your production system.

- For easing the understanding of this sample, user, password and expiration date have been encrypted directly in the cookie. This should not be done on a production site. If somebody found the decryption algorithm, he would be able to read user and password values in clear. We recommend you review the encryption mechanism and provide better security by encrypting a hash of the login + password, instead of the login + password.
• Function shown in this sample are “fake” functions adapted especially for this example. They may contain some dummy code. Review these functions in detail before adapting them to your production environment.
• A single unique user name and password are hard-coded in the sample source code. A production site requires a complete user management solution.
• Production sites requires the use of the HTTPS protocol rather than HTTP in order to avoid the transmission of clear data through the network.

Connect to the application database with SSO

There are several solutions for automatically connecting to the database server, after starting an application program with a Single sign-on (SSO) delegation.

Overview

The goal of a complete SSO solution is to let the end-user enter credentials once in a login form on the front-end, authenticate that user with an GAS/SSO mechanism, then start the application program and connect to the database without having the end user input other credentials for the database server.

Depending on the features of the target database server, you can implement different techniques to connect to the database automatically, without having to provide more credentials.

The goal of this topic is not to provide a complete example. There are different authentication methods available, and your SSO solution must be adapted to the type of database and operating system. Consider learning about database user security within the database engine of your application.

**Note:** Because Kerberos SSO support is deprecated by the Genero Application Server, this type of SSO mechanism is not covered in this topic. However, database engines like IBM® Informix® IDS support Kerberos SSO with the Generic Security Services CSM (GSSCM) feature.

User handling depends on the type of Web Application

Regarding application users, we can distinguish the following type of web applications:

• Typical **public web applications**, for an undefined number of end users, who can register themselves to the application.
• Typical **enterprise web applications**, for a defined number of known end users, with strong data access control, managed by application administrators in an enterprise directory.

The SSO solution implemented will depend on the type of web application.

Connecting to the database from the application program

Once the end user is authenticated with one of the SSO mechanisms supported by the GAS, the application is started in the context of the GAS operating system user. For example, when on an Apache server, the application program will execute as the Apache user.

Most application programs then connect to a database server, to store and query application data. Connecting to a database server requires the application to identify and authenticate the end user as a database user.

**Note:** A database user is typically created by a `CREATE USER` SQL statement.

The most common way to authenticate a user in a database connection is to provide the login name and the associated password of the user object existing in the database:

```
MAIN
  DEFINE uname, upswd VARCHAR(50)
  ...
  CONNECT TO "dbsource" USER uname USING upswd
  ...
```
For more details about the CONNECT TO instruction, see "Database Connections" chapter in the Genero Business Development Language Reference Guide.

**Note:** With IBM® Informix® IDS servers, database users are traditionally authenticated at the OS level; there must be an OS login from group "informix", created for each DB user. However, IDS 11.7 introduces the concept of *internal users*, to integrate with external authentication mechanisms or to define pure database users based on logins and passwords.

### Database user creation

In order to use database engine features to control privileges and audit activity, end users must be identified in the database server, as db user objects in the database system. This is typically done with the `CREATE USER` SQL instruction.

When creating a database user object, you must specify the authentication method.

The basic default authentication method is to specify a password be provided each time the SQL session is created. For example, to create a database user with password authentication in an Oracle® database:

```sql
CREATE USER username IDENTIFIED BY password
```

It is also possible to define database users with an authentication method based on credentials issued from a trusted part. For example, in Oracle, you can create a database user that will be authenticated with the Oracle Internet Directory®:

```sql
CREATE USER username IDENTIFIED GLOBALLY AS distinguished name (LDAP DN)
```

### Connecting to the DB with predefined database users

This technique can be implemented for a web application where the number of end users is unknown, and where users can register themselves to the application with requiring the database administrators to create database users. End users enter an application login and password, that will be checked and stored by the application in a dedicated table of the database.

In this solution, use an SSO technology where end users can create credentials from an open identity provider (IdP), as with the GAS OpenID implementation. Since anyone can freely register with the web application, there is no application administrator task regarding the creation of an application user.

To access the application data, a set of predefined database users must be exist in the database, with a fixed name and password, that are hidden to the end users. Each predefined database user will be assigned to several real physical end users. For example, you can create four types of database users, each with specific application permissions and database privileges:

- An *application administrator* can manage application users, can read/write all application data.
- A *read/write access user* can read/write all data of the application.
- A *read access user* can only read all application data.
- A *guest user* can only read a limited set of application data.

Application users are managed and controlled at the application level, and stored in a database table, or in an external resource file (with passwords encrypted).

**Important:** Because application programs will implicitly connect to the database with predefined database users, no security holes can exist in programs that would allow an end user to connect as a database user to attack the database, for example by using SQL injection.

Once the web application is allowed by the GAS SSO mechanism to start and needs to connect to the database, the program must get application user information (login name, password and type of user), from...
the GAS SSO procedure. For example, when using OpenID, the web application must get the end user login, password and user type from OpenID attributes through the corresponding environment variables:

```
LET user_name = FGL_GETENV("OPENID_user_name")
LET user_pswd = FGL_GETENV("OPENID_user_pswd")
LET user_type = FGL_GETENV("OPENID_user_type")
```

The user type identifies the predefined database user that will be used to connect to the database, and in turn determines the privileges allowed for the end user.

In order to connect to the database server, the predefined database credentials must be found according to the user type got from the SSO attributes. For example, the program can get the database user name and password from a encrypted configuration file:

```
CALL get_db_login("config_file", user_type) RETURNING db_user, db_pswd
CONNECT TO dbname USER db_user USING db_pswd
```

Once connected to the database, the application program can issue SQL queries as the predefined database user.

**Note:** Because physical/end users are mapped to predefined/anonymous database users, db server auditing services will not be able to trace end user activity. If needed, this feature must be implemented at the application level.

At this level, the application user must be validated with simple SQL queries. The application user definition can be stored in an application table, where the password should be encrypted:

```
LET user_pswd_encrypted = my_encode(user_pswd)
SELECT last_login INTO ts FROM app_users
  WHERE app_users.u_name = user_name
    AND app_users.u_pswd = user_pswd_encrypted
IF SQLCA.SQLCODE == 100 THEN
  -- application user does not exist: ask for registration, or deny access
  ...
END IF
```

### Connecting to the DB with custom SSO implementation

This technique can be implemented for a public web application (where end users can register themselves), or for an enterprise web application (where end users are known and where creation is controlled).

The principle is similar to the Connecting to the DB with predefined database users solution, but instead of using an standard SSO protocol, the GAS SSO delegation feature is used to implement a custom single sign-on procedure.

Application users (SSO login and password) are handled by the delegation program, and associated database user credentials can be stored in a file or light-weight database, which can then be passed through environment variables to the application program. The application program then issues a regular CONNECT TO instruction with USER db_username USING db_password option.

### Compression in Genero Application Server

Compression is enabled by default for the Genero Application Server. You can disable compression on selected resources or for the entire Genero Application Server.

Files managed by the Genero Application Server can be compressed to reduce the size of files sent to the User Agent. The files can be compressed by hand and deployed in the GAS, but it is also possible to configure the GAS to compress application files on the fly.
Compression is configured in the $FGLASDIR/etc/imt.cfg file. This file lists the type of resources that can be compressed. GAS automatically compresses the files with the file extensions listed in imt.cfg.

**When does compression take place?**

Compression takes place at:

- Installation time
- At application deployment with Genero Archive
- At runtime, when communication files, i.e. HTTP request/response type files that do not need to be saved on disk, are exchanged between GAS and applications. These are compressed on the fly.
- At runtime when files on disk are requested by applications.

**Note:** At runtime, the GAS checks the imt.cfg to see if the requested file is expected to be a compressed file. If the compressed file is not up to date or is missing, the GAS performs the following:

- It compresses the requested resource on the fly and sends a compressed result.
- A warning is displayed in the GAS log (see Logging on page 167) if the compressed file is out of date or is missing. (To prevent these warnings, you will need to update compressed files. You can manually generate the compress files (.gz) with the gasadmin -z command or using another appropriate compression utility command).
- You can compress static files as required using the gasadmin -z command, see gasadmin tool.

**Enabling and disabling GAS compression**

Both the gasadmin -z compression command and the GAS use the imt.cfg file to identify the resources that can be compressed.

imt.cfg sample:

```plaintext
# [-] Internet media type Extension
# The optional '-' sign at the beginning can be used to explicitly
disable compression when sending resources
# Example: see below application/java-archive
#
# Uncomment the following line to completely disable compression
gas:disable-compression
application/andrew-inset ez
 application/java-archive jar
 application/font-woff woff
 application/mac-binhex40 hqx
 application/mac-compactpro cpt
 application/msword doc
...
```

To disable compression for a specific resource, place a '-' sign at the beginning of that resource in the listing. For example:

```plaintext
- application/java-archive jar
```

To disable all compression by the Genero Application Server, uncomment the gas:disable-compression entry in the file.

**Compression enabled (entry is a comment; default):**

```plaintext
# gas:disable-compression
```

**Compression disabled (entry is not a comment):**

```plaintext
gas:disable-compression
```
Configure development environment

Configure a development environment to troubleshoot an Genero Web Client for JavaScript (GWC-JS) application.

To troubleshoot and debug an application, you may need to view the application log files and the AUI tree. For this you will need to set up a development environment. For example, to configure a development environment for applications using uaproy protocol, perform the following steps:

1. In the GAS configuration file (default $FGLASDIR%/etc/as.xcf), change:

```xml
<RESOURCE Id="res.uaproxy.param" Source="INTERNAL"></RESOURCE>
```

to:

```xml
<RESOURCE Id="res.uaproxy.param" Source="INTERNAL">--development</RESOURCE>
```

**Tip:**

Or alternatively:

You can also run the dispatcher from the command line and override the settings in the GAS configuration file for res.uaproxy.param:

```
httpdispatch.exe -E res.uaproxy.param=--development
```

2. Restart the dispatcher.

   The dispatcher must be restarted whenever you modify the application server configuration file in order for the changes to take effect.

3. Enter the application URL in your browser.

   You should now see the **Debug Tools** icon (next to the close application icon) on the right-hand side of your web application window. See **GWC-JS application in debug mode** on page 197.

Configure multiple dispatchers

If you need to configure multiple dispatchers, you must configure log, temporary and session directories in order to distinguish the entries for each dispatcher.

Multiple dispatchers are typically not needed; in fact they are rarely used. You would need to start multiple dispatchers if you needed to have different environments using the same version of the Genero Application Server on the same host. For example, you may wish to co-locate your production, training, and development environments.

To start multiple dispatchers on a single host, you create a copy of the application server configuration file (default as.xcf) for each dispatcher you will start. At a minimum, each dispatcher will have its own uniquely-named configuration file, and the file will have different values for the listening port.

It is important to also ensure that your log, temporary and session directories remain independent. In the default application server configuration file, these directories are located under the directory specified by the resource res.appdata.path. These examples are extracted from the default application server configuration file:

```xml
-- res.appdata.path specifies the directory for application data
-- for the dispatcher

<RESOURCE Id="res.appdata.path"
Source="INTERNAL">C:\ProgramData\FourJs\gas\2.50.07-130789</RESOURCE>
```
Configuring the Genero Application Server

Configuring Genero Report Viewer on GAS

Understand the different options available for configuring the GAS when a Genero Report Engine is operating locally or in distributed mode on another server.

- Configure GRE in local mode on page 157
- Configure GRE in distributed mode on page 158

Configure GRE in local mode

When a Genero Report Engine (GRE) is operating on the GAS locally, the GAS is configured with URLs and environment variables to handle the processing of reports.

The GAS provides three URLs with the report prefix to locate the Genero Web Report viewer and resources.

- $(connector.uri)/ua/report/viewer
- $(connector.uri)/ua/report/public
- $(connector.uri)/ua/report/private/${session.id}

Table 12: URL prefixes

<table>
<thead>
<tr>
<th>URL prefix</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewer</td>
<td>The viewer URL prefix is used to load the HTML report viewer implementation included in the GRE package. It is bound to the $GREDIR/viewer directory by default.</td>
</tr>
</tbody>
</table>
The public URL prefix is used for public resources such as documents, images, or fonts. These resources can be shared between applications. The default location is APPDATA/public.

The private URL prefix is used for private resources. These resources are not shared between applications or users. The access to private files is controlled by the GAS and are deleted when the session ends. The default location is APPDATA/tmp/dispatcher_name/$(session.id)/reports.

The GAS provides several environment variables to the started application. These environment variables are used with reports for the GRE operating in local mode.

**Table 13: Environment variables dedicated to GRE in local mode**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRE_PRIVATE_URL_PREFIX</td>
<td>prefix</td>
<td><a href="http://localhost:6394/ua/report/private/89be6b3867148e738fc6bd1daad1ff2e">http://localhost:6394/ua/report/private/89be6b3867148e738fc6bd1daad1ff2e</a></td>
</tr>
<tr>
<td>GRE_PUBLIC_URL_PREFIX</td>
<td>prefix</td>
<td><a href="http://localhost:6394/ua/report/public">http://localhost:6394/ua/report/public</a></td>
</tr>
<tr>
<td>GRE_PRIVATE_DIR</td>
<td>Private resource directory</td>
<td>private/89be6b3867148e738fc6bd1daad1ff2e</td>
</tr>
<tr>
<td>GRE_PUBLIC_DIR</td>
<td>Public resource directory</td>
<td>public</td>
</tr>
</tbody>
</table>

**Configure GRE in distributed mode**

If you are running the Genero Report Engine (GRE) daemon on a different machine than the DVM, you must configure the REPORT_REMOTE_URL_PREFIX with the URL of the remote server.

The GAS provides three URLs with the report-r prefix to locate the Genero Web Report viewer and resources.

**Note:** The URL prefix used is report-r as distinct from report used in local mode. The use of the viewer, private and public URL prefixes are the same as those used for local reports.

- $(connector.uri)/ua/report-r/viewer
- $(connector.uri)/ua/report-r/public
- $(connector.uri)/ua/report-r/private/$(session.id)

The GAS proxies these URLs on the remote server, adding first the URL of the server specified in the REPORT_REMOTE_URL_PREFIX in the application configuration file.
The $\{\text{connector.uri}\}$ is provided by the GAS. This will vary as it is implementation dependent. In our example it is "viewer":

```
http://remotehost:12345/viewer
```

### Get private resources for report in distributed mode

When the GAS receives a request for a resource with a URL with report-r, this indicates that the GRE is operating in distributed mode on another server. For example:

```
http://localhost:6394/ua/report-r/private/
  ebf19aba4d09e80cd1c9cfe1a7868a92a/4BF69549-BA5C-4258-8F42-BFB4A3494339/
  page000001.svg
```

The GAS will proxy the request to the configured server using the URL specified in the REPORT_REMOTE_URL_PREFIX in the application configuration file (xcf), for example like this:

```
http://remotehost:12345/reports/private/
  ebf19aba4d09e80cd1c9cfe1a7868a92a/4BF69549-BA5C-4258-8F42-BFB4A3494339/
  page000001.svg
```

### GAS end of session notification

**Note:** At the end of the session the GAS will send an HTTP request to the remote host to notify that the session has ended for the purpose of deleting the private data associated with the session identifier.

```
DELETE /reports/delete/ebf19aba4d09e80cd1c9cfe1a7868a92a
```

The GAS provides several environment variables to the started application. With these the GAS is able to access the GRE, and provide applications with the public and private resources that are configured for reports.

**Important:** Resources must be located in the associated public and private directories on the remote server.

### Table 14: Environment variables dedicated to GRE in distributed mode

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRE_REPORT_VIEWER_R_URL_PREFIX</td>
<td>URL of the server where the GRE daemon is running.</td>
<td><a href="http://localhost:6394/ua/report-r/viewer">http://localhost:6394/ua/report-r/viewer</a></td>
</tr>
<tr>
<td>GRE_PRIVATE_R_URL_PREFIX</td>
<td>Private URL prefix</td>
<td><a href="http://localhost:6394/ua/report-r/private/">http://localhost:6394/ua/report-r/private/</a></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Example Values</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>GRE_PRIVATE_R_DIR</td>
<td>Private resource directory.</td>
<td>private/89be6b3867148e738fc6bd1daad1ff2e</td>
</tr>
<tr>
<td>GRE_PUBLIC_R_DIR</td>
<td>Private URL prefix</td>
<td>public</td>
</tr>
</tbody>
</table>
Administering the Genero Application Server

Understand the options available for the administration of the Genero Application Server.

- Monitoring on page 161
- Logging on page 167
- Using the debugger on page 168
- Performance tuning on page 169
- Load balancing on page 170

Monitoring

The GAS Monitor displays information on the GAS dispatcher and on active applications. This information is available via an URL, as an XML document.

Statistics about the GAS are provided as an XML document through the /monitor URL. You can modify the Statistics presentation by customizing the monitor.xslt file located in $FGLASDIR/web/fjs.

- Usage on page 161
- Statistics on page 161

Usage

Enter a URL to access the monitor.

Standard URL with direct connection:

http://appserver:port/monitor

For more details on monitor configuration, see Monitor - Configuration Reference in the Configuration Reference section.

If you are using a Web server, you will enter in a different URL. Refer to The Application Web Address for more information.

Statistics

The statistics or information provided by the GAS Monitor can be viewed in two steps.

Step 1: Viewing information about the dispatcher

When you first access the GAS Monitoring page, information is provided about the Dispatcher.

Example: http://localhost:6394/monitor provides the dispatcher details.
Figure 33: Monitoring: Dispatcher details

Example: http://localhost/gas/monitor provides the dispatcher details.
Figure 34: Monitoring: Dispatcher details

From this page, you can view Dispatcher Resources usage information by clicking on the Dispatcher Resources usage link.
Dispatcher details

- Launched at Tue Jun 22 12:28:06 2010
- Dispatcher: httpdispatch
- Version: 2.30.00-88289
- Build date: Jun 21 2010
- **Dispatcher Resources usage**
  - Process id: 3588
  - Resources used since startup:
    - Real time: 167.641000s, CPU: 0.090000
    - User time: 0.046875s, CPU: 0.020000
    - System time: 0.109375s, CPU: 0.060000
  - Resources used since last check:
    - Incremental Real time: 67.313000s, CPU: 0.040000
    - Incremental User time: 0.000000s, CPU: 0.000000
    - Incremental System time: 0.031250s, CPU: 0.040000
- Number of running threads: 1
- **Monitoring for http requests**
  - Handled requests: 22
  - In-progress requests: 1
  - Successful requests: 21
  - Detailed info per request type:
    - unknown
    - /ws/r
    - /ja/r
    - /ja/sua
    - /wa/r
    - /wa/sua
    - /wa/i
    - /wa/ka
    - /wa/gtree
    - /wa/gpaths
    - /wa/ft
    - /monitor
- **Monitoring for GWS proxies**
  - Number of active pools: 0
- **Monitoring for GDC proxies**
  - Number of active sessions: 0
- **Monitoring for GWC proxies**
  - Number of active sessions: 2
    - `_default/gwc-demo`
      - 3a7267721ea08d18b381005ebd1c2b9f)
      - Average keep-alive time: 0.234375s
      - Inactivity time: 83.421875s
      - Connections: 1
    - `_default/gwc-demo`
      - 5c196489d9108b34ff789c2b49f70ba3)
      - Average keep-alive time: 0.062500s
      - Inactivity time: 9.125000s
      - Connections: 1

Figure 35: Monitoring: Dispatcher Resources usage
You can monitor detailed information per request type by clicking one of the request type links. For more information on what each of the request types are, see **URLs Acknowledged by the GAS**.

For example, to see statistics relating to the launching of a GWC application, you could select to view the statistics for the request type `/wa/r`.

![Dispatcher details](image)

**Figure 36: Monitor: Statistics for /wa/r**
Step 2: Viewing information about the VMProxies

In addition to viewing statistics about the dispatcher, you can also view statistics about a session by clicking on session links under the Monitoring for GWS / GDC / GWC proxies sections. The statistics and information for the session - for the proxy - appear in a separate window. To return to the Dispatcher window, you must close the Session window.

![Dispatcher details](image)

**Figure 37: Viewing proxy statistics**
**Logging**

When you run an application, the Genero Application Server (GAS) creates separate log files for its dispatchers, proxies, and the Dynamic Virtual Machines (DVMs) started by those proxies. These files may be viewed for troubleshooting.

- A log file is generated for each dispatcher. This log file captures incoming requests, the starting of proxies, responses sent, and system error messages.
- A log file is generated for each proxy started. A separate log file is generated for each proxy started.
- A log file is generated for each DVM started. DVM standard error and standard output are sent to the dedicated DVM log files.

When in development mode, the ending page for a Genero Web Client for JavaScript (GWC-JS) Web application contains a link to the DVM log file.

**The application ended**

```
Session ID : 5cd65d65cac5af6992530dbdbdc31e93
```

- Get the VM logs
- ✕ Restart the same application

**Close**

**Figure 38: Example GWC-JS application ended page**

Log files are also accessible from the GAS monitor via the (LOG) link next to the proxy name or PID display.

**Note:** See Table 5: appdata Directories and Files on page 41 for details on the location of log files.

**Log file names**

For the dispatcher log, the name specifies the type of dispatcher. Example:

- `httpdispatch.log`

For the proxy log, the name indicates the type of proxy. Examples:

- `uapoxy-<session-id>.log`
- `gwsproxy-<group>-<app>.log`

For the DVM log, if the DVM is started by the `gdcproxy` or `gwcproxy`, the name includes the `session-id`:

- `vm-<session-id>.log`

When working with Web services, a GWS proxy can spawn multiple DVMs. Each DVM gets its own log file. The log file is suffixed with a number from 0 to MAXAVAILABLE-1. A log file is reused for new DVM logs if the previous DVM has finished, to avoid the accumulation of log files on disk.

- `vm-<group>-<app>-<number>.log`
Examples:

- vm-demo-Calculator-0.log
- vm-demo-Calculator-1.log

**Manage the Genero Application Server log files**

The GAS creates a log for each application session. As a result, you can end up with a lot of log files. You should have some plan for archiving and removing log files. For UNIX-based platforms, you can use utilities such as logrotate to compress and move log files. For Windows, any program that can compress and archive log files can be used.

*Note:* If using `logrotate` on Apache web server logs, logrotate will start and stop the Apache server. When Apache restarts, it also starts a new `fastcgidispatch` process (see Apache: mod_fastcgi on page 90) while it may not stop the existing process. If you observe this behavior, you can set logrotate’s `prerotate` script to get the pid of the running `fastcgidispatch` process, it should then be possible to stop the old `fastcgidispatch` process in the `postrotate` script, for more information see Logrotate.

**Using the debugger**

This section provides instructions for using the debugger for the httpdispatch.

In addition to using the instructions below, you can use the graphical debugger in Genero Studio. For more information on the graphical debugger, refer to the Genero Studio User Guide.

- Run the debugger for the GAS on the Windows platform on page 168
- Run the debugger for the GAS on Linux/UNIX on page 169

**Run the debugger for the GAS on the Windows® platform**

To run the FGL debugger, the dispatcher must open a DOS command or a xterm window and then run `fglrunc -d`.

1. In the GAS configuration file (default `%FGLASDIR%/etc/as.xcf), change:

```xml
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">
  $(res.fgldir)\bin\fglrunc.exe</RESOURCE>
```

to:

```xml
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">
  c:\Fjs\gas\debug.bat</RESOURCE>
```

(Windows®) for example, where debug.bat contains `cmd /K start cmd`

2. In the application configuration file, change the DVM availability timeout value to allow you time to type your debug commands.

For example, change:

```xml
<DVM_AVAILABLE>10</DVM_AVAILABLE>
```

to:

```xml
<DVM_AVAILABLE>60</DVM_AVAILABLE>
```

This change allows you 60 seconds in which to type your debug commands.

3. Restart the dispatcher.
(The dispatcher must be restarted whenever you modify the application server configuration file in order for the changes to take effect.)

4. Enter the application URL in your browser.
   This opens a shell window.

5. Type the commands to run the application:
   
   ```
   fglrun -d test.42r  
   (fgldb) b test:20  
   (fgldb) run
   ```
   
   Opens the debugger tool and sets it on program test.42r.
   Sets a break point at line 20.
   Runs the application.
   
   This refreshes the browser, like the FGL debugger does with the GDC.

   **Tip:**

   You can also run the dispatcher from the command line and override some of the settings for res.dvm.wa:

   ```
   httpdispatch -E res.dvm.wa="cmd /K start cmd" (Windows®)
   ```

**Run the debugger for the GAS on Linux®/UNIX™**

These instructions assume that you are operating within a graphical environment. If you are not operating within a graphical environment, simply enter the commands you want to process in the script.

To run the dispatcher, enter the following:

```
httpdispatch -E res.dvm.wa="/home/test/xterm.sh"
```

In the `xterm.sh` shell, you have: `/usr/bin/xterm` (the complete path to xterm).

This removes all of the options given by the dispatcher along with all error messages. A new xterm is opened. At this point, proceed as you would if you were running your applications from a Windows® platform.

**Performance tuning**

These topics cover various performance tuning considerations for configuring your Web server, your Genero Application Server, and your Genero applications.

- **Web server configuration: Keep Alive** on page 169
- **SPDY** on page 169

**Web server configuration: Keep Alive**

Recommendations for the keep alive settings on the Web server.

To improve performance, we recommend that you turn on the HTTP Keep-Alive feature on the Web server.

Prior to version 2.50, if you were connecting to the GDC through the GAS (via the GDCProxy), it was recommended that the connection timeout be longer than 120 seconds. The recommendation is removed starting with version 2.50.

**SPDY**

SPDY (pronounced “SPeeDY”) is an experimental network protocol created to transport Web content. The Genero Application Server is compatible with the SPDY protocol.

The main goal of SPDY is to reduce web page load time by:

- Multiplexing unlimited concurrent file transfers over a single connection. In other words, it allows many concurrent HTTP requests across a single TCP connection.
• Ordering file transfers by priority, preventing the channel from being congested with non-critical resources.
• Reduce bandwidth by compressing headers and eliminating unnecessary headers. For example, User-Agent, Host, Accept are typically static and do not need to be resent.
• To enable the server to initiate communications with the client and push data to the client whenever possible.
• SSL is the underlying transport protocol:
  • better security
  • compatibility with existing network infrastructure
  • ensure communication across existing proxies is not broken

The SPDY protocol is built-in with Firefox® (version 13 and greater) and Google Chrome.

It is the responsibility of the system administrator to install SDPY for their Web servers (where available). It is transparent to the GAS.

**Important:** GAS 2.50 provides a built-in compression feature. If SPDY is used and compression is active in SPDY, you should disable the built-in GAS compression. SPDY will do a better job of compression as it can compress HTTP headers as well as HTTP request/response bodies.

For more information:
• SPDY home page
• SPDY white paper
• SPDY protocol
• Apache module for SPDY / installation instructions

### Load balancing

One way to increase the capacity of the Genero Application Server (GAS) is to scale it out by deploying multiple instances of the GAS on different servers.

As the GAS is fully integrated with existing Web servers, third-party tools can be used to implement load balancing so that network traffic is distributed across a group of backend servers.

**Load balancing options**

GAS applications can be load balanced using standard load balancing techniques, including software load balancers such as Windows® Network Load Balancing or HAProxy, as well as hardware-based load balancing appliances.

**Note:** There is no reason to have a Web server, its dispatcher, and the VMProxy on different machines. While the load balancer will balance load across different machines, on each machine you will find one Web server, one dispatcher, all attached proxies, and all attached DVMs. In production, you should think of the GAS as a group comprised of the Web server, dispatcher and proxies, and DVMs.

**Configuring load balancer**

The load balancer gateway is the entry point of your application. It routes the client request to the appropriate GAS. The GAS behind the load balancer needs to know this entry point in order to generate the appropriate URLs in your applications. To configure this entry point use the ROOT_URL_PREFIX on page 402 element.

Topics in this section discuss considerations for load balancing GAS applications on multiple instances. The provided examples illustrate simple configurations for Internet Information Services and Apache Web servers.

• GAS requests on page 171
GAS requests

The secret to load balancing GAS requests involves equally dispatching the requests between the different GAS instances.

However, all requests are not the same. Some requests must go to a specific GAS instance. These requests can be recognized through the session identifier that is part of their URL. The session identifier represents a GAS session, and because a GAS session is known by a single GAS instance only, all requests that are relative to that session must be driven to that specific GAS instance.

Figure 39: Load Balancing GAS Requests

Session-bound requests that share the same session identifier are part of a specific GAS session. Of the various kinds of GAS applications, only two create a session: GDC and GWC applications. Since GWS applications don't create sessions, GWS requests can always be routed to any of the available GAS instances. Sites that serve only GWS applications do not have to deal with session-bound request issues. However, if a site serves all kinds of applications, it must be configured to serve session-bound requests only.

Managing the load balancing means handling the two types of requests:

- Load balancing sessionless requests among several GAS instances.
- Driving session-bound requests to the GAS instance that handles the session.

Sessionless request processing

Load balancing sessionless requests among several GAS instances does not require any particular knowledge of the incoming request.

Generally, the load balancing tool offers rules to do sessionless load balancing by itself. Depending on the tool, those rules may go from a simple round-robin algorithm to more sophisticated ones like choosing a server according to the available memory or to the time that it takes to respond to a network request.
Figure 40: Sessionless Request Processing

Figure 40: Sessionless Request Processing on page 172 shows the processing of a sessionless request. It takes the following steps:

1. A Web Service client tool sends a request to the load balancer server.
2. The load balancer server is configured to dispatch the requests over two GAS instances. It does not matter if those instances are installed on separate servers, or if they are on the same server configured to listen at different ports. The routing really depends only on the capabilities of the load balancing tool. In this example, it happens to choose the instance G1.
3. The request is forwarded to G1.
4. The response is returned to the load balancer server.
5. The response is returned to the client.

Session-bound request processing

With session-bound requests, more work has to be done in order to route session-bound requests to the GAS (G1) instance that handles the session.

For them, G1 must be retrieved from the data of the request. The load balancing tool does not record the currently active GAS sessions. Actually, it even doesn't know anything about GAS sessions at all. It offers other means to retrieve G1: either through a cookie sent along with the request that will contain the identity of G1, or through a part of the URL of the request that will link to G1. This piece of information is added to the request that will create the session. This starting-session request is sessionless, and can therefore be processed by any of the available GAS instances. However, once that instance has been chosen (G1 in our example), the piece of information to identify G1 will be added to the request and/or to its response in order to route following requests to G1.

The following paragraphs discuss the two kinds of configuration for the load balancing of session-bound requests: cookie-based and path-based. For each method, first the processing of the starting-session request is shown, and then the processing of the session-bound requests is shown.
Cookie-based starting-session request processing

Figure 41: Cookie-based starting-session request processing on page 173 shows the processing of a cookie-based starting-session request. It takes the following steps:

1. GDC (or a user agent on the behalf of GWC) sends a request to the load balancer server.
2. The load balancer server is configured to dispatch the requests over two GAS instances. In this example, it happens to choose the instance G1.
3. The request is forwarded to G1.
4. G1 creates the session S1.
5. The response is returned to the load balancer server.
6. A session cookie that holds the identity of G1 is added to the response so that following requests will also contain this cookie.
7. The response is returned to the client.
Cookie-based session-bound request processing

Figure 42: Cookie-based session-bound request processing on page 174 shows the processing of a cookie-based session-bound request. It takes the following steps:

1. GDC (or a user agent in behalf of GWC) sends a request that contains the cookie $C_1$ to the load balancer server.
2. The load balancer server, thanks to $C_1$, recognizes that the request must be routed to $G_1$.
3. The request is forwarded to $G_1$.
4. The response is returned to the load balancer server.
5. The response is returned to the client.

**Note:** Every request received by the load balancer server that contain $C_1$ will be routed to $G_1$, whether it is a session-bound or a sessionless request.
Path-based starting-session request processing

Figure 43: Path-based starting-session request processing on page 175 shows the processing of a path-based starting-session request. It takes the following steps:

1. GDC (or a user agent in behalf of GWC) sends a request to the load balancer server.
2. The load balancer server is configured to dispatch the requests over two GAS instances. Each of these instances must be configured so that URLs that it handles begin with a part that is unique between all GAS instances, whether the GAS instances are installed on the same machine or different ones. In this example, G₁ handles requests whose URLs begin with /GAS1, and G₂ handles requests whose URLs begin with /GAS2. In this example, it happens to choose the instance G₁.
3. Once G₁ has been chosen, the URL of the request is rewritten so that it begins with /GAS1. This prefix will be recognized by the GAS as the connector URI part of the URL. The GAS adds the connector URI to every URL that is part of the request responses.
4. The request is forwarded to G₁.
5. G₁ creates the session S₁.
6. The response is returned to the load balancer server.
7. The response is returned to the client.
Path-based session-bound request processing

Figure 44: Path-based session-bound request processing

Figure 44: Path-based session-bound request processing on page 176 shows the processing of a path-based session-bound request. It takes the following steps:

1. GDC (or a user agent in behalf of GWC) sends a request whose URL begins with /GAS1 to the load balancer server.
2. The load balancer server, thanks to the /GAS1 prefix, recognizes that the request must be routed to G1.
3. The request is forwarded to G1.
4. The response is returned to the load balancer server.
5. The response is returned to the client.

Note: Every request received by the load balancer server that begins with /GAS1, respectively /GAS2, will be routed to G1, respectively G2, whether it is a session-bound or a sessionless request.

Load Balancing Configuration Examples

Configuration examples depend on the tools available on the Web Server.

The following configuration examples are based on two sample architectures. The first architecture is configured as follows:
Sample architecture 1

Figure 45: Load Balancing Sample Architecture 1

- Three machines.
- The load balancer machine contains all the tools to do the load balancing.
- Two machines each contain a GAS instance. Each GAS instance listens to the default HTTP port and are configured on the same virtual directory.
- The host names of the machines that contain the GAS instances are GAS1.corporate.com and GAS2.corporate.com.

The second architecture is configured as follows:

Sample architecture 2

Figure 46: Load Balancing Sample Architecture 2

- It is the same configuration as Figure 45: Load Balancing Sample Architecture 1 on page 177, but the load balancer server is configured on the gas virtual directory and the GAS instances are configured on distinct virtual directories.
- Internet Information Services 5.x and 6.0 on page 178
- Internet Information Services 7.x on page 179
- Apache 1.3.x and 2.0.x on page 181
- Apache 2.2.x on page 182
Internet Information Services 5.x and 6.0
Internet Information Services (IIS) 5.x and 6.0 have no built-in tools to do load balancing. However, third party tools do exist.

One third-party tool is ISAPI_Rewrite, which aims to enable Apache mod_rewrite on IIS.

Sessionless requests

The following example shows how to configure the ISAPI_Rewrite filter to do load balancing for sessionless requests. It assumes the architecture illustrated by the sample architecture 1.

```
# Helicon ISAPI_Rewrite configuration file
# Version 3.1.0.66

RewriteEngine on
RewriteMap servers rnd:hosts.txt
RewriteRule ^/(.*) http://${servers:host}/$1 [P,L]
```

- These configuration entries can be put at the server level in the httpd.conf file.
- The RewriteMap instruction uses a round-robin algorithm to select a host based on the content of the hosts.txt file, which has following content:

  ```
  host GAS1.corporate.com|GAS2.corporate.com
  ```

- The RewriteRule instruction replaces the server host of incoming requests by the content of the servers: host variable, then forwards the request to the chosen machine.

For example, the http://localhost/ws/r/echo URL will be rewritten to http://GAS1.corporate.com/ws/r/echo, assuming that the GAS1.corporate.com server has been chosen.

Session-bound requests

The following example shows how to configure the ISAPI_Rewrite filter to do load balancing for session-bound requests. It assumes the architecture illustrated by the sample architecture 2. ISAPI_Rewrite doesn't support cookie-based request routing. The example illustrates path-based session-bound request routing.

```
# Helicon ISAPI_Rewrite configuration file
# Version 3.1.0.66

RewriteEngine on
RewriteMap servers rnd:vdirs.txt
RewriteRule ^/gas/(.*) /${servers:vdir}/$1
RewriteRule ^/GAS1/(.*) http://GAS1.corporate.com/GAS1/$1 [P,L]
```

- These configuration entries can be put at the server level in the httpd.conf file.
- The RewriteMap instruction uses a round-robin algorithm to set the value of the servers: vdir variable based on the content of the vdirs.txt file, which has following content:

  ```
  vdir GAS1|GAS2
  ```

- For starting-session requests, the RewriteRule instruction replaces the gas part of the URL by the content of the servers: vdir variable, which is either GAS1 or GAS2; then other rewrite rules will be applied to set the host. For session-bound requests, the rule will not match so the rule doesn't apply.
- The host name of requests who's URL begin with /GAS1/ will be set to GAS1.corporate.com.
• The host name of requests who's URL begin with /GAS2/ will be set to GAS2.corporate.com.

For example, the http://localhost/gas/ua/r/gwc-demo URL will be rewritten to http://localhost/GAS2/ua/r/gwc-demo, assuming that the servers:vdir variable contains GAS2 at this time, then will be rewritten to http://GAS2.corporate.com/GAS2/ua/r/gwc-demo.
Likewise, the http://localhost/GAS2/wa/sua/93837374/1 URL will be rewritten to http://GAS2.corporate.com/GAS2/wa/sua/93837374/1.

Internet Information Services 7.x
Internet Information Services (IIS) 7.x provides load balancing tools through IIS extensions.

The IIS extensions are:
• Application Request Routing
• IIS URL Rewrite

Both tools are required to do the job.

The following examples show the resulting configuration entries. It can be done either by editing the configuration files manually or by using the IIS manager user interface. For more information on how to use the IIS manager user interface, see HTTP Load Balancing using Application Request Routing.

Sessionless requests
The following example shows how to configure IIS 7.x to do load balancing for sessionless requests. It assumes the architecture illustrated by the sample architecture 1.

<configuration>
...<system.webServer>...
<rewrite>
<globalRules>...
  <rule name="ARR_GASFarm_loadbalance" enabled="true" patternSyntax="Wildcard" stopProcessing="true">  
    <match url="*" />
    <conditions />
    <action type="Rewrite" url="http://GASFarm/{R:0}" />
  </rule>
</globalRules>
</rewrite>

<proxy enabled="true" />
</system.webServer>
...
<webFarms>
...
<webFarm name="GASFarm" enabled="true"><server address="GAS1.corporate.com" enabled="true"><applicationRequestRouting weight="100" />
</server>
<server address="GAS2.corporate.com" enabled="true"><applicationRequestRouting weight="100" />
</server>
<applicationRequestRouting><loadBalancing algorithm="WeightedRoundRobin" />
<affinity useCookie="false" />
</applicationRequestRouting>
</webFarm>
</webFarms>
...
• These configuration entries can be put at the server level in the applicationHost.config file.
• In the globalRules section of the rewrite section, the rule named ARR_GASFarm_loadbalance tells the module to rewrite all incoming requests so that they are routed to the GASFarm web farm.
• The proxy is enabled in order to forward the requests to the two GAS servers.
• The web farm named GASFarm is declared with the two GAS servers.

For example, the http://localhost/ws/r/echo URL will be rewritten to http://GAS1.corporate.com/ws/r/echo, assuming that the GAS1.corporate.com server has been chosen.

Session-bound requests

The following example shows how to configure IIS 7.x to do load balancing for session-bound requests. It assumes the architecture illustrated by the sample architecture 1. The example illustrates cookie-based request routing.

```xml
<configuration>
  ...
  <system.webServer>
    ...
    <rewrite>
      <globalRules>
        ...
        <rule name="ARR_GASFarm_loadbalance" enabled="true" patternSyntax="Wildcard" stopProcessing="true">
          <match url="*" />
          <conditions />
          <action type="Rewrite" url="http://GASFarm/{R:0}" />
        </rule>
        </globalRules>
      </rewrite>
      <proxy enabled="true" />
    </system.webServer>
    ...
    <webFarms>
      ...
      <webFarm name="GASFarm" enabled="true">
        <server address="GAS1.corporate.com" enabled="true">
          <applicationRequestRouting weight="100" />
        </server>
        <server address="GAS2.corporate.com" enabled="true">
          <applicationRequestRouting weight="100" />
        </server>
        <applicationRequestRouting>
          <loadBalancing algorithm="WeightedRoundRobin" />
          <affinity useCookie="true" />
        </applicationRequestRouting>
      </webFarm>
      ...
    </webFarms>
  ...
  </configuration>
```

• These configuration entries can be put at the server level in the applicationHost.config file.
• In the globalRules section of the rewrite section, the rule named ARR_GASFarm_loadbalance tells the module to rewrite all incoming requests so that they are routed to the GASFarm web farm.
• The proxy is enabled in order to forward the requests to the two GAS servers.
• The web farm named GASFarm is declared with the two GAS servers.
• The useCookie attribute of the affinity element is set to true; actually, this is the only difference compared to the sessionless request routing.

For example, the http://localhost/ua/r/gwc-demo URL will be rewritten to http://GAS1.corporate.com/ua/r/gwc-demo, assuming that the GAS1.corporate.com server has been chosen.

**Apache 1.3.x and 2.0.x**

Apache HTTP Server versions 1.3.x and 2.0.x provide load balancing by using mod_rewrite Apache module.

Refer to the following:

- Apache 1.3: Module mod_rewrite: URL Rewriting Engine
- Apache 2.0: Module mod_rewrite

All configurations examples explained here work on both Apache version 1.3.x and 2.0.x

### Sessionless requests

The following example shows how to configure the Apache mod_proxy module to do load balancing for sessionless requests. It assumes the architecture illustrated by the sample architecture 1.

```apachectl
# Apache configuration file
<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteMap servers rnd:hosts.txt
  RewriteRule ^/(.*) http://${servers:host}/$1 [P,L]
</IfModule>
```

- These configuration entries can be put at the server level in the httpd.conf file.
- The RewriteMap instruction uses a round-robin algorithm to select a host based on the content of the hosts.txt file, which has following content:

  ```
  host GAS1.corporate.com|GAS2.corporate.com
  ```

- The RewriteRule instruction replaces the server host of incoming requests by the content of the servers:host variable, then forwards the request to the chosen machine.

For example, the http://localhost/ws/r/echo URL will be rewritten to http://GAS1.corporate.com/ws/r/echo, assuming that the GAS1.corporate.com server has been chosen.

### Session-bound requests

The following example shows how to configure the Apache mod_proxy module to do load balancing for session-bound requests. It assumes the architecture illustrated by the sample architecture 2. The example illustrates path-based session-bound request routing.

```apachectl
# Apache configuration file
<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteMap servers rnd:vdirs.txt
  RewriteRule ^/gas/(.*) /${servers:vdir}/$1 [P,L]
  RewriteRule ^/GAS1/(.*) http://GAS1.corporate.com/GAS1/$1 [P,L]
</IfModule>
```

- These configuration entries can be put at the server level in the httpd.conf file.
- The RewriteMap instruction uses a round-robin algorithm to select a host based on the content of the vdirs.txt file, which has following content:

  ```
  host GAS1.corporate.com|GAS2.corporate.com
  ```

- The RewriteRule instruction replaces the server host of incoming requests by the content of the servers:vdir variable, then forwards the request to the chosen machine.
• These configuration entries can be put at the server level in the httpd.conf file.
• The RewriteMap instruction uses a round-robin algorithm to set the value of the servers:vdir variable based on the content of the vdirs.txt file, which has following content:

  vdir GAS1|GAS2

• For starting-session requests, the RewriteRule instruction replaces the gas part of the URL by the content of the servers:vdir variable, which is either GAS1 or GAS2; then other rewrite rules will be applied to set the host. For session-bound requests, the rule will not match so the rule doesn’t apply.
• The host name of requests who’s URL begin with /GAS1/ will be set to GAS1.corporate.com.
• The host name of requests who’s URL begin with /GAS2/ will be set to GAS2.corporate.com.

For example, the http://localhost/gas/ua/r/gwc-demo URL will be rewritten to http://localhost/GAS1/ua/r/gwc-demo, assuming that the servers:vdir variable contains GAS2 at this time, then will be rewritten to http://GAS2.corporate.com/GAS2/ua/r/gwc-demo. Likewise, the http://localhost/GAS2/ua/sua/93837374/1 URL will be rewritten to http://GAS2.corporate.com/GAS2/ua/sua/93837374/1.

Apache 2.2.x
Apache HTTP Server versions 2.2.x provides load balancing tools through two Apache modules

The two modules are:
• Apache 2.2: Module mod_proxy_balancer
• Apache 2.2: Apache Module mod_headers

The mod_proxy_balancer module provides support of HTTP requests load balancing. The mod_headers module provides HTTP cookies management needed by the Cookie-based request processing. For Sessionless request processing, the mod_headers module is not required.

Sessionless requests
The following example shows how to configure Apache 2.2.x to do load balancing for sessionless requests. It assumes the architecture illustrated by Session-bound request processing on page 172.

```
<IfModule mod_proxy.c>
  ProxyPass / balancer://GASFarm
  <Proxy balancer://GASFarm>
   BalancerMember http://GAS1.corporate.com
   BalancerMember http://GAS2.corporate.com
  </Proxy>
</IfModule>
```

• These configuration entries can be put at the server level in the httpd.conf file.
• The ProxyPass instruction tells the module to rewrite all incoming requests so that they are routed to the GASFarm web farm.
• The <Proxy balancer://GASFarm>... </Proxy> block defines which are the members of GASFarm web farm.
• The BalancerMember instructions declare two HTTP web server members of GASFarm web farm: GAS1.corporate.com and GAS2.corporate.com.

For example, the http://localhost/ws/r/echo URL will be rewritten to http://GAS1.corporate.com/ws/r/echo, assuming that the GAS1.corporate.com server has been chosen.
Session-bound requests

The following example shows how to configure Apache 2.2.x to do load balancing for session-bound requests. It assumes the architecture illustrated by Session-bound request processing on page 172. The example illustrates cookie-based request routing.

```html
<IfModule mod_proxy.c>
  ProxyPass / balancer://GASFarm stickysession=GAS_AFFINITY
  <Proxy balancer://GASFarm>
    BalancerMember http://GAS1.corporate.com route=GAS1
    BalancerMember http://GAS2.corporate.com route=GAS2
  </Proxy>
</IfModule>

<IfModule mod_headers.c>
  Header add Set-Cookie "GAS_AFFINITY=balancer.%{BALANCER_WORKER_ROUTE}e; path=/; domain=.corporate.com" env=BALANCER_WORKER_ROUTE
</IfModule>

• These configuration entries can be put at the server level in the httpd.conf file.
• The ProxyPass instruction tells the module to rewrite all incoming requests so that they are routed to the GASFarm web farm. The stickysession parameter gives the name of the cookie to use to retrieve the GAS that holds the session.
• The <Proxy balancer://GASFarm>...</Proxy> block declares which are the members of GASFarm web farm.
• The BalancerMember instructions declare two HTTP web server members of the GASFarm web farm: GAS1.corporate.com and GAS2.corporate.com. The route parameter gives the name of the route associated with that member. If that route name is found in the request cookie, then that member will be chosen.
• The Header instruction will set, in the HTTP response, the cookie named GAS_AFFINITY to value balancer.BALANCER_WORKER_ROUTE. The BALANCER_WORKER_ROUTE variable is assigned to the route of the GASFarm member that has been chosen for the current request. Lastly, the env parameter tells the module to set that cookie only if the BALANCER_WORKER_ROUTE variable is defined.

For example, the http://localhost/ua/r/gwc-demo URL will be rewritten to http://GAS1.corporate.com/ua/r/gwc-demo, assuming that the GAS1.corporate.com server has been chosen.
Developing Web applications

The Genero Application Server allows you to deliver Web applications, as well as Web services. Learn how the Genero Web Client for JavaScript works and get started with ways for customizing the user interface for your GWC-JS applications.

- Genero Web Client for JavaScript (GWC-JS) on page 184
- Accessing Genero Web Services on page 253

Genero Web Client for JavaScript (GWC-JS)

These topics provide information about the Genero Web Client for JavaScript (GWC-JS) client.

Developing and deploying Web applications requires you to configure the Genero Application Server to launch the application. Existing applications may require some slight modifications in order to work properly given the limitations of what an application can do from a browser. A general knowledge of how the Genero Web Client for JavaScript operates can be helpful in the planning and deploying of Web applications.

Tip: The Genero Web Client for JavaScript (GWC-JS) was initially referred to as the Genero Browser Client. As such, the product id of "gbc" is often seen in file and directory names.

- What is GWC-JS? on page 184
- GWC-JS application bootstrap on page 202
- GWC-JS applications and use of cookies on page 202
- Customizing GWC-JS applications on page 203
- Migrating from GDC to GWC-JS on page 245

What is GWC-JS?

The Genero Web Client for JavaScript (GWC-JS) delivers your applications over the Web using the latest technologies for Web application development.

The GWC-JS is a JavaScript client that works with the well-known and widely-used frameworks like node.js and sass. Its UI is adapted from concepts of material design inspired by recommendations from Google’s material design Web site. This innovative design approach opens up limitless possibilities for you to customize the GWC-JS features and styles according to your requirements.

The GWC-JS allows you to deliver true Web applications developed in the Genero Business Development Language (BDL). With the underlying source written in Genero BDL, the GWC-JS is flexible enough to let you build a full range of Web applications, from simple to corporate applications, with few limitations on what you can achieve.

GWC-JS brings Genero applications to the Internet world with the capacity of integration in a Web site. It provides you with the opportunity of working with state of the art Web technologies.

Why deliver an application as a Web application?

- Web application deployment is easier and cheaper than desktop application deployment.
- The end user requires a browser; no software needs to be installed on the client by the end user.

GWC-JS principles

GWC-JS has evolved from the earlier Genero Web Client for HTML5. The earlier client used templates and snippet sets to create dynamic Web pages that were rendered on the Application Server for delivery to the client browser. The GWC-JS on the other hand is a JavaScript client that works with Node technology. It supports Genero real-time Web applications based on the following principles:
• Nothing is computed or rendered on the Application Server side.
• Genero applications are rendered based on DVM instructions (not HTML).
• GWC-JS interprets the same protocol as Genero Desktop Client (GDC) to build Web interfaces.
• GWC-JS project sources (JavaScript, HTML and CSS) are provided in the GAS package. You can modify and adapt these to customize your application’s features and styles to meet your needs, see Customizing GWC-JS applications on page 203.

GWC-JS can deliver an application to any device as long as your browser supports HTML5.

• Key Players on page 185
• How GWC-JS works on page 185
• Features and limitations on page 187
• Overview of the GWC-JS interface on page 188
• GWC-JS application bootstrap on page 202
• GWC-JS applications and use of cookies on page 202

Key Players
When working with applications deployed in a Web environment, you will need to identify or add team members who are proficient in various Web technologies. Many of these technologies will be unfamiliar to your traditional Genero BDL application developer.

The key players involved in developing Genero Web Client for JavaScript (GWC-JS) applications are listed by role:

Table 15: Web application development key players

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front-end Web developer</td>
<td>Responsible for the customization of aspects of the application within GWC-JS by adding JavaScript and HTML code and modifying CSS using Sass to influence the look-and-feel of the application.</td>
</tr>
<tr>
<td>Application Developer</td>
<td>Responsible for the development of Genero applications, concentrating on the business logic. Members of this team are proficient in the Genero BDL language. They also support the customization work of the front-end Web developers.</td>
</tr>
</tbody>
</table>

It is rare that a single person fulfills the requirements demanded in each of these areas.

If you are working with the GWC-JS, you must have some understanding of Web technologies like HTML, XML, style sheets, Node.js, Sass, Grunt, and JavaScript. You can find Web standards at the World Wide Web Consortium (W3C) site.

Tip: For basic tutorials about Web standards, visit http://www.w3schools.com. While the w3schools site provides basic, free tutorials, it is a private venture and not affiliated with the W3C. It provides a starting point for learning new Web technologies.

How GWC-JS works
Knowing a little of how the Genero Web Client for JavaScript (GWC-JS) uses AUI tree, controllers and widgets to create and manage UI elements will help you on your way to building a customized UI interface.

How GWC-JS works
The GWC-JS Web client interface is built dynamically using JavaScript. When the GWC-JS is launched, it receives an AUI tree from the GAS. Effectively each node of the AUI tree is managed by a GWC-JS controller and widget.

The GWC-JS uses its controller and widget elements to create the HTML DOM. Through DOM elements, JavaScript is able to dynamically implement the elements of the HTML pages of the UI by responding to
changes to their properties and listening for events that change their behavior. See this illustrated in Figure 47: GWC-JS Controller and Widget Function on page 186.

**Figure 47: GWC-JS Controller and Widget Function**

**GWC-JS Controller**

The GWC-JS controller and widget elements are interdependent but have specific and distinct roles. The JavaScript controller (`Controller.js`) plays a role in:

- Creating the widget.
- Maintaining its behaviors.

Behaviors involves mapping one or more attributes of the AUI tree to an aspect of the widget. For example, the `color` and `reverse` attributes control a widget's background color. To reflect the two-way interaction between the Web client and the application, there are two types of behavior that a controller needs to maintain:

- **VM behavior**: the controller is listening to the DVM and applying the changes received from DVM instructions.
- **UI behavior**: the controller is also listening to the UI and sending UI modification to the DVM.

  **Note:** These built-in behaviors are internal and not meant to be modified by users.

**GWC-JS Widget**

The JavaScript widget (`Widget.js`) plays a role in updating the DOM tree to reflect the application state. It has:

- APIs to manipulate the DOM.
- Listeners to react to DOM events.

  **Understanding GWC-JS widgets** on page 209
Features and limitations
Knowing the features and limitations of the different Genero front-ends help you determine which front-end is the most suitable for your applications.

While a Genero application largely reacts the same across different front-ends (Genero Web Client for JavaScript (GWC-JS), Genero Desktop Client (GDC), and so on), there are limitations for applications.

Legend
• X : available
• X* : available with limitations
• NA : not available

Table 16: Rendering: Features and limitations for Genero front-ends

<table>
<thead>
<tr>
<th>Look</th>
<th>GDC</th>
<th>GWC-HTML5</th>
<th>GWC-JS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>windows</td>
<td>custom</td>
<td>custom</td>
</tr>
<tr>
<td></td>
<td>manager theme</td>
<td>theme</td>
<td>theme</td>
</tr>
<tr>
<td>Layout</td>
<td>X</td>
<td>X*</td>
<td>X*</td>
</tr>
<tr>
<td>MDI windows</td>
<td>X</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Stacked window</td>
<td>X</td>
<td>X*</td>
<td>X</td>
</tr>
<tr>
<td>Styles: .4st colors, fonts (common styles)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Styles: .4st positioning</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Widgets Customization</td>
<td>NA</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Web Components</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Traditional GUI Mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 17: Behavior: Features and limitations for Genero front-ends

<table>
<thead>
<tr>
<th>Synchronous triggers (predefined actions completed together)</th>
<th>GDC</th>
<th>GWC-HTML5</th>
<th>GWC-JS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Drag and Drop</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Keying: dialogtouched</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Keying: Type ahead</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Keying: Accelerator keys</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Keying: Local actions</td>
<td>X</td>
<td>X*</td>
<td>X*</td>
</tr>
<tr>
<td>Natural accelerator</td>
<td>X</td>
<td>X*</td>
<td>X*</td>
</tr>
<tr>
<td>Cursor: fgl_dialog_getselectionend</td>
<td>X</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>Cursor: fgl_dialog_setselection</td>
<td>X</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>Cursor: fgl_dialog_getcursor</td>
<td>X</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>Cursor: fgl_dialog_setcursor</td>
<td>X</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>Cursor: fgl_getcursor</td>
<td>X</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>Widgets</td>
<td>X</td>
<td>X*</td>
<td>X*</td>
</tr>
<tr>
<td>Feature</td>
<td>GDC</td>
<td>GWC-HTML5</td>
<td>GWC-JS</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>RIP widgets</td>
<td>X</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Table: Resizing columns</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Table: Display/Hide columns</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Table: Frozen columns</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Dockable toolbars/menus</td>
<td>X</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Genero Report Viewer</td>
<td>X</td>
<td>NA</td>
<td>X</td>
</tr>
<tr>
<td>CANVAS</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Richtext</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PictureFlow</td>
<td>X</td>
<td>X</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 18: Interaction (with third party): Features and limitations for Genero front-ends

<table>
<thead>
<tr>
<th>Feature</th>
<th>GDC</th>
<th>GWC-HTML5</th>
<th>GWC-JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>File transfer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Session variables/Cookies</td>
<td>NA</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Stored settings</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>front-end Call</td>
<td>X</td>
<td>X*</td>
<td>X*</td>
</tr>
</tbody>
</table>

Table 19: Deployment: Features and limitations for Genero front-ends

<table>
<thead>
<tr>
<th>Feature</th>
<th>GDC</th>
<th>GWC-HTML5</th>
<th>GWC-JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>front-end System</td>
<td>All / Internet Explorer</td>
<td>HTML5 Browsers</td>
<td>HTML5 Browsers</td>
</tr>
<tr>
<td>Connections: HTTP / HTTPS</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Connections: rlogin / telnet / SSH / SSH2</td>
<td>X*</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Single Sign On</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Hot key support

When using the Genero Web Client for JavaScript, some hot key combinations are not supported by specific browsers; they are trapped by the browser instead. The behavior is not consistent amongst the available browsers, and Four Js is unable to provide accurate and up-to-date information regarding hot key support for each supported browser.

To make an action available for when a hot key combination is not supported, you must make the action visible. For example, you can add the action to the action panel. This allows the user to click on a button when the hot key combination is not working.

Overview of the GWC-JS interface

When you launch a Genero Web Client for JavaScript (GWC-JS) application, it is displayed in the browser tab within the user interface. The default interface provides panels for the application, a side bar, and a toolbar.

- Quick Start: Tour of GWC-JS interface on page 189
- GWC-JS home page on page 194
Quick Start: Tour of GWC-JS interface
Use this tour to quickly become familiar with the Genero Web Client for JavaScript (GWC-JS) client interface while exploring the demo applications.

- Quick start: stacked windows on page 189
- Quick start: run without waiting on page 190

Quick start: stacked windows
The Genero Web Client for JavaScript (GWC-JS) user interface provides a mechanism for working with applications that use stacked windows.

This quick start guide provides you with the steps to explore working with stacked windows in the demos Web application using GWC-JS. Imagine an application’s window stack is composed of several windows, W1, W2, ..., Wn. Typically, the GWC-JS only displays the window at the top of the stack - the Wn window; also called the current window. The other windows, W1 to Wn-1, are not displayed but you can access them from the side bar view.

Before you begin:
Start the standalone dispatcher from the command line using httpdispatch and then open the demos application in your browser by entering the URL http://localhost:6394/ua/r/gwc-demo.

1. In the Topic tree of the demo directory, navigate to User Interface >> UI Basics>> Window. Double-click on the demo CurrentWindow1 in the panel to the right.

Figure 48: Application windows in GWC-JS user interface

- The window at the top of the stack (Window W3) is displayed.
- The side bar or navigation panel is updated with the CurrentWindow1 application tree, which has the following links to windows in the stack:
  - Screen
  - Window w1
2. To set an active window, click on one of the Test menu buttons:
   The menu button options are:
   - w1
   - w2
   - w3
   - Screen

   The Is current field displays YES for the current window, and the window name is displayed in the application title bar.

3. In the side bar panel, click on one of the links in the CurrentWindow1 tree to set the focus to another window in the stack.

   Note: You can see that although you can switch to windows other than the current one in the CurrentWindow1 stack, their functions are disabled and their menus invisible. In this example, you only have one current window in the stack.

   The result of your window selection will depend on the following:
   - If the selected window is the current window, the Is current field value is YES and the Test menu panel is displayed.
   - If the selected window is not current, the Is current field value is NO and the Test menu panel is not displayed.

What to do next

Explore the mechanism for running Web applications in the background and switching between these as described in Quick start: run without waiting on page 190.

Quick start: run without waiting

The Genero Web Client for JavaScript (GWC-JS) provides a mechanism for applications to be run as child programs that execute in the background, that is the parent program can continue to run without waiting for the child program to finish.

About this task:

The example BDL program (Navigation.42m) starts an application with stacked windows. It gives you the option to use a RUN WITHOUT WAITING command to start another instance of the application as a background process.

1. Copy the sample BDL program code to a text file and save it as Navigation.4gl

   The program's main code block opens four windows and displays form menus.

```plaintext
# Property of Four Js*
# (c) Copyright Four Js 1995, 2015. All Rights Reserved.
# * Trademark of Four Js Development Tools Europe Ltd
# in the United States and elsewhere
#
# Four Js and its suppliers do not warrant or guarantee that these
# samples are accurate and suitable for your purposes. Their inclusion is
# purely for information purposes only.

MAIN
  DEFINE cw STRING
  DEFINE i INT

  IF num_args() = 0 THEN LET i=1 ELSE LET i=arg_val(1) END IF

  OPEN FORM f FROM "Navigation"
  DISPLAY FORM f
```
DISPLAY "screen" TO fname

OPEN WINDOW w1 WITH FORM "Navigation"
CALL fgl_setTitle("Window w1 - " || i)
DISPLAY "w1" TO fname

OPEN WINDOW w2 WITH FORM "Navigation"
CALL fgl_setTitle("Window w2 - " || i)
DISPLAY "w2" TO fname

OPEN WINDOW w3 WITH FORM "Navigation"
CALL fgl_setTitle("Window w3 - " || i)
DISPLAY "w3" TO fname

WHILE
  DISPLAY "YES" TO info
  MENU "Test"
    COMMAND "w1" LET cw="w1" EXIT MENU
    COMMAND "w2" LET cw="w2" EXIT MENU
    COMMAND "w3" LET cw="w3" EXIT MENU
    COMMAND "screen" LETcw="screen" EXIT MENU
    COMMAND "RWW" RUN "fglrun Navigation " || i+1 WITHOUT WAITING
    COMMAND KEY(] INTERRUPT) "Exit" EXIT PROGRAM
  END MENU
  DISPLAY "NO" TO info
  CASE cw
    WHEN "w1" CURRENT WINDOW IS w1
    WHEN "w2" CURRENT WINDOW IS w2
    WHEN "w3" CURRENT WINDOW IS w3
    OTHERWISE CURRENT WINDOW IS SCREEN
  END CASE
  DISPLAY "YES" TO info
END WHILE
END MAIN

Note: The "RWW" menu command contains the instruction which allows you to start another instance of the application:

COMMAND "RWW" RUN "fglrun Navigation " || i+1 WITHOUT WAITING

2. Copy the program's form specification code to a text file and save it as Navigation.per

# Property of Four Js*
# (c) Copyright Four Js 1995, 2015. All Rights Reserved.
# * Trademark of Four Js Development Tools Europe Ltd
#   in the United States and elsewhere
#
# Four Js and its suppliers do not warrant or guarantee that these
# samples are accurate and suitable for your purposes. Their inclusion is
# purely for information purposes only.

LAYOUT
GROUP(text="Test Current Window")
GRID
  Window name [fw         ]
  Is current [info       ]
END

ATTRIBUTES
  LABEL fw=formonly.fname;
  EDIT info=formonly.info;
3. At the command line, compile the source code modules you have created (Navigation.4gl and Navigation.per) by typing the following commands:

Run:

```
fglcomp Navigation.4gl
fglform Navigation.per
```

Compiled files are created for the source code module and the form:

```
Navigation.42m
Navigation.42f
```

4. Create an application configuration file.

Use a text editor or if you are using Studio, go to File >> New >> Web/AS >> Application Configuration (.xcf)

Provide an absolute path to the location of your compiled application file in the PATH element, and in the MODULE element specify that Navigation.42m is the module required to launch your application.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH><path_to_your_local_directory></PATH>
    <MODULE>Navigation.42m</MODULE>
  </EXECUTION>
  <UA_OUTPUT>
    <GWC-JS>gwc-js</GWC-JS>
  </UA_OUTPUT>
</APPLICATION>
```

5. Save the configuration file (e.g. Navigation.xcf) in your $(res.appdata.path)/app directory.

6. Start the GAS standalone dispatcher from the command line using httpdispatch and open the Navigation application in your browser by entering the URL:

```
http://localhost:6394/ua/r/Navigation
```

**Figure 49: Current Application Window in GWC-JS**
• The current window at the top of the stack (W3) is displayed in the GWC-JS user interface. The current window's menu button options are:
  - w1
  - w2
  - w3
  - Screen
  - RWW
  - Exit
• The Window name field displays W3 for the current window.
• The Is current field displays YES for the current window.
• The side bar panel is updated with the Navigation tree, which has the following links to windows in the stack:
  - Screen
  - Window w1 - 1
  - Window w2 - 1
  - Window w3 - 1

7. To have another instance of the Navigation application running at the same time:
   In the current window's Test menu panel, select the RWW menu button.
   The side bar view is updated with another Navigation tree.

Figure 50: Two Instances of Application Running in GWC-JS
8. From the side bar view, select the current windows in each running instance of the Navigation application, for example complete the following steps:

a) Select Window w3 - 1
b) Select Window w3 - 2

You should see that both applications have current windows. The first instance of the Navigation application you started (the parent program) continues to run without waiting for the second instance of Navigation (the child program) to finish. The result of your selection will depend on the following:

- If the selected window is the current window, the Is current field value is YES and the Test menu panel is displayed.
- If the selected window is not current, the Is current field value is NO and the Test menu panel is not displayed.

GWC-JS home page
The GWC-JS home page displays when launched or when applications are closed. From the home page, you can open or replay applications.

To open the GWC-JS user interface, type the URL into your browser as follows:
http://host:port/gwc-js/index.html, where gwc-js is the name of the front-end directory or a symbolic link to it. For example working on the standalone dispatcher as the Web server, enter http://localhost:6394/gwc-js/index.html.

Views
The home page has two panels:

History
Lists all recently-opened applications. Click on an application to open it. Applications are automatically added to the view. Remove an application from the history list by clicking on the delete icon for the application.

Bookmarks
Lists all bookmarked application. Click on an application to open it. Remove an application from the bookmark list by clicking on the delete icon for the application.
Features

Log Player

The Log Player feature allows you to replay an application using its session log. Click on the Run button next to the Log Player to begin the Log Player. See Replay a log with Log Player on page 201.

Launch an application

Run an application that is configured on your GAS by typing its name in the UA Url launcher field and clicking Run. For example, type gwc-demo to run the demo application.

Figure 51: GWC-JS user interface home page

- GWC-JS side bar on page 199

GWC-JS application page

The Genero Web Client for JavaScript (GWC-JS) interface is designed to display your applications. Knowing its views and icons will help your work with applications.

Views

The application page has two main panels:

Main container

The current application is displayed in this panel. You can expand or contract the panel by moving the divider between it and the side bar.

Side bar

Lists all open applications.

Note: Depending on the width of your browser window, you may not see the side bar. If the side bar is not displayed, you can access it from an icon (commonly called the "hamburger icon") on the toolbar/title panel. See GWC-JS side bar on page 199.
Standard Icons

When an application is open, icons are shown in the toolbar/title panel at the top of the application page.

**Figure 53: Application icons**

**Application information**

Open the **Product identification** window to display information about the GWC-JS version.

**Settings**

Open the **Settings** window to select a language for the user interface.

**Bookmark**

Bookmark the current application. The **Bookmark** view in the home page lists the bookmarks you have added. See GWC-JS home page on page 194.

**Close window**

Close the current application window. Selecting an application in the **side bar** panel activates the **Close window** icon for that application.

Clicking on the **Close window** icon closes the selected application and returns you to:

- The next open application on the **side bar** list.
- If it is the last application in the **side bar** list, you are returned to **The application ended** page. See GWC-JS application ended page on page 198.

**Note:** The **Close window** icon may not be shown while an application is expecting user input or other action.
GWC-JS application in debug mode
The Genero Web Client for JavaScript (GWC-JS) interface provides debug options for troubleshooting applications.

![GWC-JS user interface debug icons](image)

**Figure 54: GWC-JS user interface debug icons**

Running a GWC-JS app in debug mode
Debug mode can be activated by:

- Configuring the Genero Application Server to run in development mode with the `--development` flag. This enables debug mode for all launched applications. See [Configure development environment](#) on page 156.
- Adding the query string parameter "debugMode=1" to the URL. For example: `http://localhost:6394/ua/r/HelloWorld?debugMode=1`
- Typing "gbc.DebugService.activate()" in the browser JavaScript console. The JavaScript console is provided by Web browsers as part of their developer tools. Once activated, the application remains in debug mode until the browser is refreshed.

Debug Icons
When the GWC-JS is running in debug mode, the debug icons are displayed along side the standard icons in the toolbar (see [GWC-JS application page](#) on page 195). Hovering your mouse over the icons shows text that describe their function.

![Debug icons](image)

**Figure 55: Debug icons**

**Debug Tools**

The **Debug tools** icon opens the AUI tree for the current application. See [View the AUI tree](#) on page 200.

**GWC**

The **Run in GWC-HTML5** icon opens the current application with the `wa` protocol. The application opens in a new tab.
**GDC**
The **Run in GDC** icon downloads a shortcut allowing you to open the current application with the Genero Desktop Client.

**PXY**
The **Proxy Logs** icon opens the current application session log in a new tab. Use this to view the status of proxies, responses sent, and system error messages. See **Logging** on page 167.

**DVM**
The **VM Logs** icon opens the Dynamic Virtual Machine (DVM) log for DVMs started by the application. The log opens in a new tab.

**GWC-JS application ended page**
The Genero Web Client for JavaScript (GWC-JS) interface application ended page is displayed when all open applications have closed. From this page you have an options to view logs or restart the closed application.

When you close applications using the **Close window** icon as opposed to closing the tab, the Genero Web Client for JavaScript (GWC-JS) interface exits to display the **application ended** page. The application ended page provides you with the following options:

**Session ID**
The session id is displayed. Use this to identify session logs for the closed application.

**Get the VM logs**
The **Virtual Machine log** link opens the Dynamic Virtual Machine (DVM) log for DVMs started by the application. The log opens in a new tab. See **Logging** on page 167.

**Get the proxy logs**
The **Proxy log** link opens the current application session log in a new tab. Use this to view the status of proxies, responses sent, and system error messages.

**Note:** The links to the VM and proxy logs are only visible in debug mode. See **GWC-JS application in debug mode** on page 197

**Restart the same application**
The **Restart the app** link restarts the closed application.

**Close**
The **Close** icon closes the **application ended** page and opens the home page. See **GWC-JS home page** on page 194.
Tip: The application end can be redirected to a site or page of your choice. See Redirect the application end on page 244.

GWC-JS side bar
Access your open applications in the Genero Web Client for JavaScript (GWC-JS) interface from a side bar panel.

The side bar lists all the applications that you have currently open in the user interface. You can switch between your open applications, or windows of an application if more than one, by selecting them from the list to make them active in the main application panel to the right.

Note: Depending on the width of your browser window, you may see a menu icon (commonly called the "hamburger icon") instead of the side bar. The icon consists of three horizontal stripes. Click on the side bar icon and the side bar opens, giving you access to all opened applications.

Figure 57: The side bar menu icon
When you close an application, it is removed from the side bar list.
View the AUI tree
Inspecting the application user interface (AUI) tree helps you in the debugging and testing of your Genero Web Client for JavaScript (GWC-JS) application.

To view the AUI tree, you need to be in debug mode. If you do not see the Debug Tools menu icon on the right-hand side of your Web application window, you are not in debug mode. See Configure development environment on page 156.

1. If the GWC-JS is running in debug mode, click on the Debug Tools icon.

The GWC-JS Debug tools page opens in a new browser tab. You see the AUI tree of the current application you have open in the GWC-JS browser tab.
2. Select a node of the AUI tree.
   The properties and values of those properties are shown in the panel to the right.

3. Close the GWC-JS Debug tools tab to exit the AUI tree.

Replay a log with Log Player
The Genero Web Client for JavaScript (GWC-JS) log player allows you to play back a session log in the browser.

You can use the log player feature to:

- debug your applications
- create a demo

What is logged? Only the interaction between the front-end and the DVM is recorded and replayed, so the runtime system is not needed when you replay your log.

**Important:** What is not logged?

- Images and resources are not taken into account during the replay.
- All user “local actions”, such as copy, paste, and so on, are not saved and replayed. For example, only the sent value of a field is saved.

**Tip:** The Genero Desktop Client (GDC) has a logging feature similar to the Log Player. For more information see the Using the Logging System topic in the Genero Desktop Client User Guide.

**Before you begin:** Set the CATEGORIES_FILTER on page 361 in the LOG element to ALL DEBUG in your GAS configuration file, FGLASDIR/etc/as.xcf. For example,

Change:

```
< CATEGORIES_FILTER > $(res.log.categories_filter) </CATEGORIES_FILTER >
```

To:

```
< CATEGORIES_FILTER > ALL DEBUG </CATEGORIES_FILTER >
```

If you see this message in the debug frame after opening the log file, set the categories filter.

Log fields missing. Enable ”ALL DEBUG” categories

If you make changes to the GAS configuration file, you need to restart the GAS by stopping and starting the dispatcher, Dispatcher: httpdispatch on page 304.

1. In the Genero Web Client interface, click on the Run button next to the Log Player.

2. Click Browse to choose the log file.
   To find the session logs, choose from the following options:

   - On Linux®/UNIX™, navigate to FGLASDIR/appdata/dispatcher_name/yyyyymmdd.
   - On Windows®, navigate to C:\ProgramData\vendor\gas\gas_version\log\dispatcher_name\yyyyymmdd

Where yyyyymmdd is the directory for the current day.

**Note:** Session log file names are prefixed with “uaproxy”, for example uaproxy-3d5af84d3ad846d6463dfb0d776bddd. The long string is the session ID, which uniquely identifies a session.
The "File Upload" window opens, allowing you to select the log file.

**Tip:** An application's log file is only available once you exit the application. Check the time the file was created to make sure you select the correct log.

3. Click on the play button.

In the main program frame, you can see the user interaction with the application replayed. In a frame below the main program, you can see here what was output to the session log as the application ran.

Other available options in the log player menu are:

1. The "Delay between orders" field indicates how long the log player waits between two DVM instructions, or "log steps". Change this delay with the **Delay between orders (ms)**. This can be used to speed up or slow down the replay.
2. The **Reset** icon allows you to restart the log.
3. The **Next Step** icon replays the application log one step at a time.
4. The **Forward** icon allows you to fast forwards through the log. You set the number of steps to "jump" ahead. Use this to go to a specific location of the log file when replaying.

**GWC-JS application bootstrap**

When a Genero Web Client for JavaScript (GWC-JS type) application starts, a bootstrap mechanism is called to initialize information.

To start a GWC-JS type application, type the URL of your application into your browser as follows:

http://host:port/gas/ua/r/group/myapp

As the application starts, a **bootstrap.html** file is called. This bootstrap page loads the required JavaScript files and provides information for rendering the application:

- application style sheet
- application icon
- JavaScript injectors.

The bootstrap is configured in the **GWC-JS** element of the **UA_OUTPUT** configuration element. See **UA_OUTPUT** on page 414.

The **bootstrap.html** file, including all the resources required for rendering the application, are located in the directory referenced by the **GWC-JS** element. The default directory is **gwc-js** located in $FGLASDIR/web/.

**GWC-JS applications and use of cookies**

GWC-JS applications use cookies to secure the communication with the Genero Application Server.

**Using cookies in GWS-JS applications**

You can use cookies in your applications if you save or exchange some data that you want to persist between different runs of an application. For example, Web applications that run on the same domain as your GAS and whose GWC cookie is set on the root path (/) of the URL are able to retrieve cookies in later requests for an application. When a request is made, the cookie is sent back from browsers to the GAS where it is recognized.

The GWC-JS uses local storage, which does not involve cookies, to store data specific to an application after the browser is closed. This data is restricted for reuse to the application URI.

**Note:** Browsers generally need cookies turned on.
Customizing GWC-JS applications

This section provides you with topics on how to customize the Genero Web Client for JavaScript (GWC-JS) user interface. It provides you with a customization overview and some procedures to get started with customizing.

Note: As development work on the GWC-JS continues, we will update this section with more customization options as new versions of the GWC-JS are released.

- Customization framework on page 203
- Project directory on page 205
- Understanding GWC-JS widgets on page 209
- Configure your environment on page 217
- Customizing the theme on page 227
- Add header and footer text on page 231
- Add an image to the application title on page 234
- Create widgets with the ModelHelper class APIs on page 213
- Add localized texts on page 243
- Compiling customization projects on page 219
- Apply customization to an application on page 223

Customization framework

The Genero Web Client for JavaScript (GWC-JS) is designed to be customized. It consists of a set of files that define its components, and tools, such as the grunt tool, that implement changes you make to the files to build the front-end.

The customization framework overview

The GWC-JS infrastructure consists of two components:

Project directory

The project directory (project_dir) contains JavaScript (js), template HTML (tpl.html) and sass stylesheets (scss) files used to create the front-end application. See Project directory on page 205.

Tools

The developer working on customization installs these third-party tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Project directories/files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node.js (<a href="http://nodejs.org/">http://nodejs.org/</a>)</td>
<td>Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes your GWC-JS Web applications lightweight and efficient.</td>
<td>\customization_project</td>
</tr>
<tr>
<td>Tool</td>
<td>Description</td>
<td>Project directories/files</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>git</td>
<td>Git looks after the version control for the project development, keeping track of the changes you make to your project.</td>
<td>n/a</td>
</tr>
<tr>
<td><a href="http://git-scm.com/">http://git-scm.com/</a></td>
<td>Node Package Manager (npm) keeps track of your project's packages and their dependencies in a manifest file.</td>
<td>package.json</td>
</tr>
<tr>
<td>Grunt</td>
<td>Grunt is the JavaScript task runner which builds the project with compilation options and automates the compilation and the reconfiguration of the GWC-JS client after customization.</td>
<td>Gruntfile.js</td>
</tr>
<tr>
<td>Sass</td>
<td>Sass is an extension of CSS that allows use of variables, mixins, and inline imports. The project directory contains a set of sass stylesheets that provide default definitions for all key HTML components and Genero application</td>
<td>\customization_project\</td>
</tr>
</tbody>
</table>
Developing Web applications | 205

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Project directories/files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>elements: such as forms, buttons, navigation and other interface components. By editing these styles in the scss files, you customize the components in your project.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note: The open source text editor Brackets (<a href="http://brackets.io/">http://brackets.io/</a>) makes working with scss and json files much easier.</td>
<td></td>
</tr>
</tbody>
</table>

The setup for customization involves these steps

1. Installing third-party tools. See Install tools on page 216
2. Extracting sources from the GWC-JS project zip file and creating your project directory. See Configure your environment on page 217
3. Creating a customization project, see Customization Project on page 207.
4. Customizing the user interface, see Customize the User Interface on page 226.
5. Linking the project distribution (project_dir/dist/customization/custom_project_dir) to your $FGLASDIR/web/<gwc-dev>.
6. Configuring your apps to use your customized GWC-JS front-end, see Apply customization to an application on page 223.

**Project directory**

Knowing the directory structure and content of the project directory helps you create your own customization of the Genero Web Client for JavaScript (GWC-JS) Web client.

Before you can customize your GWC-JS Web client, you need to unzip the GWC-JS project zip file to a directory. A project directory (project_dir) is created. It serves as the GWC-JS project root directory. Amongst its directories and files you will find the following:
Table 20: Project Directory Structure

<table>
<thead>
<tr>
<th>Directory Level 1</th>
<th>Directory Level 2</th>
<th>Directory Level 3</th>
<th>Directory Level 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>project_dir</td>
<td>customization</td>
<td>default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>template</td>
<td></td>
<td></td>
</tr>
<tr>
<td>customization</td>
<td></td>
<td></td>
<td>custom_project_dir</td>
</tr>
<tr>
<td>src</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dist</td>
<td>customization</td>
<td>default</td>
<td>custom_project_dir</td>
</tr>
</tbody>
</table>

Child Directories

This topic discusses (only) three of the subdirectories of project_dir:

- **customization**: The customization directory is the root directory to your customization projects. You can have as many customization projects as you wish within this directory. Each customization project has its own customization project directory (custom_project_dir).

  At installation, the customization directory contains two subdirectories.

  - The default directory is provided as the default customization.
  - The template directory provides a structure that you can copy to start a new customization project.

  For details about the custom_project_dir, see Customization Project on page 207.

- **src**: The src directory contains all the core source files that make up the default GWC-JS front-end, to include JavaScript, HTML, CSS, and locale files. Examine the source files to familiarize yourself with how the front-end is implemented.

  **Caution**: We recommend that you leave the core sources untouched.

- **dist**: The dist/customization directory contains generated deliverables for distribution.

  - A default directory is provided as the default customization.
  - A directory is created for each compiled customization project. The name of the directory is taken from the name of the customization project, resulting in dist/customization/custom_project_dir. See Customization deliverable on page 208.

  **Note**: Make your custom GWC-JS alias/link in (FGLASDIR/web) point to
Customization Project
You modify features and styles of a Genero Web Client for JavaScript (GWC-JS) front-end within a customization project.

**Note:** This document will use the variable `custom_project_dir` when referring to the customization project directory.

In the `custom_project_dir`, you add your custom JavaScript, HTML, and CSS sources. These sources are compiled into a new front-end for distribution.

Within `custom_project_dir`, these are the directory structures and files that you must understand and manage.

**Table 21: Default customization directory**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Usage examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources</td>
<td>Project resources, to include images in an <code>img</code> subdirectory.</td>
<td>See Add an image to the application title on page 234</td>
</tr>
<tr>
<td>js</td>
<td>The project's JavaScript and widget template HTML files.</td>
<td>• See Widget JavaScript file on page 210</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• See Add header and footer text on page 231</td>
</tr>
<tr>
<td>locales</td>
<td>The project's translation texts for localization.</td>
<td>See Add localized texts on page 243</td>
</tr>
<tr>
<td>sass</td>
<td>Cascading Style Sheet files for instructing the browser how to display specific widgets. You can change colors, font, layout, and how text is positioned, etc.</td>
<td>See Widget scss file on page 213</td>
</tr>
<tr>
<td>theme.scss.json</td>
<td>Change the default GWC-JS color and theme settings for your windows, widgets, messages, buttons, tables, etc.</td>
<td>• Customizing the theme on page 227</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Display or hide the side bar on page 230</td>
</tr>
</tbody>
</table>

**Creating a customization project**
Create your customization projects in the `project_dir/customization` directory. To create a new customization project, create a copy of the customization template directory (`project_dir/customization/template`) and rename your copy. When you copy the template, you copy the directory structures needed for a customization project. This copy becomes your new `custom_project_dir`, and you can start adding your customized files.

You can have many customization projects. Each must have a unique `custom_project_dir` name.

A customization project must be compiled to be usable. See Compile a customization project on page 220. To have an application use a customization project, see Apply customization to an application on page 223.
Finding the right UI component

Customization initially consists of working with the GWC-JS client to identify which component in the interface to customize. Each element of the UI is a widget/component.

Each widget has a defined class name in the format "gbc_***Widget", where "***" is the widget name. The widget name is often the name used in the AUI tree.

**Tip:** Use your browser's developer tools to select the UI element you want to change.

Finding the widget files

Once you have identified the widget, locate the corresponding widget files in the `project_dir/src` subdirectories:

- The `js` subdirectories contain the `***Widget.js` and `***Widget.tpl.html` files.
- The `sass` subdirectories contain the `***Widget.scss` files.

For more information on widgets, see the Understanding GWC-JS widgets on page 209 section.

**Caution:** We recommend that you leave the core sources untouched.

Extending widgets

For the widgets you want to customize, create `My***Widget.tpl.html` and `My***Widget.js` files in your customization project's `js` directory. These contain the minimal HTML and JavaScript code needed to inherit and extend the parent widgets.

Create your `My***.scss` file in your customization project `sass` directory. Import your `.scss` in the `customization.scss` file.

Follow the recommended naming convention to define new widgets with the same name in the format "My***Widget", where "***" is the parent widget name. For example; `MyHeaderBarWidget.js`, `MyHeaderBarWidget.tpl.html`, and `MyHeaderBarWidget.scss`.

The How Do I ... ? on page 226 section provides examples that illustrate how to extend widget classes.

Compiling and testing as you go

As you add or update your project files, run the `grunt` command to compile your project.

Open an application to test the customization. See Apply customization to an application on page 223.

Customization deliverable

When you build a customization project, you create a Genero Web Client for JavaScript (GWC-JS) deliverable. Knowing the contents of that deliverable can help you understand how to access the resources.

The compiled GWC-JS compilation project is generated in `project_dir/dist/customization/custom_project_dir`.

The home page is `index.html`.

**Tip:** To reference a resource, you can use a path relative to `index.html`.

The stylesheet (.scss) files are combined into a single file `project_dir/dist/customization/custom_project_dir/css/main.css`.

The template (.tpl.html) files are combined in a single file `project_dir/dist/customization/custom_project_dir/js/compiledTemplates.js`.

The JavaScript (.js) files are combined in a single file in `project_dir/dist/customization/custom_project_dir/js/gbc.js`. 

The localized strings are combined in a single file in `project_dir/dist/customization/custom_project_dir/js/compiledLocales.js`.

Understanding GWC-JS widgets

Understanding how Genero Web Client for JavaScript (GWC-JS) widgets work can aid you in your customization efforts.

GWC-JS relies on a number of basic built-in widgets that define the user interface, such as application, application host, layout grids, widgets for form objects, and so on. Each widget is defined with the following:

- A JavaScript (**js**) file managing the widget creation, behavior, and listeners
- A template HTML (**.tpl.html**) file containing the minimal HTML code structure of the widget
- A sass (**.scss**) file defining the widget style

Examine the core source widget files that make up the default Web client in the `project_dir/src` directory to familiarize yourself with how the front-end is implemented.

This section explains how you can create your own customized widgets by extending the built-in widgets in your project's GWC-JS front-end web client.

- [Widget template file](#) on page 209
- [Widget JavaScript file](#) on page 210
- [Widget scss file](#) on page 213

Widget template file

A widget template (**.tpl.html**) file contains HTML code that implements an element of the user interface. Working with these components individually or grouped in containers simplifies your work in building a customized UI.

The template contains blocks of HTML code that can be easily selected by CSS. Templates define HTML elements for two main types of widgets:

- Simple widget
- Container widget

Simple widget

A *simple widget* defines elements that implement basic functions, such as form field elements. The `EditWidget`, `LabelWidget`, and `TextEditWidget` are good examples where the `<div>` container holds code for a single widget.

For example, examine the `project_dir/src/js/base/widget/widgets/formfields/EditWidget.tpl.html` template shown here:

```html
<div class="mt-field gbc_dataContentPlaceholder">
  <input type="text" />
</div>
```

The CSS selector targets elements of both the `mt-field` and `gbc_dataContentPlaceholder` CSS classes. When a widget is used in the DOM, elements and attributes of these classes will be appended in the DOM node. The `input` element with type "text" represents markup that is placed inside the DOM node.

Container widget

A *container widget* provides a placeholder for child widgets. It is defined with a class attribute with its value set to `containerElement`.
The `project_dir/src/js/base/widget/widgets/actions/MenuWidget.tpl.html` template is a good example of this type of template:

```html
<div>
  <div class="gbc_MenuWidgetTitle">
    <div class="gbc_MenuWidgetText"></div>
  </div>
  <div class="gbc_MenuWidgetScrollContainer">
    <div class="containerElement"></div>
  </div>
</div>
```

### Widget JavaScript file

The `js` file contains specific JavaScript code that implements a widget as an element of the user interface. Examples are shown of how to extend existing widgets and implement the changes by registering them as new widget classes.

The `js` files contain functions for rendering the widget template file (.tpl.html) and dynamically building the DOM element. If you wish to customize a widget, you need to extend the standard widget class in a new `js` file contained in your customization project directory.

The `js` is associated with a template file, which needs to use the same name.

**Note:** If the template file does not have the same name as the `js`, it can be reference explicitly in the code using the `__templateName` value. For an example of this usage, see Add image with style class on page 236.

### Extending a basic widget

This sample code shows the minimal code you need to extend a basic widget in `js`.

```javascript
"use strict";
modulum('newWidget', ['baseWidget', 'WidgetItem'], function(context, cls) {
  cls.newWidget = context.oo.Class(cls.baseWidget, function($super) {
    return {
      __name: "newWidget",
      /* your custom code */
    }; 
  });
}

cls.WidgetFactory.register('widgetType', cls.newWidget);
```

where:
- `newWidget` is your custom widget
- `baseWidget` is the built-in widget that it extends
- `widgetType` is the widget representation (node name) in the AUI tree.

### Extending a basic widget and registering a new widget style

Register a new widget style to allow for selective application of your customization as specified by the style settings in your form source.

```javascript
"use strict";
modulum('newWidget', ['baseWidget', 'WidgetItem'], function(context, cls) {
```
Developing Web applications  

```javascript
cls.newWidget = context.oo.Class(cls.baseWidget, function($super) {
  return {
    __name: "newWidget",
    /* your custom code */
  };
});
cls.WidgetFactory.register('widgetType', 'widgetStyle', cls.newWidget);
```

where:

- newWidget is your custom widget
- baseWidget is the built-in widget that it extends
- widgetType is the widget representation (node name) in the AUI tree.
- widgetStyle is the style name.

The widgetStyle argument (optional) allows you to reference your widget in your Genero form source. For example, in this form (.per) file snippet, the style "widgetStyle" is applied to the defined Edit form field:

```
EDIT f01 = formonly.edit, STYLE="widgetStyle";
```

Only fields with the style applied will be customized.

**Example: Extending a simple widget**

For simple widget customization, see the Edit widget example in [project_dir/customization/default/js/MyEditWidget.js](#):

```
"use strict";
modulum('MyEditWidget', ['EditWidget', 'WidgetFactory'],
  function(context, cls) {
    /**
    * Edit widget.
    * @class classes.MyEditWidget
    * @extends classes.EditWidget
    */
    cls.MyEditWidget = context.oo.Class(cls.EditWidget, function($super) {
      /** @lends classes.MyEditWidget.prototype */
      return {
        __name: "MyEditWidget".
      setTitle: function(title) {
        $(this.getElementById()).find(".title").text(title);
      },
      getTitle: function() {
        return $(this.getElementById()).find(".title").text();
      }
    });
  });
cls.WidgetFactory.register('Edit', cls.MyEditWidget);
```

```
The customization extends the built-in EditWidget. You can reference an element in the HTML template for the widget (project_dir/customization/default/js/MyEditWidget.tpl.html) by using the this.getElement() function.

Example: Extending a container widget

For container widget customization, see the example in project_dir/customization/default/js/MyApplicationHostMenuWidget.js

"use strict";

modulum('MyApplicationHostMenuWidget', ['WidgetGroupBase', 'WidgetFactory'],

    function(context, cls) {

        cls.MyApplicationHostMenuWidget = context.oo.Class(cls.WidgetGroupBase, function($super) {
            /** @lends classes.MyApplicationHostMenuWidget.prototype
             */
            return {
                __name: "MyApplicationHostMenuWidget",
                _windowIconImage: null,
                _titleElement: null,
                _defaultTitle: "Customized Title Bar",
                _aboutMenu: null,
                _debugMenu: null,

                constructor: function() {
                    $super.constructor.call(this);
                    this._aboutMenu = cls.WidgetFactory.create('ApplicationHostAboutMenu');
                    this._debugMenu = cls.WidgetFactory.create('ApplicationHostDebugMenu');

                    this.addChildWidget(this._aboutMenu);
                    this.addChildWidget(this._debugMenu);
                },

                [...] destroy: function() {
                    this.removeChildWidget(this._aboutMenu);
                    this.removeChildWidget(this._debugMenu);
                    this._aboutMenu.destroy();
                    this._aboutMenu = null;
                    this._debugMenu.destroy();
                    this._debugMenu = null;
                    $super.destroy();
                },

                [...]}
            }
        },

        cls.WidgetFactory.register('ApplicationHostMenu', cls.MyApplicationHostMenuWidget);
    });

For a Container widget, the base widget is always WidgetGroupBase. You add the child elements with this.addChildWidget and remove child elements with this.removeChildWidget.
Widget scss file
The Genero Web Client for JavaScript (GWC-JS) widget scss file contains the widget styles.

The scss files contain the colors, fonts and variables that define styles.

When the sass file is processed, it takes the variables you define for colors, for example $gbc-primary-medium-color, and outputs normal CSS with our variable values placed in the CSS.

```css
.extending_a_basic_widget_style
```

Extending a basic widget style
This sample code comes from project_dir/customization/default/sass/MyEditWidget.scss:

```
.gbc_MyEditWidget {
  > .title {
    padding: 0 5px;
    font-size: 8pt;
    color: $gbc-primary-medium-color;
    text-transform: none;
  }
  > input {
    flex: 1 1 0;
    line-height: 32px;
    border-bottom: solid 2px $gbc-primary-light-color;
  }
  &.gbc_Focus > input {
    border-bottom: solid 2px $gbc-primary-medium-color;
  }
}
```

A widget with a style in the form has a dedicated selector.

For example, in the form source (.per) for your Genero BDL application you have:

```
EDIT f01 = formonly.edt, STYLE="mystyle1 mystyle2 ...";
```

You get a generated selector in the HTML structure:

```
class="... gbc_style_mystyle1 gbc_style_mystyle2 ...
```

Create widgets with the ModelHelper class APIs
The ModelHelper class in the Genero Web Client for JavaScript (GWC-JS) project provides APIs to help you customize widgets.

GWC-JS has two main widgets types:
- Simple widgets (see Simple widget on page 209)
- Container widgets (see Container widget on page 209)

The ModelHelper class provides APIs to help customization. It is available at project_dir/src/js/base/helpers/ModelHelper.js.

Table 22: ModelHelper APIs on page 213 lists the ModelHelper APIs and describes their function.

**Table 22: ModelHelper APIs**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addNewApplicationListener(fct)</td>
<td>fct is the function called when a new application is started.</td>
</tr>
<tr>
<td>returns unfct</td>
<td>unfct is the function to call to unregister the listener</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>addCloseApplicationListener(fct)</code></td>
<td><code>fct</code> is the function called when an application is closed. <code>unfct</code> is the function to call to unregister the listener.</td>
</tr>
<tr>
<td><code>addCurrentWindowChangeListener(fct)</code></td>
<td><code>fct</code> is the function called when the current window changes. <code>unfct</code> is the function to call to unregister the listener.</td>
</tr>
<tr>
<td><code>addAuiUpdateListener(fct)</code></td>
<td><code>fct</code> is the function called when any DVM response is received. Be aware that this mechanism is global to all applications. On heavy updates this can slow down the application. <code>unfct</code> is the function to call to unregister the listener.</td>
</tr>
<tr>
<td><code>getCurrentApplication</code></td>
<td><code>win</code> is the current application object or null if the application cannot be found.</td>
</tr>
<tr>
<td><code>getApplication</code></td>
<td><code>app</code> is the application to which the widget belongs.</td>
</tr>
<tr>
<td><code>getNode(idref)</code></td>
<td><code>idref</code> is the AUI tree id reference. <code>node</code> is the related widget node or null if the node is not found.</td>
</tr>
<tr>
<td><code>getUserInterfaceNode</code></td>
<td><code>node</code> is the widget representing the User Interface node in the AUI tree or null if not found.</td>
</tr>
<tr>
<td><code>getAnchorNode</code></td>
<td><code>node</code> is the node holding the representation of value in the AUI tree.</td>
</tr>
<tr>
<td><code>getFieldNode</code></td>
<td>Applies to FormField, Matrix or TableColumn. <code>node</code> is the field node corresponding to the widget or null if it does not apply.</td>
</tr>
<tr>
<td><code>getDecorationNode</code></td>
<td>Applies to FormField, Matrix or TableColumn. <code>node</code> is the node holding the visual information (Edit, CheckBox, ComboBox, etc...) or null if not found.</td>
</tr>
</tbody>
</table>

**Extending a basic widget using ModelHelper**

The sample code creates a basic widget for the header bar using some functions in the ModelHelper class. The code in the example is found in `project_dir/customization/default/js/MyHeaderBarWidget.js`

```
"use strict";
modulum('MyHeaderBarWidget', ['WidgetBase', 'WidgetFactory'],
```
/**
 * @param {gbc} context
 * @param {classes} cls
 */
function(context, cls) {

/**
 * @class classes.MyHeaderBarWidget
 * @extends classes.WidgetBase
 */
cls.MyHeaderBarWidget = context.oo.Class(cls.WidgetBase,
function($super) {
  /** @lends classes.MyHeaderBarWidget.prototype */
  return {
    __name: "MyHeaderBarWidget",
    /** @type {classes.ModelHelper} */
    _model: null,
    /** @type {number} */
    _appsCount: null,

    constructor: function() {
      $super.constructor.call(this);
      this._appsCount = 0;
      this._model = new cls.ModelHelper(this);
      this._model.addNewApplicationListener(this.onNewApplication.bind(this));
      this._model.addCloseApplicationListener(this.onCloseApplication.bind(this));
      this._model.addCurrentWindowChangeListener(this.onCurrentWindowChanged.bind(this));
    },

    onNewApplication: function(application) {
      ++this._appsCount;
      var elt = this.getElement().querySelector(".MyHeaderBarWidget-counter");
      elt.textContent = this._appsCount.toString();
    },

    onCloseApplication: function(application) {
      --this._appsCount;
      var elt = this.getElement().querySelector(".MyHeaderBarWidget-counter");
      elt.textContent = this._appsCount.toString();
    },

    onCurrentWindowChanged: function(windowNode) {
      var elt = this.getElement().querySelector(".MyHeaderBarWidget-title");
      if (windowNode) {
        elt.textContent = windowNode.attribute('text');
      } else {
        elt.textContent = "<NONE>";
      }
    };

}};
}
Managing customization
These procedures assist you in setting up your development environment for Genero Web Client for JavaScript (GWC-JS) customization, compiling your customization projects, and having your application use a specific customization.

- Install tools on page 216
- Configure your environment on page 217
- Create customization project on page 219
- Compiling customization projects on page 219
- Compile a customization project on page 220
- Test customization using the demo on page 221
- Apply customization to an application on page 223
- Deactivate a customization project on page 224
- Prepare your customization project for production on page 224
- Build a project continuously on page 225
- Troubleshooting environment configuration on page 225
- Troubleshooting customization configuration issues on page 226

Install tools
The development tools Node.js and Git are required for customization of the Genero Web Client for JavaScript (GWC-JS) user interface.

Customization is typically done by a Web developer. As such, the tools are installed on the Web developer's machine. The Web developer machine can be different from the development server where the Genero applications reside; once the Web developer completes the customization, the compiled customization project can be deployed to the development server for testing.

1. Download the following tools:
   - Node.js: https://nodejs.org/
     Note: It is recommended that you download the stable version (v4.2.2) of node.js.
   - Git: http://git-scm.com/download/

2. Install Node.js and git on your system.
   - In Windows®:
     Execute the package names to start the Setup Wizard.
   - In Linux®/UNIX™, or Mac® OS®:
     Execute the required package management tool to start the install.

   The installation completes.

3. To check the installed version, at the command line:
   a) For Node.js, type:
      
      node --version
   b) For git, type:
      
      git --version
What to do next
Set up the environment for your project directory. See Configure your environment on page 217.

Configure your environment
Configure your environment for Genero Web Client for JavaScript (GWC-JS) customization.

Before you begin
You must have the node.js and git tools installed on your system. See Install tools on page 216.

1. Navigate to your FGLASDIR/tpl directory. Locate the GWC-JS customization project zip file (in the format of fjs-gwc-js-xxx-build123456789-project.zip) and unzip it.

You can unzip the project file anywhere you wish, however there are some things to consider:

   Caution:
   - It is recommended that you not use the FGLASDIR/web/ directory, as this is the public directory where your sources may be exposed.
   - To avoid losing your project when you upgrade the GAS, consider not placing your project in the FGLASDIR path.

The customization project zip file name is identified by:

   - gwc-js-xxx is the version number, for example gwc-js-1.00.02
   - 123456789 is the build number consisting of a date-time stamp, for example 201504141750.

A directory is created in the extracted location using the project version number as the title, in the format of gwc-js-xxx. This is your project_dir. See Project directory on page 205.

2. Install all necessary tools in the following steps:

   Tip: Installation steps are also found in your project_dir/readme.md file.

   a) Open the Node.js command prompt window.
   b) Navigate to your project_dir
   c) Install the Node Package Manager (npm) and its dependencies, type:

      npm install

      Only run the following two commands with the global option (-g) if this is the first time you have installed Node.js on the machine.

      npm install -g grunt-cli
      npm install -g bower

      Note: They must be run at administrator level on Windows®, or as the user with privileges to install in Linux®/UNIX™ or MAC® OS®.

      As tools are installed, you see tasks being run as components are updated.
   d) To finalize, install the dependencies by typing:

      grunt deps
The `grunt` `deps` command outputs a graphical report at the end showing that the `npm` and `bower` dependency packages have been updated successfully.

![Image of command prompt](image)

**Figure 61: grunt deps command updates dependencies**

If you encounter any problems during the installation, see Troubleshooting environment configuration on page 225.

Tip: For installation on Linux®/UNIX™, you can use the Node Version Manager (nvm) tool nvm tool instead of `npm`. It allows you to switch between different versions of node. To install and use, for example, node.js 4.2.2, run these commands:

```
nvm install 4.2.2
nvm use 4.2.2
```

3. Run `grunt --customization=customization/default`.
   The `grunt` command builds the project with default compilation options. A `dist` directory is created in `project_dir`, and a compiled version of the GWC-JS front-end is written to the `dist/customization/default` directory.

4. Create a symbolic link to `project_dir/dist/customization/default` in your FGLASDIR/web directory.
   Navigate to FGLASDIR/web/
   - On Windows®, run the command as administrator:

```
mklink /D link_name project_dir/dist/customization/default
```

   Important: You must run the command as administrator, it is not enough to simply have administrator permissions.
Developing Web applications

Figure 62: Create Symbolic Link in Windows®

• On Linux®/UNIX™, run the command

    ln -s project_dir/dist/customization/default link_name

    Tip: Alternatively, you can configure the GWC_JS_LOOK UP PATH on page 376 element to specify the path to your custom GWC-JS front-end.

You can now use your customized GWC-JS front-end directly in your local GAS server.

What to do next

You are all ready to begin your customization. See Create customization project on page 219.

Create customization project

Create a directory where you add files for customizing your Genero Web Client for JavaScript (GWC-JS).

The project_dir/customization/template directory provides a structure that you can copy to start your own customization project.

1. Create a copy of the customization template directory.
2. Rename your copy, for example, project_dir/customization/customization_project.

What to do next

Your next task is to activate your customization. See Compile a customization project on page 220.

Compiling customization projects

The custom.json file defines a basic customization configuration. Options provided for the grunt command allow you to override the settings in config.json.

    Tip: For simple instructions on compiling a single customization project, see Compile a customization project on page 220.

Understand custom.json

The custom.json file defines the customization compilation settings that are applied when you issue a grunt command from within the project directory with no options.

```json
{
    "compile": {
        "mode": "cdev",
        "withStats": false,
        "customization": "customization/default"
    }
}
```

"mode"

• "cdev" compiles the selected customization project for development.
Developing Web applications

- **"prod"** compiles the selected customization project for production. This compilation mode combines the customization files into as few files as possible to reduce the size of the delivery to the user agent over the network, and it improves GWC-JS performance by lessening browser cache size. See Customization deliverable on page 208.

**"withStats"**
- **"true"** compiles with statistics.
- **"false"** compiles without statistics.

**"customization"**
- **"false"** generates deliverables in dist/web. The generated deliverables are the same as those provided in $FGLASDIR/web/gwc-js.
- **"true"** generates the default customization in dist/customization/default. It is the same as having "customization": "customization/default".
- **"customization/<customization_project>"** generates the specified customization project in dist/customization/<customization_project>.

Override custom.js setting with grunt options

To override the custom.js settings, use these options:

grunt --customization=customization/default will compile customization/default in dist/customization/default.

grunt --customization=customization/<customization_project> will compile customization/<customization_project> in dist/customization/<customization_project>.

grunt --customization=ALL will compile all customization projects in the customization folder, creating the necessary deliverables in dist/customization (one folder per customization project, with the names matching).

Compile a customization project

Compile your Genero Web Client for JavaScript (GWC-JS) customization project.

When you have multiple customization projects defined in your project_dir/customization directory, only one customization is active.

**Tip:** To create a new customization project, make a copy of the template directory and rename it. Switch between projects by setting customization.

1. Open the project_dir/custom.json file using a text editor.
2. Set the compilation mode.
   
   Set mode to cdev for development.

   **Note:** Compilation modes are set to "cdev" for development, and "prod" for production.

3. Set the project.
Set the value for `customization` to either `customization/default`, provided as the default customization, or your `customization/customization_project` directory.

```json
{
    "compile": {
        "mode": "cdev",
        "withStats": false,
        "customization": "customization/default"
    }
}
```

**Note:** `customization` takes a path relative to `project_dir`.

4. Save your changes.
5. Rebuild using `grunt`.

**What to do next**

To view your customized GWC-JS, open an application in your browser. If the application fails to load, see Troubleshooting customization configuration issues on page 226.

**Test customization using the demo**

Configure the demo application to use your customized Genero Web Client for JavaScript (GWC-JS) project.

**Tip:** This procedure references a specific demo app. For instructions on configuring for your custom application, see Apply customization to an application on page 223.

**Before you begin**

- You created a customization project. The project sits in a project directory `project_dir`.
- You created a symbolic link in `FGLASDIR/web` to point to the directory `project_dir/dist/web`. See Configure your environment on page 217.

1. Create an application configuration file, for example `cust_demo.xcf`.
   Use a text editor, or if you are using Studio, go to:
   **File > New > Web/AS > Application Configuration (.xcf)**
   a) For the `PATH` element, provide the `${res.path.fgldir.demo}` resource.
   b) For the `MODULE` element, provide `demo.42r` as the module to launch the demo application.
   c) For the `GWC-JS` element, provide the name of the symbolic link to your project directory.

   **Note:** In this example, `gwc-dev` is a symbolic link from `FGLASDIR/web` to your `project_dir/dist/web` directory.

```
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
    <EXECUTION>
        <PATH>${res.path.fgldir.demo}</PATH>
        <MODULE>demo.42r</MODULE>
    </EXECUTION>
    <UA_OUTPUT>
        <PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
        <GWC-JS>gwc-dev</GWC-JS>
    </UA_OUTPUT>
</APPLICATION>
```

2. Save the configuration file (e.g. `cust_demo.xcf`) in your `${res.appdata.path}/app` directory.
3. View the demo application in your browser. You can do this by starting the standalone dispatcher from the command line using `httpdispatch` and then opening the application by entering the URL.

In this example, the URL is looking for a configuration file named `cust_demo`:

```
http://localhost:6394/ua/r/cust_demo
```

If the application fails to load, see Troubleshooting customization configuration issues on page 226.

Customize the user interface's theme style

In the following steps you apply some customization to the user interface by modifying styles in the `theme.scss.json` file.

4. Open your `project_dir/customization/customization_project/theme.scss.json` file in a text editor.

5. Change the primary color palette from blue to, for example, your corporate colors or a color of your choosing.

   Edit the values for the `gbc-primary` variables.

   ```
   "gbc-primary-color" : "$mt-blue-700",
   "gbc-primary-medium-color" : "$mt-blue-500",
   "gbc-primary-light-color" : "$mt-blue-100",
   ```

   In this example, the default primary-colors are changed to amber and white.

   ```
   "gbc-primary-color" : "$mt-amber-900",
   "gbc-primary-medium-color" : "$mt-amber-500",
   "gbc-primary-light-color" : "$mt-white",
   ```

   For more information see theme.scss.json file on page 227.

6. Save your changes.

7. Rebuild using grunt.

8. Test your customization by viewing the demo application in your browser:

```
http://localhost:6394/ua/r/cust_demo
```

   **Tip:** You may need to use CTRL + F5 to clear the browser cache before you see your changes.

You see the effect of the changes to the three `gbc-primary` color variables when applied to elements of the UI:

- application header and footer: `gbc-primary-light-color`
- toolbar/title panel: `gbc-primary-medium-color`
- side bar panel: `gbc-primary-color`
What to do next

See the How Do I … ? on page 226 topics on customization that provide procedures such as adding your company’s logo and/or changing the favicon, etc.

Apply customization to an application

You created a customization project that achieves a desired look-and-feel for the user interface. Set the GWC–JS configuration element to have the application use this customization.

Before you begin

- You created a customization project. The project sits in a project directory project_dir.
- You created a symbolic link in FGLASDIR/web to point to the directory project_dir/dist/web. See Configure your environment on page 217.

With a customization project in place, follow these steps to provide the necessary configuration information to the Genero Application Server.

1. Using a text editor, create an application configuration file for your application.
   If you are using Genero Studio, select File > New > Web/AS > Application Configuration (.xcf).
   a) For the PATH element, provide an absolute path to the location of your compiled application files.
   b) For the MODULE element, specify the module required to launch your application.
   c) For the GWC–JS element, provide the name of the symbolic link to your project directory.
   In this example, gwc-dev is a symbolic link from $FGLASDIR/web to your project_dir/dist/web directory.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH>/home/generoapps/prog1</PATH>
    <MODULE>myapp.42m</MODULE>
  </EXECUTION>
  <UA_OUTPUT>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <GWC-JS>gwc-dev</GWC-JS>
  </UA_OUTPUT>
</APPLICATION>
```
2. Save the configuration file in the `$(res.appdata.path)/app` directory.

   **Tip:** Use the name of the module as the name of the application configuration file. For example, if the module is `myapp.42m`, save the application configuration file as `myapp.xcf`.

**What to do next**

View the application in your browser by starting the standalone dispatcher from the command line using `httpdispatch` and opening the application by entering the URL. In this example, the URL is looking for a configuration file named `myapp.xcf`:

```
http://localhost:6394/ua/r/myapp.
```

If the application fails to load, see Troubleshooting customization configuration issues on page 226.

**Deactivate a customization project**

Deactivate a customization project in order to use the default Genero Web Client for JavaScript (GWC-JS) interface.

To use the default GWC-JS user interface instead of a customized project, edit the `custom.json` file in your project directory to specify.

1. Navigate to your `project_dir` directory and open the `custom.json` file in a text editor.
2. Set the value for `customization` to `false`.

```javascript
{
    "compile": {
        "mode": "cdev",
        "customization": false
    }
}
```

3. Save your changes.
4. Rebuild using `grunt`.

**What to do next**

To verify the interface displays as expected, launch an application in a browser.

**Prepare your customization project for production**

Prepare a compiled version of your Genero Web Client for JavaScript (GWC-JS) customization project for a production environment.

When you are ready to distribute your customized GWC-JS client to a production environment, set the value for `compile mode` to `prod`.

1. Open the `project_dir/custom.json` file using a text editor.
2. Set the value for `mode` to `prod`.

```javascript
{
    "compile": {
        "mode": "prod",
        "customization": "customization/default"
    }
}
```

**Note:** Compilation modes are set to "cdev" for development, and "prod" for production.

"customization" can point to any customization directory.

3. Save your changes.
4. Rebuild using grunt.

What to do next

Make a copy of `project_dir/dist/web` directory for your production server.

Build a project continuously

The grunt tool provides options for compiling your project. The `grunt` with the dev option automatically builds as you customize.

Before you begin, you must prepare your environment for customization and have a customization project identified.

1. Open a command prompt window at your `project_dir` directory.
2. At the command line, run the command:
   ```
   grunt dev
   ```
   You see tasks being run as components are updated. Then the date and time of completion is shown and a message that `grunt` is waiting. Leave the command window open as you work on your customization's project files. As you make changes or modify files, `grunt` rebuilds your project automatically.

   ![grunt dev Command Automatically Builds as you Customize](image)

3. To finish, press CTRL+C.

Troubleshooting environment configuration

Tips for troubleshooting issues you may encounter when you configure your development environment for Genero Web Client for JavaScript (GWC-JS) customization.

Unable to connect to github

Git is required to fetch bower components. When doing a `grunt deps`, if you get this error:

```
fatal: unable to connect to github.com: github.com
Use --force to continue.
```

You may have a proxy that is preventing the retrieving of bower components.
1. At the command line, type:
2. Clean your installation by choosing from the following options:
   • Type grunt clean.
   • Type npm cache clean.
   • Remove the project_dir/node_modules directory.
3. Redo the node components installation as described in the step for installing the necessary tools.

Troubleshooting customization configuration issues
Tips for troubleshooting issues encountered when launching an application in a customized Genero Web Client for JavaScript (GWC-JS) environment.

Troubleshooting topics include:
• Unable to expand bootstrap template on page 226
• Blank page displays on page 226

Unable to expand bootstrap template

A message states "Unable to expand bootstrap template". Your bootstrap.html cannot be found. You need to rebuild your project directory.
1. Open the Node.js command prompt window.
2. Navigate to project_dir.
3. Type: grunt.

For more information and support contact your local Four Js support center.

Blank page displays

Your application displays a blank page. If the application debug information indicates that Handlebars is missing, you likely need to run the grunt command.
1. Check in your browser’s debug window to see if you get a JavaScript "HandleBars is missing" message. If you find this message, continue to the next step.
2. Open the Node.js command prompt window.
3. Navigate to project_dir.
4. Type: grunt.

For more information and support contact your local Four Js support center.

How Do I … ?
Procedures for the customization of your Genero Web Client for JavaScript (GWC-JS) front-end.

Topics in this section will help you understand what widgets you need to extend to change the look and feel of your application.

Customize the User Interface
Regardless of your customization needs, there is a general procedure to follow when customizing your user interface for a Genero Web Client for JavaScript (GWC-JS) front-end.

Before you begin, you must prepare your environment for customization and have a customization project identified.

This procedure provides the generic outline for any customization project. For specific examples of customization and to see sample code, see the procedures found in this section.

1. Identify the widget you want to modify.
   Most of the UI elements are widgets that use the same name as in the AUI tree, such as the EditWidget or the TableWidget. If you are not sure which widget to modify, you can use browser developer tools
to inspect the elements of a GWC-JS application interface and identify the widget. Browser developer tools are accessed by pressing the F12 key.

2. Modify the stylesheets (.scss).
   a) Add entries directly in customization.scss, or create additional stylesheet (.scss) files. Place new stylesheet files in the project's sass directory.
   b) Reference any added stylesheet (.scss) files in the customization.scss file.
      For example, if the stylesheet is MyHeader.scss, you would add the following to customization.scss:
      ```
      @import "MyHeader";
      ```

3. If additional customization is needed, extend the widget.
   a) Create the JavaScript (.js) and template (.tpl.html) files needed to extend the widget.
      By default, the JavaScript and template files share the same name. Save these files in the project's js directory.
      Specify a template file with a different name with __templateName:
      ```
      __name: xxx,
      __templateName: "yyy"
      ```

4. Rebuild your project with grunt.
5. Test your customization.

Customizing the theme
The simplest way to customize the look and feel of your Genero Web Client for JavaScript (GWC-JS) front-end is to change theme styles in the theme.scss.json file.

This section includes topics that describes the theme.scss.json file found in your project_dir/customization/customization_project directory, and provides you with details of how you can modify the look and feel of your application.

**Tip:** In order to not lose your work when you upgrade the product, do not modify the default theme installed by product. Modify the theme for your customization project.

- **theme.scss.json file** on page 227
- **Customize theme settings** on page 229

theme.scss.json file
The theme.scss.json file contains a list of theme settings. Changing a theme setting is an easy way to modify the appearance of the user interface.

**Important:** Changes made to theme.scss.json require a rebuild of the customization project before changes take effect. Run grunt to rebuild your project.

Syntax
Understanding this customization requires some knowledge of JSON file syntax. The syntax of each theme setting consists of a variable_name, followed by a colon (:) and then the variable_value. Variable names and values are in double quotes. Theme styles are separated by commas.

```
"variable_name" : "variable_value",
```

- The variable_name is a unique identifier used throughout the Genero Web Client for JavaScript (GWC-JS) project to set theme styles, for example the UI window's primary color is referenced by the variable gbc-primary-background-color. These variables are standard theme variables and should not be subject to change.
- The variable_value can be a color, a number, or a variable, depending on the style. If the value is defined by a variable, for example $vname, “vname” is an existing variable name in the GWC-JS project.

**Color variables naming convention**

Variables that define colors follow this naming convention:

\$mt-colorname[-intensity]

- **colorname** is the name of the color as defined in the material color palette.
- **intensity** (optional) is the number for the gradient, for example: "$mt-grey-200", defines an intensity of light grey. For more information on material color, go to [http://www.google.com/design](http://www.google.com/design) and navigate to the “Color palette” page.

**Example of customized theme styles**

The *theme.scss.json* file contains the list of styles as shown in the example. See the table **Table 23: theme.scss.json** on page 228 for a description of the variables.

```json
{
  "gbc-font-size-ratio" : "0.9",
  "gbc-margin-ratio" : "0.6",
  "gbc-field-height-ratio" : "0.7",
  "gbc-radio-button-size" : "14px",
  "gbc-checkbox-size" : "14px",
  "gbc-primary-background-color" : "$mt-grey-50",
  "gbc-secondary-background-color" : "$mt-white",
  "gbc-field-background-color" : "$mt-white",
  "gbc-field-disabled-background" : "rgba(0,0,0,0.04)",
  "gbc-primary-color" : "$mt-blue-700",
  "gbc-primary-medium-color" : "$mt-blue-500",
  "gbc-primary-light-color" : "$mt-blue-100",
  "gbc-secondary-color" : "$mt-grey-600",
  "gbc-disabled-color" : "$mt-grey-400",
  "gbc-separator-color" : "$mt-grey-400",
  "gbc-header-color" : "$mt-grey-100",
  "gbc-message-color" : "$mt-grey-800",
  "gbc-error-color" : "$mt-red-800",
  "gbc-sidebar-always-visible-min-width" : "1400px",
  "gbc-sidebar-default-width" : "300px",
  "gbc-animation-duration" : "0.7s"
}
```

**Table 23: theme.scss.json**

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>gbc-font-size-ratio</strong></td>
<td>The font size adjustment ratio in relation to Droid Sans font family sizes. For example, if the browser's available font for the <code>&lt;body&gt;</code> element is Times (12px), size of letters are adjusted by multiplying <strong>gbc-font-size-ratio</strong> by font size. In our example, they display at 10.8 pixels high, ((0.9 \times 12)).</td>
</tr>
<tr>
<td><strong>gbc-margin-ratio</strong></td>
<td>Ratio applied to the space between fields (padding) calculated in relation to the width of the fields’ container.</td>
</tr>
<tr>
<td><strong>gbc-field-height-ratio</strong></td>
<td>Ratio applied to the field height calculated in relation to the height of the fields' container.</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gbc-radiobutton-size</td>
<td>Size of the radio button in pixels.</td>
</tr>
<tr>
<td>gbc-checkbox-size</td>
<td>Size of the checkbox in pixels.</td>
</tr>
<tr>
<td>gbc-primary-background-color</td>
<td>Window primary background color.</td>
</tr>
<tr>
<td>gbc-secondary-background-color</td>
<td>Window/containers (group, drop-down menu) secondary color.</td>
</tr>
<tr>
<td>gbc-field-background-color</td>
<td>Widget background (bg) color.</td>
</tr>
<tr>
<td>gbc-field-disabled-background</td>
<td>Widget disabled background color.</td>
</tr>
<tr>
<td>gbc-primary-color</td>
<td>Button/widget color gradient for dark intensity.</td>
</tr>
<tr>
<td>gbc-primary-medium-color</td>
<td>Button/widget color gradient for normal intensity.</td>
</tr>
<tr>
<td>gbc-primary-light-color</td>
<td>Button/widget color gradient for light intensity.</td>
</tr>
<tr>
<td>gbc-secondary-color</td>
<td>Widget (RADIOGROUP, CHECKBOX) border color.</td>
</tr>
<tr>
<td>gbc-disabled-color</td>
<td>Widget disabled color.</td>
</tr>
<tr>
<td>gbc-separator-color</td>
<td>Toolbar separator color.</td>
</tr>
<tr>
<td>gbc-header-color</td>
<td>Table header background color.</td>
</tr>
<tr>
<td>gbc-message-color</td>
<td>Message background color.</td>
</tr>
<tr>
<td>gbc-error-color</td>
<td>Error message background color.</td>
</tr>
<tr>
<td>gbc-sidebar-always-visible-min-width</td>
<td>Browser window width size (in pixels) for the side bar to appear. Use this setting to hide the side bar. See Display or hide the side bar on page 230.</td>
</tr>
<tr>
<td>gbc-sidebar-default-width</td>
<td>Default size of side bar in pixels.</td>
</tr>
<tr>
<td>gbc-animation-duration</td>
<td>Animation duration (in seconds), such as the speed of Product identification window pop-down appearance.</td>
</tr>
</tbody>
</table>

**Tip:** You can reference a color variable setting from a .scss file by prefacing the color variable name with $. For example:

```
color: $gbc-primary-medium-color;
```

See Widget scss file on page 213.

- Display or hide the side bar on page 230
- Customize the pop-up animation on page 230

Customize theme settings
Complete this procedure to modify the theme settings in the theme.scss.json file.

1. Open your project_dir/customization/customization_project/theme.scss.json file in a text editor.
2. Change theme settings by modifying or adding style values or variables.
   For information on theme settings, see theme.scss.json file on page 227.
3. Save your changes.
4. Rebuild using grunt.
5. Close and reopen the application to view the current theme.

**Tip:** You may need to use CTRL + F5 to clear the browser cache before you see your changes.
Customize the pop-up animation
Complete this procedure to change the time it takes for a pop up window animation to open.

The `gbc-animation-duration` style specifies the delay (in seconds) that pop up windows take to open and close. The lower the number, the faster the pop up window is likely to appear.

The style is defined in the `theme.scss.json` file.

1. Open `project_dir/customization/customization_project/theme.scss.json` file in a text editor.
2. To set the animation to a slower speed, set the `gbc-animation-duration` style to a high number of seconds.

```json
{
    ...
    "gbc-sidebar-always-visible-min-width" : "1400px",
    "gbc-sidebar-default-width"            : "350px",
    "gbc-animation-duration"               : "4.200s"
}
```

In this example, the animation is set to 4.2 seconds (a high number). When a pop up window is called, it appears to move slowly as it drops down from the menu bar.

3. To set the animation to a faster speed, set the `gbc-animation-duration` style to a fraction of a second.

```json
{
    ...
    "gbc-sidebar-always-visible-min-width" : "9999px",
    "gbc-sidebar-default-width"            : "350px",
    "gbc-animation-duration"               : "0.200s"
}
```

In this example, the animation is set to 0.2 seconds (a low number). Pop up windows appear much faster.

4. Rebuild using grunt.
5. Test the animation works as expected. Open the demos application and click on the Application information menu icon to open the Product identification pop up window.

   Tip: You may need to use CTRL + F5 to clear the browser cache before you see your changes.

Display or hide the side bar
Whether the side bar displays depends on the setting of the `gbc-sidebar-always-visible-min-width` style.

About this task
The `gbc-sidebar-always-visible-min-width` style specifies the width (in pixels) that the browser window must reach before the side bar is always visible. The lower the number, the more likely the side bar will be visible. When the browser width is less than the width specified by the style, the side bar is replaced by the side bar icon, which must be used to access the contents of the side bar.

The style is defined in the `theme.scss.json` file.

1. Open `project_dir/customization/custom_project_dir/theme.scss.json` file in a text editor.
2. To display the side bar, set the `gbc-sidebar-always-visible-min-width` style to a low number of pixels.

```json
{
    ...
```
In this example, the minimal width is 1400 pixels (a low number). When the browser width is greater than 1400 pixels, the side bar displays.

3. To hide the side bar, set the `gbc-sidebar-always-visible-min-width` style to a high number of pixels.

```
{...
  "gbc-sidebar-always-visible-min-width" : "9999px",
  "gbc-sidebar-default-width"            : "350px",
  "gbc-animation-duration"               : "0.200s"
}
```

In this example, the minimum width is 9999 pixels (a high number). The side bar is hidden when the browser window is less than this size.

4. Rebuild using grunt.

5. Test the side bar works as expected. Open your application and expand / contract the browser width.

**Tip:** You may need to use CTRL + F5 to clear the browser cache before you see your changes.

### Add header and footer text

Procedures for adding header and footer text to the application title of your Genero Web Client for JavaScript (GWC-JS) front-end.

Topics in this section will help you change the look and feel of your application by adding header and footer text to the application.

- Add header text on page 231
- Add footer text on page 232

**Add header text**

Add header text to the MainContainerWidget of your Genero Web Client for JavaScript (GWC-JS) project user interface.

**Before you begin:**

You have a project directory and your `project_dir/customization/custom_project_dir` directory contains the following `.js`, `.tpl.html`, and `.scss` files in the appropriate directories:

- `MyMainContainerWidget.tpl.html`
- `MyMainContainerWidget.tpl.js`
- `MyMainContainerWidget.scss`
- `MyHeaderBarWidget.tpl.html`
- `MyHeaderBarWidget.js`

**Note:** These files may be copied from your `project_dir/customization/default/` directory or you can use the default customization.

1. Open the `project_dir/customization/custom_project_dir/js/MyHeaderBarWidget.tpl.html` file with a text editor.

2. Replace the existing `div` element with your changes:

   ```html
   <div>
   This is the header text
   </div>
   ```
3. Open your `project_dir/customization/custom_project_dir/MyMainContainerWidget.js` file.

The header bar widget is added. Check that the main container class has code that adds the header bar class as a child of the main container widget.

```javascript
"use strict";

modulum('MyMainContainerWidget', ['WidgetGroupBase', 'WidgetFactory'],
...

class constructor: function() {
    $super.constructor.call(this);

    var headerBar = new cls.MyHeaderBarWidget();
    headerBar.setParentWidget(this);

    this.getElementById().querySelector("header").appendChild(headerBar.getElementById());
}
);

cls.WidgetFactory.register('MainContainer', cls.MyMainContainerWidget);
```

In the highlighted code in the JavaScript example, you can see:

- The variable `headerBar` references an instance of the `MyHeaderBarWidget` class.
- The `.getElement().querySelector` method inserts the header bar widget as a child of the `header` element in the main container element of the interface.

4. In your `project_dir/customization/custom_project_dir/scss/customization.scss` file add a reference to the `MyMainContainerWidget.scss`. In this example we import the `MyMainContainerWidget` style.

```scss
@import "MyMainContainerWidget";
```

5. Rebuild using grunt.

What to do next

View the changes by opening the GAS demos application in your browser. You can do this by configuring the demos application for your project, see Test customization using the demo on page 221.

The GWC-JS header text is now changed.

Add footer text

Add footer text to the MainContainerWidget of your Genero Web Client for JavaScript (GWC-JS) project user interface.

Before you begin:

You have a project directory and your `project_dir/customization/custom_project_dir` directory contains the following `.js`, `.tpl.html`, and `.scss` files in the appropriate directories:

- `MyMainContainerWidget.tpl.html`
- `MyMainContainerWidget.js`
- `MyMainContainerWidget.scss`

Note: These files may be copied from your `project_dir/customization/default/` directory or you can use the default customization.
1. Copy the sampleItem.js file found in project_dir/customization/template directory to your project_dir/customization/custom_project_dir/js/ directory, and rename it MyFooterBarWidget.js.

2. Open MyFooterBarWidget.js with a text editor.

   Replace the references to sampleItem with the new widget name where found:
   - in the class creation method (modulum)
   - in the context function variable (cls)
   - in the reference to the template file name (_name)

   Save your changes.

   ```javascript
   // declare the dependency to inheritedWidget (here WidgetBase) module
   modulum('MyFooterBarWidget', ['WidgetBase', 'WidgetFactory'],
   function(context, cls) {
     cls.MyFooterBarWidget = context.oo.Class(cls.WidgetBase,
     function($super) {
       return {
         __name: "MyFooterBarWidget"
       };
     });
   });
   ```

3. Create the project_dir/customization/custom_project_dir/js/MyFooterBarWidget.tpl.html file with a text editor.

4. Inside a div element add your footer text. Save your changes:

   ```html
   <div>
     This is the Footer text
   </div>
   ```

5. Open your project_dir/customization/custom_project_dir/MyMainContainerWidget.js file.

   The footer bar widget is added. Check that the main container class has code that adds the footer bar class as a child of the main container widget.

   ```javascript
   "use strict";

   modulum('MyMainContainerWidget', ['WidgetGroupBase', 'WidgetFactory'],
   [...]
   constructor: function() {
     $super.constructor.call(this);
     var headerBar = new cls.MyHeaderBarWidget();
     headerBar.setParentWidget(this);
     this.getElement().querySelector("header").appendChild(headerBar.getElement());

     var footerBar = new cls.MyFooterBarWidget();
     footerBar.setParentWidget(this);
     this.getElement().querySelector("footer").appendChild(footerBar.getElement());
   });
   ```
Developing Web applications

In the highlighted code in the JavaScript example, you can see:

- The variable `footerBar` references an instance of the `MyFooterBarWidget` class.
- The `.getElement().querySelector` method inserts the footer bar widget as a child of the `footer` element in the main container element of the interface.

6. In your `project_dir/customization/custom_project_dir/scss/customization.scss` file add a reference to the `MyMainContainerWidget.scss`. In this example we import the `MyMainContainerWidget` style.

```javascript
.cls.WidgetFactory.register('MainContainer',
 cls.MyMainContainerWidget);
```

7. Rebuild using grunt.

What to do next

View the changes by opening the GAS demos application in your browser. You can do this by configuring the demos application for your project, see Test customization using the demo on page 221.

The GWC-JS footer text is now changed.

Add an image to the application title

Procedures for adding images to the application title of your Genero Web Client for JavaScript (GWC-JS) front-end.

Topics in this section will help you change the look and feel of your application by adding images to the application title.

- Add image/logo on page 234
- Add image with style class on page 236

Add image/logo

Add an image or company logo to the header bar widget of your application.

Before you begin:

- The following is assumed:
  - You have a project directory and your `project_dir/customization/custom_project_dir` directory contains the following `.js`, `.tpl.html`, and `.scss` files in the appropriate directories:
    - `MyMainContainerWidget.tpl.html`
    - `MyMainContainerWidget.tpl.js`
    - `MyMainContainerWidget.scss`
    - `MyHeaderBarWidget.tpl.html`
    - `MyHeaderBarWidget.js`
  - Your `project_dir/customization/customization_project/resources/img` directory contains your image file.

1. Open the `project_dir/customization/custom_project_dir/js/MyHeaderBarWidget.tpl.html` file with a text editor.

2. Inside the `div` element add an `<img>` tag, and set its `src` attribute to the path to your image. Save your changes:

```html
<div>
  <img src="./resources/img/my_logo.jpg"/>
</div>
```
In this example you can see:

- An image is added in the `img` element.
  
  **Important:** When referencing an image in your project's `resources` directory from a file in the `js` directory, the path needs to be set relative to your `customization_project` application root at compilation time (`./`).

- Text displays the number of applications opened, referencing the `MyHeaderBarWidget-counter` class.

- Text displays the application title by referencing locale custom keys with the `MyHeaderBarWidget-title` class.

3. Open your `project_dir/customization/custom_project_dir/MyMainContainerWidget.js` file.

The header bar widget is added. Check that the main container class has code that adds the header bar class as a child of the main container widget.

```javascript
"use strict";

modulum('MyMainContainerWidget', ['WidgetGroupBase', 'WidgetFactory'],
...

constructor: function () {
  $super.constructor.call(this);

  var headerBar = new cls.MyHeaderBarWidget();
  headerBar.setParentWidget(this);

  this.getElementById().querySelector("header").appendChild(headerBar.getElementById());

  cls.WidgetFactory.register('MainContainer',
  cls.MyMainContainerWidget);
});
```

In the highlighted code in the JavaScript example, you can see:

- The variable `headerBar` references an instance of the `MyHeaderBarWidget` class.

- The `.getElement().querySelector("header")` method inserts the header bar widget as a child of the `header` element in the main container element of the interface.

4. In your `project_dir/customization/custom_project_dir/scss/customization.scss` file add a reference to the `MyMainContainerWidget.scss`. In this example we import the `MyMainContainerWidget` style.

```scss
@import "MyMainContainerWidget";
```

5. Rebuild using `grunt`.

6. Test the image is displayed as expected by closing and reopening your application.

  **Tip:** You may need to use CTRL + F5 to clear the browser cache before you see your changes.
Add an image or logo to the application using a style class.

**Before you begin:**

- The following is assumed:
  - You have a project directory and your `project_dir/customization/custom_project_dir` directory contains the following `.js`, `.tpl.html`, and `.scss` files in the appropriate directories:
    - `MyMainContainerWidget.tpl.html`
    - `MyMainContainerWidget.tpl.js`
    - `MyMainContainerWidget.scss`
    - `MyHeaderBarWidget.tpl.html`
    - `MyHeaderBarWidget.js`
  
  **Note:** These files may be copied from your `project_dir/customization/default/` directory or you can use the default customization.
  - Your `project_dir/customization/customization_project/resources/img` directory contains your image file.

1. Open the `project_dir/customization/custom_project_dir/js/MyHeaderBarWidget.tpl.html` file with a text editor.
2. Inside the `div` element add an `<img>` tag, and set its `class` attribute to the `mylogo` style. Save your changes.

   ```html
   <div>
     <img class="mylogo"/>
     Applications started: <b class="MyHeaderBarWidget-counter"></b>
     <span data-i18n="mycusto.window.currentTitle"></span> : <b class="MyHeaderBarWidget-title"></b>
   </div>
   
   **Tip:** If the logo is to appear on the right-hand side of the title, add the `img` element after the window title in the `<div>`.

3. In your `project_dir/customization/custom_project_dir/scss` directory, add a new style sheet `scss` file, called `MyHeaderBarWidget.scss`. In this `scss` example we define the `mylogo` style class.

   ```scss
   /* Header bar */
   .mylogo {
     width: 50px;
     height: 50px;
     background-image: url("../resources/img/my_logo.png");
   }
   
   You can see:
   - The `.mylogo` style class has settings for the size of the logo.
   - The `background-image` property sets the URL for locating the image file.
   
   **Note:** As `scss` stylesheets are compiled in the `main.css` in the `css` directory, the path to a resource is set from the `main.css` of your distribution to the resource using relative path (`../`).

4. Open your `project_dir/customization/custom_project_dir/MyMainContainerWidget.js` file.
   The header bar widget is added. Check that the main container class has code that adds the header bar class as a child of the main container widget.

   "use strict";
modulun('MyMainContainerWidget', ['WidgetGroupBase', 'WidgetFactory'],
...
constructor: function() {
  $super.constructor.call(this);

  var headerBar = new cls.MyHeaderBarWidget();
  headerBar.setParentWidget(this);

  this.getElementById().querySelector("header").appendChild(headerBar.getElementById());
}
});
cls.WidgetFactory.register('MainContainer', cls.MyMainContainerWidget);
});

In the highlighted code in the JavaScript example, you can see:

- The variable `headerBar` references an instance of the `MyHeaderBarWidget` class.
- The `.getElement().querySelector` method inserts the header bar widget as a child of the `header` element in the main container element of the interface.

5. In your `project_dir/customization/custom_project_dir/scss/customization.scss` file add a reference to the `MyHeaderBarWidget.scss`.

In this example we import the `MyHeaderBarWidget` style.

```scss
@import "MyHeaderBarWidget";
@import "MyMainContainerWidget";
```

6. Rebuild using grunt.

7. Test the logo is displayed as expected by closing and reopening your application.

   Tip: You may need to use CTRL + F5 to clear the browser cache before you see your changes.

Add a favicon image

Add a favicon image to the user interface of your Genero Web Client for JavaScript (GWC-JS) project.

About this task:

Follow this procedure to replace the default favicon image with your own.

1. Save your favicon image with the name "gbc_logo.ico"

   The default favicon is called `gbc_logo.ico`. It is referenced in `project_dir/src/index.html`. This page is currently not customizable, therefore you need to name your favicon "gbc_logo.ico".

2. Add the favicon image to your `project_dir/customization/customization_project/resources/img` directory.

3. Rebuild using grunt.

4. Test the favicon is displayed as expected on the left side of the browser’s address bar and on the tab by closing and reopening your application.

   Tip: You may need to use CTRL + F5 to clear the browser cache before you see your changes.

Add file loading images

Describes different procedures you can use for customizing images that display when file loading is in progress in your Genero Web Client for JavaScript (GWC-JS) applications.

- Add a file loading image on page 238
- Add a centered file loading icon on page 238
Add a file loading image
Add a file loading image to the user interface of your Genero Web Client for JavaScript (GWC-JS) project.

About this task:
Customize the customization.scss style for the ApplicationHostMenuRuntimeWidget to change the animation icon shown when processing a file loading request.

1. In your project_dir/customization/customization_project/scss/customization.scss file add a style for the ApplicationHostMenuRuntimeWidget.
   In this example we add the style for when the .gbc_ApplicationHostMenuRuntimeWidget.processing selector has the processing value. The icon is shown only when the widget is processing the loading of a file. This behavior is specified in the widget file ApplicationHostMenuRuntimeWidget.js.

```scss
@import "MyMainContainer";
.gbc_ApplicationHostMenuRuntimeWidget.processing {
  background-image: url("../resources/img/html5_circle.gif");
  a i::before {
    content: none;
  }
}
```

You can see:
- The background-image property sets the URL for locating the new image file.
- The :before selector removes the previously-used icon.

2. Rebuild using grunt.
3. View the changes by opening the GAS demos application in your browser. You can do this by configuring the demos application for your project, see Test customization using the demo on page 221.
4. In the Topic tree of the demo directory, navigate to User Interface > UI Basics > Widgets. Double-click on the ProgressBar application in the panel to the right.
   Tip: You may need to use CTRL + F5 to clear the browser cache before you see your changes.
5. Click on the Go1 menu button to start a file loading.
   The new icon is displayed on the application title bar while the file is being loaded.

Add a centered file loading icon
Add a file loading image that displays in the center of the page when file loading is in progress in your Genero Browser Client (GBC) applications.

About this task:
This customization involves extending the ApplicationHostMenuWidget and the ApplicationHostMenuRuntimeWidget to define the animation icon behavior. For this you create the following files:

- MyApplicationHostMenuRuntimeWidget.js
- MyApplicationHostMenuWidget.js.
- MyApplicationHostMenuRuntimeWidget.scss
- MyApplicationHostMenuRuntimeWidget.tpl.html.

1. With a text editor create a file called MyApplicationHostMenuRuntimeWidget.js in your project_dir/customization/custom_project_dir/js directory.
   Copy the code shown in the example and save.

```
"use strict";
```
modulum('MyApplicationHostMenuRuntimeWidget', ['WidgetBase', 'WidgetFactory'],
/**
 * @param {gbc} context
 * @param {classes} cls
 */
function(context, cls) {

/**
 * @class classes.MyApplicationHostMenuRuntimeWidget
 * @extends classes.WidgetBase
 */
cls.MyApplicationHostMenuRuntimeWidget =
context.oo.Class(cls.WidgetBase, function($super) {
    /** @lends classes.MyApplicationHostMenuRuntimeWidget.prototype */
    return {
        __name: "MyApplicationHostMenuRuntimeWidget",
        _waitingTime: 100,
        _timer: null,

        _initElement: function() {
            $super._initElement.call(this);
            this._waitingTime = 1000*parseFloat(gbc.constants.theme["gbc-loader-threshold"]);
        },

        setIdle: function() {
            if(this._timer){
                window.clearTimeout(this._timer);
                this._timer = null;
            }
            this.removeClass("processing");
        },

        setProcessing: function() {
            if(this._timer){
                window.clearTimeout(this._timer);
                this._timer = null;
            }
            this._timer = window.setTimeout(function(){
                this.addClass("processing");
                this._waitingTime;
            }.bind(this),this._waitingTime);
        }
    };
}

cls.WidgetFactory.register('ApplicationHostMenuRuntime', cls.MyApplicationHostMenuRuntimeWidget);
});

In the code, you can see:

- Two variables are added, _timer and _waitingTime
- In the _initElement function, the _timer is set from a calculation using the gbc-loader-threshold value in the theme.scss.json.
- The setIdle function uses the window object clearTimeout() method to clear the timer. The function also removes the processing class.
- The setProcessing function sets the timer with the setTimeout() method, which calls a function to add the processing class and to bind to it after a delay specified by _waitingTime.

2. Create a file called MyApplicationHostMenuWidget.js in your project_dir/customization/custom_project_dir/js directory.
Copy the code shown in the example and save:

```javascript
"use strict";

modulun('MyApplicationHostMenuWidget', ['ApplicationHostMenuWidget', 'WidgetFactory'],
/**
 * @param {gbc} context
 * @param {classes} cls
 */
function(context, cls) {
  /**
   * @class classes.MyApplicationHostMenuWidget
   * @extends classes.ApplicationHostMenuWidget
   */
  cls.MyApplicationHostMenuWidget = context.oo.Class(cls.ApplicationHostMenuWidget, function($super) {
    return {
      __name: "MyApplicationHostMenuWidget",
      __templateName: "ApplicationHostMenuWidget",

      _initElement: function() {
        $super._initElement.call(this);
        this._element.addClass("gbc_ApplicationHostMenuWidget");
      },

      _createMenuItems: function() {
        this._uploadStatus = cls.WidgetFactory.create('ApplicationHostUploadsMenu');
        this.addChildWidget(this._uploadStatus);

        this._runtimeStatus = cls.Wid...
In the `_createMenuItems` function the `runtimeStatus` element (shown highlighted) is appended to the document body element.

3. In your `project_dir/customization/custom_project_dir/sass` directory create a file called `MyApplicationHostMenuRuntimeWidget.scss`

Copy the code shown in the example and save:

```scss
.gbc_MyApplicationHostMenuRuntimeWidget.processing {
  opacity: 0;
  position: fixed;
  z-index: 9999;
  top: 40%;
  left: 50%;

  &:hover {
    cursor: default;
  }

  &.processing {
    //animation: pulse 200ms infinite;
    opacity: 1;
  }

  $base-line-height: 100px;
  $white: rgb(158,158,158);
  $off-white: rgba($white, 0.2);
  $spin-duration: 1s;
  $pulse-duration: 750ms;

  @keyframes spin {
    0% {
      transform: rotate(0deg);
    }
    100% {
      transform: rotate(360deg);
    }
  }

  .loading {
    border-radius: 50%;
    width: $base-line-height;
    height: $base-line-height;
    border: .5rem solid $off-white;
    border-top-color: $white;
    animation: spin $spin-duration infinite linear;
    &:double {
      border-style: double;
      border-width: .5rem;
    }
  }
}
```

This style example defines the animation icon and its behavior:
• The style is added for when the .gbc_MyApplicationHostMenuRuntimeWidget.processing selector has the processing value. This means that the animation icon is shown only when the application is busy loading a file.
• The position of the icon is fixed to display over other elements (z-index=9999) in the center of the page (top: 40%; left: 50%) to give the rotation effect.
• An at-rule, @keyframes spin, specifies how the icon will gradually transform from the start to the end (0% and 100%) to give the rotation effect.
• The loading class has settings for the size, border, and color properties of the icon. Its animation property references the spin rule, and the spin duration value. The animation continues for ever and the linear value specifies an animation with the same speed from start to finish.

4. In your project_dir/customization/custom_project_dir/js directory create a file called MyApplicationHostMenuRuntimeWidget.tpl.html.

Copy the code shown in the example and save:

```html
<div>
  <div class="loading" title="1">
  </div>
</div>
```

The div class attribute references the loading style you created in the MyApplicationHostMenuRuntimeWidget.scss file.

5. Open your project_dir/customization/custom_project_dir/theme.scss.json file for editing.

Copy the code shown in the example and save.

```json
{
  "gbc-loader-threshold": "0.5s"
}
```

The value (0.5s) specifies a delay (in seconds) for the timer function loading the icon.

6. Open the project_dir/customization/custom_project_dir/sass/customization.scss file for editing.

Copy the code shown in the example and save.

```scss
@import "MyApplicationHostMenuRuntimeWidget";
```

In this example we import the MyApplicationHostMenuRuntimeWidget style.

7. Rebuild using grunt.

8. View the changes by opening the GAS demos application in your browser. You can do this by configuring the demos application for your project, see Test customization using the demo on page 221.

9. In the Topic tree of the demo directory, navigate to User Interface > UI Basics > Widgets. Double-click on the ProgressBar application in the panel to the right.

   Tip: You may need to use CTRL + F5 to clear the browser cache before you see your changes.

10. Click on the Go1 menu button to start a file loading.

    The animation icon is displayed in the center of the browser window while the file is being loaded.
Add localized texts
You can customize localization by adding your custom translation texts in the locale file and referencing them in a widget template file.

Adding a Custom Title for the Current Window
The Genero Web Client for JavaScript (GWC-JS) front-end provides a mechanism to internationalize your application interface. For more details about this mechanism see Translations for GWC-JS on page 49. For example to add a custom title for the current window, perform the following:

• Add a Localization Key
  In your `project_dir/customization/customization_projectlocales` directory, choose your localization file or create your own one named `xx-YY.json`, where “xx-YY” follows the standard localization code used for languages. For example, to the locale file for French `fr-FR.json`, add your custom keys:

```json
{
  "mycusto":{
    "window":{
      "currentTitle": "Titre de la fenêtre courante"
    }
  }
}
```

In this case the translation key is `mycusto.window.currentTitle`

• Reference the Translation Key in a Widget Template file
  Reference your custom translation key with the HTML attribute `data-i18n` in the widget template file for the header bar. For example, in `project_dir/customization/customization_project_dir/js/MyHeaderBarWidget.tpl.html` add this within a `<span>` element with default text.

```html
<span class="…" data-i18n="mycusto.window.currentTitle">Currently opened window title</span>
```
Figure 66: GWC-JS User Interface French Locale

An English browser displays "Currently opened window title", and a French browser displays "Titre de la fenêtre courante".

Redirect the application end

The MySessionEndWidget widget, provided as part of the default customization, allows you to specify a redirection URL.

**Important**: The customized files for the MySessionEndWidget widget were introduced with GWC-JS 1.00.20.

The MySessionEndWidget customization extends the SessionEndWidget to automatically redirect the user to a specified URL after a set number of milliseconds.

Updating the MySessionEndWidget customization

Edit `MySessionEndWidget.js` to change the redirect URL or the delay before redirection. Comments in the widget identify where these changes can be made.

To change the redirect URL, replace `http://www.google.com` with the desired URL:

```javascript
// update redirection link url of the template
var redirectionLink =
    this._element.getElementsByClassName("redirectionLink")[0];
redirectionLink.title = i18n.t("mycusto.session.redirectionText");
var url = "http://www.google.com";
redirectionLink.href = url;
```

To change the length of the delay before redirecting, replace `10000` with the number of milliseconds to delay:

```javascript
var modelHelper = new cls.ModelHelper(this);
// Launch the redirection after a delay of 10 seconds
// To remove the delay, remove the setTimeout, but keep the 'if' block
setTimeout(function () {
    // check if an application is running in the current session before reloading
    if(!modelHelper.getCurrentApplication()) {
```
window.location = url;
}
}.bind(this), 10000); // 10000ms

To eliminate the delay, remove the setTimeout function code surrounding `window.location = url`:

```javascript
var modelHelper = new cls.ModelHelper(this);
// check if an application is running in the current session before reloading
if(!modelHelper.getCurrentApplication()) {
    window.location = url;
}
```

**Note:** This does not completely halt the display of the end page. The ending page flashes briefly, as redirection occurs only after the ending page displays.

## Change the redirection message

The message displayed prior to the redirection is based on the value of the `redirectionText` variable in the language-specific `.json` files, located in the `locales` directory of the customization project. To update the message, edit each of the language-specific files.

## Using the MySessionEndWidget widget

The MySessionEndWidget is provided as part of the default customization. You either use the default customization, or you can copy the three widget files (.js, .tpl.html, and .scss) to the appropriate directories in your customization project.

After making any changes to a widget file, rebuild your customization project using the `grunt` command.

## Migrating from GDC to GWC-JS

There are features supported by the Genero Desktop Client (GDC) that are not fully supported by the Genero Web Client for JavaScript (GWC-JS). Understanding these limitations and planning for them during the migration will greatly help your migration efforts.

Read about the HTML5 theme and understand the features and limitations prior to migrating from the GDC application to the GWC-JS. This will help you determine whether the GWC-JS is a suitable front-end for your application.

**Note:** This document does not intend to be complete. It gives you tips and recommendations based on experience.

### Evaluate your GWC-JS application needs on page 245

### Migration tips on page 249

## Evaluate your GWC-JS application needs

Even if most Genero Desktop Client (GDC) features are supported in Genero Web Client for JavaScript (GWC-JS), you still have points to consider before migrating.

Before you migrate your application for the GWC-JS, examine the limitations and authentication specifics. You must modify your application to handle unsupported features. See **Features and limitations** on page 187.

One method of evaluating is to run the application and see what happens! If you encounter an issue, you need to determine if this is a limitation, a customization, or a bug. See **Migration tips** on page 249 for hints aimed at helping you solve your issues during migration.

## Evaluate limitations

These limitations are grouped under the headings rendering and behaviors.

### Rendering on page 246
• Pop-up windows on page 246
• Customization on page 247
• Behaviors on page 247
  • File transfer on page 247
  • GDC front-end calls on page 247
  • GDC embedded APIs on page 247
  • Printing to local printer on page 247
  • Keystrokes on page 248
  • License count on page 248
  • Evaluate authentication on page 248

Rendering

More than likely your application is capable of displaying on a number of different display sizes and devices but an exact pixel-by-pixel match with GDC can not be expected.

If in the GDC you have sized everything to fit in a certain sized area in a monitor, there is no guarantee that it will render inside a browser tab to the exact same size. It will be close but you may, for example, find scroll bars appearing. Therefore, to ensure that your application works well on your target devices, it is recommended that you review your application’s screens.

Pop-up windows

The GWC-JS is a Web client and as such it is not a fully-implemented Windows manager. It displays at most two windows at once. It always displays the top most normal (non-modal) window, and then if there is a modal window opened, it is displayed over it. This is the equivalent of running the GDC with every non modal window maximized.

If you have a situation where window B is opened over the top of window A and there is useful information that needs to be seen in window A, you should consider opening window B as modal. This can be accomplished by referencing the default.4st window style and setting the attribute STYLE="dialog" in the Genero BDL code:

```genero
OPEN window msg WITH FORM "MyWindowB" ATTRIBUTES (STYLE="dialog")
```

Window B then displays as a pop-up window.

An alternative is to consolidate windows A and windows B together as one window.

If you already have some pop-up windows in your GDC application, in the default.4st set the windowType style attribute to modal in the Genero style for Window.dialog to achieve a similar display with the GWC-JS.

  **Note:** Any style (4st) element bound to a Window may contain the windowType attribute.

Excerpt from

$FGLDIR/lib/default.4st

```xml
<Style name="Window.dialog">
  <StyleAttribute name="windowType" value="modal" />
  <StyleAttribute name="sizable" value="no" />
  <StyleAttribute name="position" value="center" />
  <StyleAttribute name="actionPanelPosition" value="bottom" />
  <StyleAttribute name="ringMenuPosition" value="bottom" />
  <StyleAttribute name="toolBarPosition" value="none" />
  <StyleAttribute name="statusBarType" value="none" />
  <StyleAttribute name="errorMessagePosition" value="popup" />
</Style>
```
Multiple Document Interface

Multiple Document Interface (MDI) is not applicable for Web applications.

Customization

It is likely you will want to apply some corporate styling to your user interface, for example, to use your corporate colors, add your logo, etc. For more information on this, please see the customization topics in the How Do I ... ? on page 226 section. Before you migrate your application to the GWC-JS, it is recommended that you try out some customization exercises. For example, change the default blue color of the user interface to a color of your choosing, and/or modify the header to display a logo, and/or change the favicon.

Behaviors

With the GDC, your application is an executable running on a front-end PC. It is therefore bound by rules of what an executable can do within the operating system. With the GWC-JS, your application is running inside a browser. As such it follows the rules of what Web pages are allowed to do inside a browser.

As the Web browser is more vulnerable to intrusive attacks, you need to take into consideration what control you, as the developer, have over how accessible and secure this environment is, and how safe data is. If you find the browser too restrictive, consider what a malicious Web site would do if it could:

• do file transfer without user permission
• perform front-end calls to your disk
• access APIs on your system
• print immediately to the local printer

This section recommends ways to perform these tasks with security in mind and suggests alternatives to perform them safely from the GWC-JS.

File transfer

File transfer in GWC-JS is not transparent. The browser requests user permission before downloading anything on the client side. For more information see FILE_TRANSFER on page 372.

GDC front-end calls

Calls by the front-end that access the desktop file systems (disk) are not supported. Check the available front-end function calls for GWC-JS in the Genero Business Development Language User Guide. The alternative is to write your own front-end call in JavaScript. For more information see the Implement front call modules for GWC - JavaScript topic in the Genero Business Development Language User Guide.

GDC embedded APIs

APIs such as WinDDE, WinMail, and WinCOM used for data exchange and communication with, for example, Microsoft® Excel®, Word®, and Outlook®, are not supported by GWC-JS. The alternative here is to use the server-side Java API solutions for Microsoft® documents, such as those provided by Apache®. For more information, see the topics about the Apache POI framework in the The Java interface section of the Genero Business Development Language User Guide.

Printing to local printer

Printing immediately to a local printer is not supported. The local printer has to be registered on the server first and print commands must originate from the back-end server. Alternatively, you can display a file that the user can print by selecting File > Print menu in the browser.
Keystrokes

Some browsers may not allow some or all keystrokes to pass through to the application, or it is possible (as in the case of tablets and smartphones) that the device does not perform keystrokes. You should review your sources and review your dependencies on keystrokes. For example, use `ON ACTION` instead of `ON KEY`, etc.

Also it is recommended to review the text messages displayed on the screen. For example, Press `<ENTER>` to continue

If you have messages like this displayed on your screens, text may need to be made more appropriate for the application.

License count

Our intent is for your GWC-JS to use the same number of licenses as a desktop application. However, there are two cases where you may observe a difference.

1. If you close a browser tab or a browser window, it does not send a signal to the `fglrunc` process to stop. It will continue to consume a license until a timeout period (default 5 minutes) is reached.

   **Note:** Even though the same occurs if you close `gdc` by killing its process, it is more probable to close a browser tab then it is to kill the `gdc` process.

   If you have a low number of licenses or run close to your limit of user licenses, you may need to add extra licenses to lessen the chance of running out of licenses when this happens. With a CPU license this is not an issue. For more information see Multi-user Licensing topics in the Genero Licensing User Guide.

2. We can identify that a program has launched another program via `RUN` (or `StartMenu`) commands. This results in the same license being used with no extra consumption of licenses. However, if you type an application URL into a browser or click on a link, there is no way of matching that up to an existing Genero program running in a browser and/or on the same PC. If you use a third party menu system, you may find that your license consumption increases if users run multiple programs. If you use a third party menu system, either use CPU based license. For more information contact your local sales office.

Evaluate authentication

Most GDC applications connect to the server with its user permissions and profile. As a result, the applications are run under this user identity.

When applications are deployed through GAS, they are launched as the user that started the GAS, unless you ask GAS to impersonate.

As the application user is different, this can have an impact on use of environment variables and permissions. The setting for the `HOME` environment variable, for example, may not be for the user you intended, and the ability to read/write files and execute sub-shells may also be affected by different user permissions.

You need to review your sources to determine how dependent you are on environment variables. If you do not already have one, it is probable that you will need to add the concept of an application user to your application. You will also need to consider how this user is authenticated. This may mean adding your own login/password dialog, or using SSO or Web server authentication.

In any case, it is recommended to migrate the application first on GDC over HTTP. If the application runs on the browser, no further action is required. Otherwise, depending on authentication needs, adaptations to your application may be needed.

**Single sign-on (SSO) authentication**

The Single sign-on (SSO) authentication mechanism now works for both Genero Desktop Client (GDC) and GWC-JS. If you were using SSO with GDC, you can continue to use it with the GAS.
Developing Web applications

Web Server authentication

Web servers such as Apache® or Microsoft® IIS provide simple authentication mechanisms that can be used for authentication. If you use the Web server’s authentication mechanism, the login details are transmitted to Genero applications by the dispatcher in an environment variable. The user login can be retrieved from the runtime environment with the FGL instruction:

```fgl_getenv("FGL_WEBSERVER_REMOTE_USER")```

For more details, see the `fgl_getenv` topic in the `Built-in functions` chapter of `Genero Business Development Language User Guide`.

Application login

With an application login, there is nothing additional to do as the application handles the login.

Migration tips

When migrating an application from the Genero Desktop Client (GDC) to the Genero Web Client for JavaScript (GWC-JS), you may encounter issues. Here are some tips that may help.

Topics:

- Configuration issues on page 249
- Rendering issues on page 250
- Application issues on page 250
- Network issues on page 250
- Web Components on page 250

Configuration issues

The Genero Application Server (GAS) is likely serving several types of applications, which might use different versions of the Genero Business Development Language and different databases. You need to configure the GAS for the environment your application will be running. This is done through a configuration file.

If you already have your application configured for GDC-HTTP, you need to enable the rendering for GWC-JS.

Example of configuration for a GDC application:

```
<APPLICATION Id="my-app" Parent="defaultgdc">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

To allow GWC-JS rendering, replace `defaultgdc` by `defaultgwc`, and add the `UA_OUTPUT` for the public image resources and the GWC-JS customization:

```
<APPLICATION Id="my-app" Parent="defaultgwc">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
  <UA_OUTPUT>
```

See How to implement Single sign-on (SSO) on page 122 for more details.
See the Configuring applications on GAS on page 100 topic to configure your execution environment, the database access and the resource deployment (pictures, reports, and so on).

If the application fails to start:

1. Look in the dispatcher log for error messages.
2. Start a debug console as explained in the What if the application doesn't start? on page 65 topic.
   
   With a console, the GAS does not launch the application, but instead launches a console with the application environment set up. You can then compare the environment variables to your GDC working environment.

3. With a console started you can run commands to check and update your environment variables.
4. You can also display the application on the GDC by changing the FGLSERVER environment variable to verify that the application runs properly with GDC.

Rendering issues

Graphical widgets that are not rendered properly or displayed with a different style are considered rendering issues. Try to identify the source of the rendering issue. This could be the browser, your code, or some GWC-JS customization. For help with troubleshooting see Rendering issues on page 112.

Application issues

Application issues are behavior related. Most of the time you need to contact your local support center. But you can first check for:

- Relevant messages in the logs
- JavaScript errors

If clues are provided in either the logs or in the errors, try to build a simple test program that replicates the issue and contact your local support center.

Network issues

Once the migration is almost done and you are in load tests phase, you might encounter sporadic disconnections. Carefully read the chapters on GAS installation for your Web server and verify the Web servers and GAS timeouts are compatible. You can also have a look at:

- Web server log (error/access log)
- The Genero Application Server logs
- Network sniffer (like Wireshark)

Web Components

The default path for a Web component is appdir/webcomponents, where "appdir" is the application directory. See the WEB_COMPONENT_DIRECTORY element in your FGLASDIR/etc/as.xcf configuration file:

```xml
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
  [...]
  <DVM>${res.dvm.wa}</DVM>
  <WEB_COMPONENT_DIRECTORY>${(application.path)/webcomponents;
                             $(res.path.as)/webcomponents}</WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```
You can change the default Web components location by configuring a `WEB_COMPONENT_DIRECTORY` element in your application's configuration. In this example, the Web component is no longer located in `appdir/webcomponents` but in `appdir/mycomponents`.

```xml
<Application Parent="defaultgwc" …>
  <Execution>
    <Path>/home/myapp</Path>
    <Module>myapp</Module>
    <WEB_COMPONENT_DIRECTORY>/home/myapp/mycomponents</WEB_COMPONENT_DIRECTORY>
  </Execution>
</Application>
```

### Migrate from GWC-HTML5 to GWC-JS

Moving your Genero Web Client for HTML5 (GWC-HTML5) applications to Genero Web Client for JavaScript (GWC-JS) involves some changes to the customization used, topics to help your migration efforts.

**Note:** It is important to review the features and limitations prior to migrating from the GWC-HTML5 application to the GWC-JS. This topic provides you with tips and recommendations based on migration experience.

### Navigating Open Applications

With GWC for HTML5, each application started with `RUN` or `RUN WITHOUT WAITING` opens a new tab in your browser. GWC for JavaScript provides a side bar panel. Each application started with `RUN` or `RUN WITHOUT WAITING` replaces the application in the current window. You can access the other applications and make them current by selecting them from the side bar panel. See GWC-JS side bar on page 199.

### File Upload to Server

To upload a file with GWC-HTML5 Web client, you used an EDIT field with the style `FileUpload` to create a file chooser dialog.

```xml
EDIT sfile1=formonly.sfile1, style="FileUpload";
```

**Important:** The `FileUpload` style is not supported by GWC-JS.

To migrate from GWC-HTML5 to GWC-JS, you need to remove `FileUpload` style and add a call to `openFile` front call, followed by call to `fgl_getfile`. This is the same file upload method as you use in Genero Desktop Client (GDC):

1. Use the `openFile` front call to open the file chooser dialog so that the user can select a file to upload.

```xml
DEFINE infile STRING
CALL ui.Interface.frontCall("standard", "openFile",
    ["c:\fjs\doc", "doc.pdf", ".pdf", "Choose a file to upload"],
    infile)
```

**Note:** For GWC-JS, the path parameter is ignored, and wildcards can only hold one type of file extension. For more information on the use of this command, please see the "Standard front calls" section in the Genero Business Development Language User Guide.
2. Use the `fgl_getfile` to upload the file the user has chosen. The call to the `fgl_getfile` function requires no interaction from the user so it can be called immediately to upload the file to a directory specified in the application server.

```
TRY
  CALL fgl_getfile(infile, /opt/myapp/received_files)
  CALL fgl_winmessage("File uploaded", infile, "info") # Display a window with message after uploading
CATCH
  ERROR sqlca.sqlcode, " ", sqlca.sqlerrm # Catch runtime execution errors from the SQLCA diagnostic record
END TRY
```

---

**Genero Web Client for HTML5 (GWC-HTML5)**

GWC-HTML5 delivers your applications over the Web using browser-based themes. This version of the GWC has been deprecated; new development should use the Genero Web Client for JavaScript (GWC-JS) instead.

**GWC for HTML5**

The GWC-HTML5 is installed as part of the Genero Application Server.

The GWC-HTML5 allows you to deliver true Web applications with applications developed in the Genero Business Development Language (BDL). Having the underlying source written in Genero BDL means that the GWC is flexible enough to let you build from a simple Web application to a corporate Web application with only a few limitations. It brings BDL applications to the Internet world and the ability to be integrated in a Web site.

GWC-HTML5 uses themes, comprised of **templates** and **snippet sets**, to create dynamic web pages.

**Note:** For details regarding these themes, consult the *Genero Application Server 2.50 User Guide*. 
The GWC-HTML5 is no longer the default client for all Genero Web Client applications. The GWC-JS is another Web client that is also installed as part of the Genero Application Server, see Genero Web Client for JavaScript (GWC-JS) on page 184.

How the GWC-HTML5 works
The GWC-HTML5's snippet-based rendering engine (SBRE) creates application Web pages dynamically on the Application Server for delivery to the client browser using technologies, such as Client-Side Framework, understood by a browser. The AUI tree (provided by the DVM) and the template and snippet files (set by the application configuration) are used to create an XML / xHTML document that is passed to the user agent.

It can deliver the application to any device equipped with a Web browser.

Accessing Genero Web Services
Describes methods and options for accessing, validating, and delivering Web Services applications on the GAS.

For information on Web service configuration see:
- Configure applications for Web service on page 107
- SERVICE_LIST on page 405

For more information on creating Web service client or server applications with Genero BDL, see the Genero Business Development Language User Guide.
- Web services URI information on page 253
- Service invalidation on page 254
- Configure sticky Web services on page 254

Web services URI information
Examples of the URIs used to access Web Services.

In the following examples, "appid" would be replaced by the application Id and "service" would be replaced by the name of the service.

To get the WSDL for a specified service:

http://appserver:6394/ws/r/appid/service?WSDL

To access the Web service:

http://appserver:6394/ws/r/appid/service

If the Web service uses a group:

http://appserver:6394/ws/r/groupid/appid/service

Access through a Web Server:

http://<webservice>/gas/ws/r/appid/service
Service invalidation

Knowing when the configuration is invalid and when it must be modified before the service can start, is provided by the service invalidation feature described here.

A Web service is typically an automated process, called on behalf of a user or process. When a Web service has an invalid configuration, there is no graphical message informing an end user of the error. Requests continue to fail. To prevent requests from failing forever, if the Genero Web Services service (gwsproxy) cannot start within DVM_AVAILABLE time, it informs the dispatcher.

For troubleshooting and monitoring Web services, please see Get list of services invalidated by dispatcher on page 59.

The next time a request is received, the dispatcher remembers that the gwsproxy cannot be started and sends a 503 HTTP status to the user agent.

The first time a service is not able to start, this message is returned:

Application or service has been stopped due to a fatal error

Any new request returns this message until the configuration is modified and corrected:

Bad configuration prevents application or service to start.

The service resumes when the configuration is updated.

If the service is configured in the Genero Application Server configuration file, you must restart the Genero Application Server.

If the service is configured in an external application configuration file (xcf), the configuration is reloaded each time it is modified. You do not need to restart the Genero Application Server.

This feature is only supported for Genero Web services. It is not supported for Genero Desktop Client or Genero Web Client applications.

Configure sticky Web services

You create a sticky Web service to ensure that requests from a specific user agent are always routed to the same DVM handling the Web service by the Genero Application Server (GAS).

Sticky Web services are set by HTTP cookies.

The first time a user agent connects to the Genero Application Server, the gwsproxy starts a new DVM to handle the sticky Web service instance. The gwsproxy sets an HTTP cookie, called GWS_S_SESSION, in the first response to the user agent.

This cookie is then sent with any further requests from the user agent to the GAS. The gwsproxy can use the cookie to identify the DVM in charge of that cookie and dispatch the request to the correct DVM.

**Important:** Genero Web service DVMs are managed in a POOL on page 395 by the GAS. When sticky mode is used, the pool is disregarded. For instance, the MAX_AVAILABLE setting, limiting the number of DVMs available to the GWS, is no longer taken into account. Therefore, you must handle the stopping of the sticky Web service in your application code. See below.

Configure a sticky Web service

To enable sticky sessions in a Web service, add the mode attribute to the application node in the Web service application configuration, and set the value to "sticky". The KEEP_ALIVE element specifies the session lifetime, in seconds.

```
<APPLICATION Parent="ws.default" mode="sticky">
  <EXECUTION>
    <PATH>$(res.path.app)/services/sticky-ws-service</PATH>
    <MODULE>sticky-main</MODULE>
  </EXECUTION>
  <TIMEOUT>
    ...</TIMEOUT>
</APPLICATION>
```
You can change the name of the cookie by setting the environment variable `FGL_GWS_PROXY_COOKIE_NAME` in the application configuration with the value of the new cookie name.

```xml
<APPLICATION Parent="ws.default" mode="sticky">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="FGL_GWS_PROXY_COOKIE_NAME">new_name</ENVIRONMENT_VARIABLE>
    <PATH>$(res.path.app)/services/sticky-ws-service</PATH>
    <MODULE>sticky-main</MODULE>
  </EXECUTION>
  <TIMEOUT>
    <KEEP_ALIVE>60</KEEP_ALIVE>
  </TIMEOUT>
</APPLICATION>
```

**When does a session expire**

The `<KEEP_ALIVE>` element specifies the session lifetime, in seconds. A DVM that does not get a request from the user agent it is servicing will stop after the time specified by `<KEEP_ALIVE>` has passed.

If a request comes from the user agent after the time specified by `<KEEP_ALIVE>`, and therefore after the DVM has stopped, it will return an HTTP 400 error.

**Stop a sticky Web service**

To properly stop a sticky Web service, have a dedicated method to be called by the user agent when it needs to close the session. The DVM can respond to the request, unset the cookie, and stop the Genero program properly.
Deploying with Genero Archive

The Genero Archive deployment feature provides a simple process for packaging applications and services into an archive to deploy on a GAS installation. After deployment, the applications and services packaged are available and can be used.

The deployment framework provides an interface to:

- Deploy applications and services packaged into a Genero Archive
- List deployed archives
- Disable a deployed archive
- Enable / disable applications and services provided by an archive
- List enabled / disabled applications and services
- Undeploy a deployed archive

The deployment of applications and services does not include:

- Database installation and setup
- Any Genero software packages installation and setup
- Any other form of external dependencies

Any operation on Genero Archive MUST be performed in mutual exclusion to ensure the deployment process and archive management safety.

- What is a Genero Archive? on page 256
- Quick start: deploying applications on page 257
- Genero Archive lifecycle on page 263
- The MANIFEST file on page 263
- File system layout of a deployed archive on page 264
- Genero Archive procedures on page 265
- Genero Archive deployment service on page 271

What is a Genero Archive?

A Genero Archive is a zip archive containing a MANIFEST file providing installation instructions and the list of application and services to make available.

The Genero Archive is a simple zip file of a directory (or files). The archive name can be any name you wish. It need not reflect the applications or services contained within.

The MANIFEST file provides deployments instructions, and must be at the root of the archive tree.

It is possible to embed external tools into a Genero Archive. The content is not strictly restricted to compiled Genero applications. Taking advantage of this capability remains the responsibility of the user packaging the applications (portability considerations, and so on.)

Example

An example Genero Archive file could include:

- ./MANIFEST
- ./modules/app.42m
- ./modules/app.42r
- ./forms/app.42f
- ./xcf/app.xcf
In the previous example, the files were organized within the root directory by a series of sub-directories. Such directories are not required. We could have placed all the files directly in the root and have provided the following archive contents for the same application:

- ./MANIFEST
- ./app.42m
- ./app.42r
- ./app.42f
- ./app.xcf

**Quick start: deploying applications**

The procedures in this section provide you with methods to deploy applications and services.

- Resource deployment overview on page 257
- Paths to application resources on page 258
- Quick start: Genero Archive on page 259
- Deploy application resources for GWC for HTML5 on page 261
- Deploy application resources for GWC-JS on page 262

**Resource deployment overview**

Before deploying applications, it is recommended that you plan how images are going to be used by your applications so as to take advantage of the optimization and caching feature provided by the GAS for Web and GDC applications.

For examples, you can divide your images into these two categories:

**Public images**
Images that are common or that can be shared by all your applications

**Private images**
Images that are private or specific to an application

**Public images shared by your applications**

Images (e.g. logos, background images, etc.) that are common to all or several of your Genero Web Client for JavaScript (GWC-JS) applications and that do not change during an application's lifetime, are considered "public".

This means that at runtime images found in subdirectory paths of the $\{res.public.resources\}$ and the $\{res.appdata.path\}/public/deployment$ directory (see Paths to application resources on page 258) will be put in the browser cache where they can be delivered quickly to the front end without having to access an application's configuration file each time.

You can deploy public images in either of the following ways:

- Using the fglgar tool. (Recommended):

Before running fglgar, place public images in a specific directory in the application archive, see Building an archive with public resources on page 260. The resources switch of fglgar (e.g. fglgar --resources) specifies the directory in the archive with your public images. Then when you deploy the archive, it is deployed as follows:

- A subdirectory is created in your $\{res.appdata.path\}/deployment$ directory identified by the archive name and the date and time deployed, e.g. $\{res.appdata.path\}/deployment/myApp1_20150423-130838$.

  The application configuration file (xcf) and all the source files contained in the archive (gar) are placed in this directory.
• Another subdirectory is created in your \$(res.appdata.path)/public/deployment/ directory identified by just the archive name, e.g. \$(res.appdata.path)/public/deployment/myApp1.

All the image files contained in the archive's resource directory are copied into this directory and the xcf files of the archive will get updated in order to have their public directory resource \$(res.public.resources) set to this directory.

• Or if you would prefer not to use fglgar, you must then copy your public images by hand into the directory specified in your `PUBLIC_IMAGEPATH`.

Caution: Public images should not be placed in the /public root directory as the `fglrun` does not look for images to be served via the GAS there. Searches start in subdirectory paths, public/common and public/deployment.

Sample application and image deployment

```plaintext
###\$(res.appdata.path)
#    ###deployment
#    #    ###myApp1_dateTimeStamp (deployed application and private images)
#    #    #    ###webcomponents (deployed application's Web components)
#    #    ###public
#    #    ###common (default PUBLIC_IMAGEPATH directory containing public images)
#    #    #    ###deployment
#    #    #    ###myApp1 (public images deployed with application by fglgar)
```

Private images of your application

Private images are resources that are only used by one application. Before deploying, you should place them in the root directory of the application's archive. Then when you deploy the application with fglgar, they will be placed in the \$(res.appdata.path)/deployment/ directory created for the application. The `fglrun` automatically searches for application resources in the application's root directory.

Note: If you change your application's private resource location down into subdirectories of the root, you will need to ensure that the environment variable FGLIMAGEPATH is included in the application's configuration file (.xcf) and is configured correctly, as shown in the example.

```xml
<ENVIRONMENT_VARIABLE Id="FGLIMAGEPATH">pics$(sep)images$(sep)private/images</ENVIRONMENT_VARIABLE>
```

You can specify a hierarchy of directories, e.g. "private/images", but then only the \$(root)/private/images directory will be searched by `fglrun` for images, not the parent directory. If there are resources in several subdirectories then you have to specify each subdirectory separately (note \$(sep) is a built-in path delimiter which can be used for both Windows® and UNIX™ platforms). For more details on FGLIMAGEPATH, see the *Genero Business Development Language User Guide*.

Paths to application resources

Describes paths to directories specific to application resource used by the DVM.

Starting with Genero 3.00, three new predefined resources have been added:

• A public resource path for all applications, \$(res.appdata.path)/public
• A resource for common images used by applications, \$(res.public.resources), which expands to \$(res.appdata.path)/public/common
• A resource where the Application Server stores files temporarily during file transfer, $(res.path.tmp), which expands to $(res.appdata.path)/tmp/$(dispatcher.name)

At dispatcher start up, a public directory with write permissions located in $(res.appdata.path)/public is created (if it doesn't already exists) which contains two sub directories:

• common
• deployment

These directories will contain public images (i.e. images common to all applications) that are cacheable by the browser and from where they can be delivered quickly to the front-end without having to access an application's configuration file each time.

For details about deploying public images, see Resource deployment overview on page 257 and Building an archive with public resources on page 260.

Note: To ensure that resources are cacheable by the browser, the default public resource settings, /public and /public/common, are not configurable.

Quick start: Genero Archive

Follow these steps to quickly archive and deploy applications and services.

As a prerequisite, you must have your applications and/or services created and tested. You must have created an external configuration file for each application or service.

This initial procedure provides you with a quick overview of the main tasks of archiving and deploying an application followed by some examples of building archives and deploying applications:

• Building an archive with public resources on page 260
• Building an archive with deployment triggers on page 260
• Deploy your application on your machine on page 260
• Run the deployed application on page 261

For a full understanding of what Genero archiving offers, please read all archiving topics.

1. Consolidate all necessary files under a root directory.

Within this directory, you must have a subdirectory for public images if you are deploying public images with your application. See Building an archive with public resources on page 260. You can also have sub-directories as needed to assist with the organization of your files.

Note: When you create the archive, you will specify a root directory only. All files and directories within that directory will be included in the archive. If you wish to add a subset of the files in a directory, you can use another zip tool to create the file. See Create a Genero Archive on page 265.

2. Update the application's configuration file (xcf) to set all resources relative to the resource $(res.deployment.path). See Create a Genero Archive on page 265.

3. Create a MANIFEST file by hand (optional) or (recommended) use the deployment framework feature of fglgar to automatically generate the MANIFEST.

See The MANIFEST file on page 263 and The fglgar command on page 308.

4. Create the archive file.

See Create a Genero Archive on page 265.

5. Deploy the archive file.

If you are on the application server, you deploy with the gasadmin command. See Deploy an archive on page 266.

If you are on a remote computer, you can deploy with the FGL tool, PublishGar, or you can use other tools such as curl. See Genero Archive deployment service on page 271.

6. Enable the archive.

See Activate (enable) a deployed archive on page 268.
Deploying with Genero Archive

The applications, services and resources included in the archive are available for your end users.

Building an archive with public resources

About this task:
This procedure guides you through steps for building an archive for an application that is deployed with public images. At this stage it is assumed you have:

• Consolidated all the necessary files for your application archive under a root directory.
• Updated the application's configuration file (e.g. xcf) to set its deployment <PATH> element to the resource $\{res.deployment.path\}$.

1. Put your application's public images in a dedicated directory (you can name it, for example, "myAppPublicImages") in your archive.
2. Create the MANIFEST file by hand (optional) or (recommended) use the deployment framework feature of fglgar to automatically generate the MANIFEST, as shown in the next step.

   The RESOURCES element specifies your public images directory. See Resource deployment overview on page 257 and The MANIFEST file on page 263.
3. Create the archive (gar) file to deploy your application by typing the following in a terminal window of the archive directory:

   fglgar --gar --resource myAppPublicImages --application myApp.xcf

   A Genero archive file (gar) is created in your current directory that has the same name as the directory. For more information about fglgar, see The fglgar command on page 308.

Building an archive with deployment triggers

About this task:
This procedure guides you through steps for building an archive for an application that is deployed with trigger commands or programs to execute when deploying and undeploying your application on the GAS. Before you begin make sure you have:

• Consolidated all the necessary files for your application archive under a root directory.
• Updated the application's configuration file (e.g. xcf) to set its deployment <PATH> element to the resource $\{res.deployment.path\}$.

1. Create the MANIFEST file by hand (optional) (see The MANIFEST file on page 263) or (recommended) use the deployment framework feature of fglgar to automatically generate the MANIFEST while creating the archive file, as shown in the next step.
2. Create an archive (gar) file to deploy your application.

   To have fglgar create the MANIFEST file and the archive (gar), type the following in a terminal window of the archive directory.

   Note: The --deploy-trigger and --undeploy-trigger (see TRIGGERS (for manifest) on page 263) switches contain your DEPLOY and UNDEPLOY commands, as shown in the example.

   fglgar --gar --application myApp.xcf --trigger-component cpn.wa.execution.local --deploy-trigger "fglrun mydeploy.42r" --undeploy-trigger "fglrun myundeploy.42r"

   A Genero archive file (gar) is created in your current directory that has the same name as the directory. For more information about fglgar, see The fglgar command on page 308.

Deploy your application on your machine

About this task:
Once you have created an archive for your application in the steps above, you can now deploy your application locally on your machine to test it by performing the following tasks:

1. Start the standalone dispatcher from the command line by typing `httpdispatch`.
2. Deploy your (gar) file locally on your machine.

To deploy an archive named `myApp_deploy.gar`:

```
fglrn $(FGLDIR)/web_utilities/services/deployment/bin/PublishGar http://localhost:6394 deploy myApp_deploy.gar
```

3. Enable your deployed application locally on your GAS with the PublishGar tool.

```
fglrn $(FGLDIR)/web_utilities/services/deployment/bin/PublishGar http://localhost:6394 enable myApp_deploy.gar
```

This enables the application by copying its configuration file (e.g. `myApp.xcf`) to your `$(res.appdata.path)` directory. It modifies the application's `xcf` file so that `fglrn` knows where to look first for the application-specific resources.

Run the deployed application

**About this task:**

Once you have deployed your application on your machine in the steps above, you can now run your application. If the standalone dispatcher is not already started, start it from the command line using `httpdispatch`:

In a browser enter the address of your deployed application, e.g. `http://localhost:6394/ua/r/myApp`

You should see your application displayed and be able to interact with it. You have successfully deployed an application.

Deploy application resources for GWC for HTML5

To enable access to your images and Web components add them to one of the default directories.

This topic describes features the GAS provides for deploying resources with applications for GWC HTML5v1 applications.

**Deploying application images for GWC for HTML5**

By default, the GAS looks for images in `$FGLASDIR/pic` and in the application directory (defined by the `PATH` element). To quickly enable access to your images, add them to one of these directories.

The default image directory is set by the `WEB_APPLICATION_PICTURE_COMPONENT` element, see `WEB_APPLICATION_PICTURE_COMPONENT` on page 419.

For example, if you are deploying an application using GWC for HTML5, this `WEB_APPLICATION_PICTURE_COMPONENT` defines the default image directory:

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.html5.picture">
  <PATH Id="Resource" Type="WEBSERVER">$(connector.uri)/fjs</PATH>
  <PATH Id="Image" Type="APPSERVER" ExtensionFilter="$(res.image.extensions);.less;.svg">
    $(res.path.tpl.html5)/img;$res.path.pic;$application.path</PATH>
  <PATH Id="SetHtml5" Type="APPSERVER" ExtensionFilter="$(res.web.extensions);.less;.svg" DVMFallbackAllowed="FALSE">$(res.path.tpl.html5);$res.path.tpl.common</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```
Note: For your legacy GWC-HTML5 applications, the WEB_COMPONENT_DIRECTORY configuration entry is ignored. Web components in this case are still required to be located in FGLASDIR/web/components as before. For more details on web component usage, see the Genero Business Development Language User Guide.

To explicitly override with an image path specific to the application, add a PICTURE element to the application configuration file. In this example, a PICTURE path is added for the GWC for HTML5 theme:

```xml
<APPLICATION Parent="defaultgwc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfextwa.xsd">
    <RESOURCE Id="application.path"
        Source="INTERNAL">$(res.path.demo.app)/card/src</RESOURCE>
    <EXECUTION>
        <PATH>$(res.path.demo.app)/card/src</PATH>
        <MODULE>card.42r</MODULE>
    </EXECUTION>
    <OUTPUT>
        <MAP Id="DUA_HTML5">
            <PICTURE>
                <PATH Id="Image" Type="APPSERVER">
                    $(res.path.demo.app)/card/src/photo;
                    $(res.path.pic);$(application.path)</PATH>
                </PICTURE>
            </MAP>
        </OUTPUT>
    </APPLICATION>
```

Deploy application resources for GWC-JS

To enable access to images and Web components for your Genero Web Client for JavaScript (GWC-JS) applications, you need to add them to specific $(APPDATA)/public/ directories.

Deploying application images for GWC-JS

A public data directory (see Paths to application resources on page 258) is used especially for ua protocol applications to enable quicker access to common or public resources such as images, reports, etc. belonging to all deployed applications on the GAS.

This allows the dispatcher to cache resources, which can then be served without having to access an application's configuration file each time.

For example, if your FGL application has an instruction to display a public image, such as DISPLAY "photo.jpg", you must put the image file in the default $(APPDATA)/public/common, the default PUBLIC_IMAGEPATH directory so that it can be correctly served by the GAS.

You can also use the application deployment framework method to add public images, see Resource deployment overview on page 257 and Building an archive with public resources on page 260.

Deploying web components for GWC-JS

You can add Web components manually for your applications in the default $(application.path)/webcomponents directory, or you can add them to any directory provided the WEB_COMPONENT_DIRECTORY element is correctly configured to reflect it, see WEB_COMPONENT_DIRECTORY on page 422.
Genero Archive lifecycle

A Genero Archive has five stages in its lifecycle: deployed, active, deactivated, undeployed, and cleaned.

Once you have created a Genero Archive, the lifecycle of that archive can include these five stages:

- **Deployed** - The archive is deployed but not activated.
- **Active** - Applications and services provided by an archive are available to users.
- **Deactivated** - Applications and services provided by an archive are no longer available to users. A deactivated archive can be reactivated.
- **Undeployed** - The archive is no longer deployed, and cannot be reactivated. The MANIFEST file is removed. Physical cleanup of the archive is not done.
- **Cleaned** - The archive is physically removed.

The MANIFEST file

A Genero Archive MANIFEST file is a simple XML file providing the list of applications and services to make available. The MANIFEST file MUST be included in any Genero Archive file. You must create a MANIFEST file for each archive you create.

A MANIFEST file has a MANIFEST element at its root. Child elements can include the following,

**Important:** Element order. If child elements are present, they must be set in the order listed below or as shown in the sample.

- One DESCRIPTION element. The DESCRIPTION element provides a textual description of the archive. It displays when listing archives.
- 0, or 1 TRIGGERS element. The TRIGGERS element (see TRIGGERS (for manifest) on page 263) specifies a trigger component which contains a program (e.g. fglrun) or script to run to deploy or undeploy the application if the deployment framework is used.
- 0, or 1 RESOURCES element. The RESOURCES element specifies the directory in your archive file where images that are specific to your applications are found. The directory must be in the archive.
- 0, 1 or more APPLICATION elements. The APPLICATION element takes one mandatory attribute. The xcf attribute specifies the path to the application configuration file. The application configuration file must be in the archive. Include an APPLICATION element for each application in the archive.
- 0, 1 or more SERVICE elements. The SERVICE element takes one mandatory attribute. The xcf attribute specifies the path to the Web service configuration file. The Web service configuration file must be in the archive. Include a SERVICE element for each service in the archive.

Sample MANIFEST file:

```xml
<MANIFEST>
  <DESCRIPTION>my description</DESCRIPTION>
  <TRIGGERS component='comp_name'>
    <DEPLOY>deploy.sh</DEPLOY>
    <UNDEPLOY>undeploy.sh</UNDEPLOY>
  </TRIGGERS>
  <RESOURCES>res_dir</RESOURCES>
  <APPLICATION xcf='app.xcf' />
  <SERVICE xcf='app.xcf' />
</MANIFEST>
```

**TRIGGERS (for manifest) on page 263**

The TRIGGERS element defines a set of deployment parameters that can be used when deploying an application with the deployment framework. It takes a component attribute, which specifies the
Deploying with Genero Archive

Environment context for the runtime (fglrunt) required by the DEPLOY and UNDEPLOY elements. These child elements specify commands to execute when deploying and undeploying applications on the GAS.

Syntax

```xml
<TRIGGERS component = 'comp_name'>
  <DEPLOY> myDeploy </DEPLOY >
  <UNDEPLOY> myUnDeploy </UNDEPLOY>
</TRIGGERS>
```

Child elements

The TRIGGERS element may contain the following child elements:

1. Zero or one DEPLOY element.
2. Zero or one UNDEPLOY element.

Example

```xml
<TRIGGERS component = 'cpn.wa.execution.local'>
  <DEPLOY>fglrunt mydeploy.42r</DEPLOY>
  <UNDEPLOY>fglrunt myundeploy.42r</UNDEPLOY>
</TRIGGERS>
```

In this example, the component value - cpn.wa.execution.local - can be referenced by the --trigger-component and the DEPLOY value by --deploy-trigger when you are creating an application archive using fglgar, for example:

```bash
fglgar --gar --application myapp.xcf --trigger-component cpn.wa.execution.local --deploy-trigger "fglrunt mydeploy.42r" --undeploy-trigger "fglrunt myundeploy.42r"
```

Note:

- In the case that the --deploy-trigger fails, the entire deployment will fail. Whereas if the --undeploy-trigger fails, the undeployment is still carried out.
- Deployment triggers are typically not required, you can deploy your applications without them by simply specifying the application's xcf file. See The MANIFEST file on page 263.

Parent elements

This element is a child of the MANIFEST element in The MANIFEST file on page 263

File system layout of a deployed archive

A Genero Archive contains application source files organised into subdirectories for modules, forms, etc. includes the MANIFEST and the external application configuration file. The directory structure is recreated when deployed.

Genero Archives are unpacked into a deployment directory. The deployment directory path is available and configurable in the GAS configuration files. The resource res.deployment.root points to the root directory for deployed archives. By default, it is configured with the value $(res.appdata.path)/deployment.

Each archive is unpacked into a dedicated directory under the $(res.deployment.root) directory. By default, the deployment tool will use the archive name (without the file extension) as the deployment directory. The directory name is completed with a timestamp representing the installation date.
Example

If the Genero Archive file included these files (in the same root directory):

- ./MANIFEST
- ./modules/app.42m
- ./modules/app.42r
- ./forms/app.42f
- ./xcf/app.xcf

The directory structure after unpacking the Genero archive would be similar to this, with the current timestamp used:

- <APPDATA>/deployment/myapp-20130522-123456/MANIFEST
- <APPDATA>/deployment/myapp-20130522-123456/modules/app.42m
- <APPDATA>/deployment/myapp-20130522-123456/modules/app.42r
- <APPDATA>/deployment/myapp-20130522-123456/forms/app.42f
- <APPDATA>/deployment/myapp-20130522-123456/xcf/app.xcf

Genero Archive procedures

The procedures provide you with the instructions to create and use your Genero Archives when on the application server host.

**Tip:** Use the Genero Archive deployment service, see Genero Archive deployment service on page 271, when working from a remote server.

- Create a Genero Archive on page 265
- Deploy an archive on page 266
- List all deployed archives on page 267
- Activate (enable) a deployed archive on page 268
- Deactivate (disable) a deployed archive on page 269
- Undeploy a deployed archive on page 269
- Clean up undeployed archives on page 270
- Upgrade an archive on page 271

Create a Genero Archive

A Genero Archive is a zip archive containing a MANIFEST file providing installation instructions and the list of application and services to make available.

As a prerequisite, all of your application files, along with your application configuration (xcf) file, must sit under one directory. For example, this would be a valid organization of files:

- ./fuzzy/modules/app.42m
- ./fuzzy/modules/app.42r
- ./fuzzy/forms/app.42f
- ./fuzzy/xcf/app.xcf
- ./fuzzy/appPublicImages/

**Note:** Public images need to go in a dedicated directory, for more information on deploying images see Resource deployment overview on page 257.

Follow these steps to create a Genero Archive from the contents of a directory. All folders and files in the directory are included in the archive.

**Note:** When you use the fglgar tool to create an archive, you are creating a zip file. The tool is designed to take a single directory as its parameter. If you need a more sophisticated archive tool
(to add only specific files to the archive, for example), you can use any other zip tool to create your GAR archive.

1. Create a MANIFEST file. See The MANIFEST file on page 263.
2. Update the application's configuration file (.xcf).
   All resources must be set relative to the resource `res.deployment.path`. For example, if your compiled files were in the `/bin` directory of your archive, you would update `<PATH>` to:
   ```
   <PATH>$(res.deployment.path)/bin</PATH>
   ```
   Complete this change for all resources used in the configuration file: pictures, templates, forms, modules, and so on.
3. Use the `fglgar` tool to create a Genero Archive.
   If you are in the directory containing your MANIFEST file and your program files:
   ```
   fglgar --gar
   ```
   This creates an archive (.gar) file with the same name as the archive directory.
   If you need to specify the directory where the archive content is located, include the `--input-source` option:
   ```
   fglgar --gar --input-source ./fuzzy
   ```
   This creates an archive file with the same name as your program, drawing its content from the `.fuzzy` directory.
   If you wish to specify a name for your archive, use the `--output` option:
   ```
   fglgar --gar --input-source ./fuzzy --output myfuzzy.gar
   ```
   This creates an archive file with the name `myfuzzy.gar`, drawing its content from the `.fuzzy` directory.
   **Note:** The archive name has no importance.

**Deploy an archive**

When you deploy an archive on a Genero Application Server, the applications and services referenced in the archive are placed on the host, yet are not yet available to users.

As a prerequisite, you have a Genero Archive file.

The archive will be unpacked in the deployment directory. Static resources are zipped for better performance at runtime.

`gasadmin` will deploy the .gar but the application deployed is named with the .xcf.

Enter the `gasadmin` command with the appropriate arguments.

To deploy an archive named `fuzzy.gar`:

```
gasadmin --deploy-archive fuzzy.gar
```
By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
  gasadmin --deploy-archive fuzzy.gar --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

**XML output example:**

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Genero Archive successfully deployed.</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to deploy the Genero Archive</ERROR>
  <ERROR>An archive with the same name is already deployed</ERROR>
</DEPLOYMENT>
```

**Text output example:**

```
Command succeeded.
Genero archive successfully deployed.
```

```
Command failed.
ERROR: Failed to deploy the Genero Archive
ERROR: An archive with the same name is already deployed.
```

All applications and services are deployed in the default group of the application server.

**List all deployed archives**

You can list all deployed archives with a single command.

Enter the `gasadmin` command with the appropriate arguments.

To list all deployed archives:

```
  gasadmin --list-archives
```

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
  gasadmin --list-archives --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

An example of text output for an archive listing:

```
Command succeeded.
2 archives deployed:
  Name : archive1
  State : Disabled
  Description : This my incredible first archive
```
Application(s):
- 22223.xcf
- 22224.xcf
Service(s):
- 15233.xcf

Name: archive2
State: Disabled
Description: This my incredible second archive
Application(s):
- 22315.xcf
Service(s):
- 17746.xcf
- 19164.xcf

An example of XML output for an archive listing:

```xml
<DEPLOYMENT success="TRUE">
  <ARCHIVE name="MyArchive" enabled="TRUE">
    <DESCRIPTION> This is my first archive </DESCRIPTION>
    <APPLICATION xcf="MyApp1.xcf" />
    <APPLICATION xcf="MyApp2.xcf" />
    <SERVICE xcf="MyServ1.xcf" />
    <SERVICE xcf="MyServ2.xcf" />
  </ARCHIVE>
  <ARCHIVE name="MyArchive2" enabled="TRUE">
    <DESCRIPTION> This is my second archive </DESCRIPTION>
    <APPLICATION xcf="app/MyApp3.xcf" />
    <APPLICATION xcf="app/MyApp4.xcf" />
  </ARCHIVE>
</DEPLOYMENT>
```

Note: The DEPLOYMENT node may contains some MESSAGE or ERROR nodes.

Activate (enable) a deployed archive

A deployed archive is not implicitly activated. When you activate (enable) a deployed archive, the applications and services provided by the archive become available to users.

Enabling an archive means enabling all applications and services provided by the Genero Archive.

1. Enter the gasadmin command with the appropriate arguments.

   To show a list of archives and details of their status:

   ```shell
   gasadmin --list-archives
   ```

   To enable an archive named fuzzy:

   ```shell
   gasadmin --enable-archive fuzzy
   ```

   By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the --xml-output option:

   ```shell
   gasadmin --enable-archive fuzzy --xml-output
   ```

   The exit status is a 0 (zero) in case of success, 1 in case of error.

2. Copy the application configuration file (xcf) into a group directory, if desired.
Deploying with Genero Archive

XML Output examples:

```xml
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully enabled</MESSAGE>
</DEPLOYMENT>
```

```xml
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to enable <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure....</ERROR>
</DEPLOYMENT>
```

Deactivate (disable) a deployed archive

When you deactivate (disable) a deployed archive, the applications and services provided by the archive are not exposed (no longer available to users).

As a prerequisite, you have a deployed archive that has been enabled.

Disabling an archive means disabling all applications and services provided by the Genero Archive. The archive remains deployed. Any running applications or services are not stopped.

**Tip:** To disable a single application or service, remove its configuration file from the group directory. The original configuration files remain available in the deployed archive directory.

Enter the `gasadmin` command with the appropriate arguments.

To disable a Genero Archive named `fuzzy.gar`:

```
gasadmin --disable-archive fuzzy.gar
```

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
gasadmin --disable-archive fuzzy.gar --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

XML output examples:

```xml
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully undeployed</MESSAGE>
</DEPLOYMENT>
```

```xml
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to undeploy <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure....</ERROR>
</DEPLOYMENT>
```

Undeploy a deployed archive

When you undeploy an archive, the archive is no longer deployed and cannot be reactivated.

An archive can be undeployed. When you undeploy an archive:

- All applications and services provided by the archive that are not disabled are disabled at this time.
- The `MANIFEST` file renames to `MANIFEST.undeployed`. Any archive with the file `MANIFEST.undeployed` represents an undeployed archive.
When you undeploy an archive, no files are removed. The cleanup of the undeployed archive directory must be explicitly requested.

Enter the `gasadmin` command with the appropriate arguments.

To undeploy an archive named `fuzzy.gar`:

```
gasadmin --undeploy-archive fuzzy.gar
```

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
gasadmin --undeploy-archive fuzzy.gar --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

**XML output examples:**

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully undeployed</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to undeploy <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure...</ERROR>
</DEPLOYMENT>
```

**Text output examples:**

```
Command succeeded.
Genero archive successfully deployed.
```

```
Command failed.
ERROR: Failed to undeploy the Genero Archive
ERROR: ... and error messages indicating the reasons of the failure...
```

**Clean up undeployed archives**

The deployment tool provides a cleanup command that physically removes undeployed archives. This process is executed on user request.

As a prerequisite ensure that applications you want to remove are not running any more. You can list the running applications with "gasadmin -l".

One or more archives have been undeployed.

The cleanup operation will only remove directories in the deployment directory if:

- The directory name matches a deployment directory name pattern, i.e. `archive-name-timestamp`
- The archive directory contains a file name `MANIFEST.undeployed`

1. Enter the `gasadmin` command with the appropriate arguments.

   To clean up all undeployed archives:

   ``
   gasadmin --clean-archives
   ```

   For each undeployed archive, you are asked to confirm the cleanup by entering `y` to clean up, `n` to skip and continue to the next undeployed archive.
By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the --xml-output option:

```
  gasadmin --clean-archives --xml-output
```

To disable confirmation for each archive removal, add the --all option:

```
  gasadmin --clean-archives --all
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

2. To quit the clean up, enter quit.

**Upgrade an archive**

You can upgrade an application without having to kill processes or wait for users to log out.

As a prerequisite, your applications and services were previously deployed as an archive. You still have the requisite archive files (see Create a Genero Archive on page 265.)

1. Deploy the new version of the archive. See Deploy an archive on page 266.
2. Disable the previously deployed archive. See Deactivate (disable) a deployed archive on page 269.
3. Enable the newly deployed archive. See Activate (enable) a deployed archive on page 268.
4. Undeploy the previously deployed archive. See Undeploy a deployed archive on page 269.

**Genero Archive deployment service**

A deployment service has been written in Genero and can be accessible on a base URL of /ws/r/services/DeploymentService. It provides several actions in the form of REST APIs.

The deployment service is a Web service provided for the Genero Application Server. To prevent this application being accessed remotely, the default configuration restricts access to the localhost (127.0.0.1). If you want to enable it for other client machines / IP addresses, you must customize the ALLOW_FROM tag or remove the ACCESS_CONTROL tag. The service configuration can be found at $FGLDIR/web_utilities/services/DeploymentService.xcf.

**Table 24: Actions available for the Genero Archive deployment service**

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>HTTP request details</th>
</tr>
</thead>
<tbody>
<tr>
<td>deploy</td>
<td>Deploys a Genero Archive on the Genero Application Server by calling gasadmin --deploy-archive. Returns the command result in XML format.</td>
<td>HTTP PUT request of following form: URL: /ws/r/services/DeploymentService/deploy?archive=name. <strong>Note:</strong> archive=name is mandatory to identify the archive on the GAS. To send the archive to the Genero Application Server, you could use curl with a PUT instruction:</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
<td>HTTP request details</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>enable</td>
<td>Enables all applications and services of a given archive by calling <code>gasadmin --enable-archive</code>.</td>
<td>HTTP GET request of following form: URL: <code>/ws/r/services/DeploymentService/enable?archive=name</code></td>
</tr>
<tr>
<td>disable</td>
<td>Disables all applications and services of given archive by calling <code>gasadmin --disable-archive</code>.</td>
<td>HTTP GET request of following form: URL: <code>/ws/r/services/DeploymentService/disable?archive=name</code></td>
</tr>
<tr>
<td>undeploy</td>
<td>Undeploys a genero archive on the Genero Application Server by calling <code>gasadmin --undeploy-archive</code>.</td>
<td>HTTP GET request of following form: URL: <code>/ws/r/services/DeploymentService/undeploy?archive=name</code></td>
</tr>
<tr>
<td>list</td>
<td>Returns a list and status of all archives available on the Genero Application Server by calling <code>gasadmin --list-archive</code>.</td>
<td>HTTP GET request of following form: URL: <code>/ws/r/services/DeploymentService/list</code></td>
</tr>
<tr>
<td>cleanup</td>
<td>The cleanup operation is not available with the REST service, as it requires user interaction to its prompts. To perform a cleanup operation, you must use <code>gasadmin</code> on the application server (performed by <code>gasadmin --clean-archives</code>).</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Upgrading

These topics talk about what steps you need to take to upgrade to the next release of Genero Application Server, and allows you to identify which features were added for a specific version.

Review the list of migration recommendations each time you move to a new version of the Genero Application Server. Failure to do so can result in issues when rendering your Web applications.

- New Features of the Genero Application Server on page 273
- Upgrade Guides for the Genero Application Server on page 287
- Migrating Templates and Snippets Customizations on page 303

New Features of the Genero Application Server

These topics provide an look back at the new features introduced with each release of the Genero Application Server.

- What's new in Genero Application Server, v 3.00 on page 275
- Genero Application Server v 2.50 New Features on page 277
- Genero Application Server 2.41 New Features on page 280
- Genero Application Server 2.40 New Features on page 280
- Genero Application Server 2.32 New Features on page 282
- Genero Application Server 2.30 New Features on page 282
- Genero Application Server 2.22 New Features on page 284
- Genero Application Server 2.21 New Features on page 284
- Genero Application Server 2.20 New Features on page 285

What's new in Genero Application Server (GAS), v 3.00 (Maintenance Releases)

A summary of new features and changes in functionality added for Genero Application Server (GAS) version 3.00 Maintenance Releases (MRs).

Important: Please read What's new in Genero Application Server, v 3.00 on page 275, for a list of features that were introduced with Genero 3.00 General Availability release.

Table 25: Genero Web Client for JavaScript (GWC-JS), Version 3.00 (MRs)

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhancements for the GWC-JS (v1.00.31):</td>
<td>No additional reference.</td>
</tr>
<tr>
<td>• Native mobile browsers are supported on iOS (Mobile Safari) and Android™ (Mobile Chrome).</td>
<td></td>
</tr>
<tr>
<td>Enhancements for the GWC-JS (v1.00.29):</td>
<td>For details about Mozilla® Firefox®, see User agents on page 61.</td>
</tr>
<tr>
<td>• Support for Mozilla® Firefox® added.</td>
<td>For details about the browserMultiPage style, see the UserInterface style attributes topic in the Genero Business Development Language User Guide.</td>
</tr>
<tr>
<td>• The browserMultiPage style can be used to specify whether the run and run without waiting instructions will be executed in the same browser tab or in a new browser tab.</td>
<td></td>
</tr>
<tr>
<td>Enhancements for the GWC-JS (v1.00.28):</td>
<td>For details about activating the debug mode, see GWC-JS.</td>
</tr>
</tbody>
</table>
Overview

- Debug mode activation is simplified.
- The `allowWebSelection` presentation style enables content selection from a table with a simple click-and-drag of the cursor.

Reference

Application in debug mode on page 197.

For details about the `allowWebSelection` presentation style, see Table Style Attributes in the Genero Business Development Language User Guide.

Enhancements for the GWC-JS (v1.00.21):

- The `grunt` command supports the creation of multiple customizations in the distribution folder.

See Compiling customization projects on page 219.

Enhancements for the GWC-JS (v1.00.19):

- Richtext is supported.
- Referencing a resource (such as an image) has been simplified. Use a path relative to `index.html`.
- Support for applications using right-to-left languages, such as Arabic, is added.

See Features and limitations on page 187, Customization deliverable on page 208, and Configure GWC-JS for right-to-left languages on page 48.

Enhancements for the GWC-JS (v1.00.16):

- Canvas elements are supported.
- Front calls `setvar` and `getvar` are supported for session variable management. See the Genero Business Development Language User Guide for more details about their usage.

See Features and limitations on page 187.

Enhancements for the GWC-JS (v1.00.16):

- The GWC-JS has a log player feature, which allows you to play back a session log in the browser.

See Replay a log with Log Player on page 201.

Table 26: Engine and Architecture, Version 3.00 (MRs)

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>GWC_JS_LOOKUP_PATH</code> element (added as a child of <code>INTERFACE_TO_CONNECTOR</code> on page 381) allows you to configure the location of your custom GWC-JS front-end.</td>
<td>See <code>GWC_JS_LOOKUP_PATH</code> on page 376</td>
</tr>
<tr>
<td>A new URI dedicated to the lookup of the GWC-JS directory. The complete format of the URI is <code>ua/w/${GWC-JS}/&lt;filename&gt;</code>.</td>
<td>See Application URIs on page 51</td>
</tr>
<tr>
<td>Enhancements for the GAS (v3.00.21):</td>
<td></td>
</tr>
<tr>
<td>• A method to add custom HTTP headers for HTTP responses is provided.</td>
<td>See HTTP on page 378, SESSION_COOKIE on page 406</td>
</tr>
<tr>
<td>• The <code>SESSION_COOKIE</code> element can be set so that cookies are created with the &quot;secure&quot; flag.</td>
<td></td>
</tr>
<tr>
<td>The GAS (v3.00.43 onwards) supports a new user agent output feature called <code>UA_OUTPUT_COMPONENT</code> which can be defined in the component for web applications. If set, an application will inherit the specified UA settings.</td>
<td>See <code>UA_OUTPUT_COMPONENT</code> on page 415</td>
</tr>
</tbody>
</table>
Table 27: Deployment, Version 3.00 (MRs)

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <strong>WEB_COMPONENT_DIRECTORY</strong> element allows for multiple paths to be specified.</td>
<td>See <a href="#">WEB_COMPONENT_DIRECTORY</a> on page 422</td>
</tr>
</tbody>
</table>

**Note:** The new features listed in this topic are available in the latest version of the GAS. Contact your support channel for more details.

What’s new in Genero Application Server, v 3.00

A summary of new features and changes in functionality added for Genero Application Server (GAS) version 3.00.

Table 28: GWC for JavaScript, Version 3.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
</table>

Table 29: Single Sign-On, Version 3.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Single sign-on (SSO) mechanism works for all clients: Genero Desktop Client (GDC) and Genero Web Client (GWC).</td>
<td>See <a href="#">What is Single sign-on (SSO)?</a> on page 38.</td>
</tr>
<tr>
<td>OpenID Connect is introduced as SSO protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under $FGLDIR/web_utilities/services/openidconnect.</td>
<td>See <a href="#">OpenID Connect SSO</a> on page 123</td>
</tr>
</tbody>
</table>

Table 30: Web Services and the GAS, Version 3.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>A new element called <strong>REQUEST_RESULT</strong> has been added to the Web services time out component, which, if set, allows the GWS proxy to release the DVM in charge of a service that has not responded within a given time frame (seconds).</td>
<td>See <a href="#">SERVICE_APPLICATION_TIMEOUT_COMPONENT</a> on page 403</td>
</tr>
</tbody>
</table>

Table 31: Engine and Architecture, Version 3.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <strong>gasadmin</strong> command has been updated with new features to manage archives. You can list, enable, deploy and undeploy archives with options of the <strong>gasadmin</strong> command.</td>
<td>See <a href="#">The gasadmin command</a> on page 310.</td>
</tr>
<tr>
<td>The GAS supports a new timeout feature called <strong>AUTO_LOGOUT_COMPONENT</strong> which can be defined in the timeout component for GWC and GDC applications. If set, an application will get a logout page or screen after a specified time (in seconds) of user inactivity.</td>
<td>See <a href="#">AUTO_LOGOUT_COMPONENT</a> on page 359</td>
</tr>
</tbody>
</table>
GAS 3.00 introduces a new universal proxy for applications using GDC v3 and GWC-JS interfaces. It is called uaproxy (ua). It replaces the gdcproxy and html5proxy proxies. It provides protocol improvements and better performance overall.

There is a new bootstrap mechanism for starting GWC-JS applications, which is used to initialize information for rendering an application.

A new element called GWC_JS_LOOKUP_PATH added to INTERFACE_TO_CONNECTOR on page 381 allows you to configure the location of your custom GWC-JS front-end.

The new user agent protocol, the uaproxy for GDC, GMA, GMI and GWC-JS, introduces a new set of resource URLs. The ua protocol does not use snippets sets or output maps as the wa protocol did previously to specify output theme.

Genero Ghost Client is a new Java framework introduced for testing different scenarios by emulating user interaction on Genero applications.

GAS 3.00 introduces a new configuration entry for the Report Viewer which allows you to configure the location of the Genero Web Report Viewer. A corresponding report viewer URL prefix /ua/grv is provided to the Genero Report Engine (GRE).

### Table 32: Deployment, Version 3.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The deployment framework (fglgar tool) provides you with new resource management features for public image files.</td>
<td>See The fglgar command on page 308</td>
</tr>
<tr>
<td>Three new predefined resources have been added:</td>
<td>See GAS directories on page 39 and Paths to application resources on page 258</td>
</tr>
<tr>
<td>• A public resource path for all applications, $(res.appdata.path)/public</td>
<td></td>
</tr>
<tr>
<td>• A resource for common images used by applications, $(res.public.resources)</td>
<td></td>
</tr>
<tr>
<td>• A resource where the Application Server stores files temporarily during file transfer, $(res.path.tmp)</td>
<td></td>
</tr>
<tr>
<td>A WEB_COMPONENT_DIRECTORY has been added, it contains the path(s) where Web components are located for an application. It replaces the WEB_COMPONENT element, which has been removed from the EXECUTION element of an application.</td>
<td>See WEB_COMPONENT_DIRECTORY on page 422</td>
</tr>
<tr>
<td>The DOCUMENT_ROOT allows multiple document root paths.</td>
<td>See DOCUMENT_ROOT on page 367</td>
</tr>
<tr>
<td>New entries in the application MANIFEST file:</td>
<td>See The MANIFEST file on page 263 and TRIGGERS (for manifest) on page 263</td>
</tr>
<tr>
<td>• A new RESOURCES entry is added to the MANIFEST file that specifies the directory in your archive file where public images for your applications are found.</td>
<td></td>
</tr>
<tr>
<td>• A new TRIGGERS element defines a set of deployment parameters that can be used when deploying an application with the deployment framework.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 33: Miscellaneous, Version 3.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The GAS can be plugged in to Internet Protocol Version 6 (IPv6) Web servers without any additional configuration if the front-ends (GWC-JS, GDC, Web Service) need to use IPv6 user agents. As communication between the GAS and Web servers is on localhost, IPv4 continues to be used.</td>
<td>Note: If you have to restrict access of some applications to specific IP addresses, in that case the <code>ACCESS_CONTROL</code> entry must be configured with IPv6 addresses. See <code>ACCESS_CONTROL</code> on page 351.</td>
</tr>
<tr>
<td>In addition to specifying access by IP address, the <code>ACCESS_CONTROL</code> and <code>MONITOR</code> configuration elements have been updated with two access control keywords (<code>NOBODY, ALL</code>) which can be used with <code>ALLOW_FROM</code>.</td>
<td>See <code>ALLOW_FROM</code> on page 353.</td>
</tr>
<tr>
<td>Introducing a new resource, <code>res.access.control</code>, that specifies access control for Web services, application programs, such as demos, and <code>MONITOR</code>. It is defined with the keyword <code>NOBODY</code> by default.</td>
<td>Note: The default deployment does not allow access to demo programs and <code>MONITOR</code>. Access has to be configured explicitly with <code>ALLOW_FROM</code>. See <code>MONITOR</code> on page 385 and <code>ACCESS_CONTROL</code> on page 351.</td>
</tr>
<tr>
<td>A new element called <code>ROOT_URL_PREFIX</code> added to <code>INTERFACE_TO_CONNECTOR</code> on page 381 supports the use of reverse proxy server between the client and the GAS. It allows for a URL prefix to be specified for the Web server so as to provide the correct interface to the client.</td>
<td>See <code>ROOT_URL_PREFIX</code> on page 402.</td>
</tr>
</tbody>
</table>

### Genero Application Server v 2.50 New Features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.50.

### Table 34: GWC for HTML5, Version 2.50

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The HTML5 is the default theme for all Genero Web Client applications. The AJAX, Silverlight, iPhone® and Basic themes are deprecated.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Browser-based themes, in the Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>HTML5 theme: while you can still customize your Genero Web Client application with CSS, the files involved have changed and any previous customization efforts will need to be revisited.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topics:</td>
</tr>
<tr>
<td>The HTML5 theme adds support for frozen columns, splitters (in forms, between the form and the</td>
<td>• Cascading Style Sheets</td>
</tr>
<tr>
<td></td>
<td>• HTML5 theme</td>
</tr>
<tr>
<td></td>
<td>in the Genero Application Server 2.50 User Guide.</td>
</tr>
</tbody>
</table>
Upgrading

Overview

**actionPanel**, between the form and the startMenu), and GridChildrenInParent.

The html5proxy manages Genero Web Client applications using the HTML5 theme.

The HTML5 theme supports StartMenus and applications displayed in folder tabs.

Add language support to a snippet file (HTML5).

Start an application while ignoring the application's stored settings (HTML5).

GDCAX is deprecated in favor of the HTML5 theme. GDC HTTP is still supported.

Control the folder size of an application rendered by the HTML5 client.

UI enhancements for the GWC (HTMTL5 theme):

- The tooltip displays beside the field. An icon (triangle) allows you to show or hide the tooltip. You can also hide the tooltip by clicking on the message text.
- The display of the calendar widget has been improved to prevent overlapping with other widgets. While it usually displays under the date field, it displays to the side if there is not enough space under.

The browser back and forward buttons can trigger actions.

The demos application is restricted to localhost by default.

---

**Table 35: Single Sign-On, Version 2.50**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kerberos is deprecated. Promoted SSO solutions are OpenID and SAML.</td>
<td>See What is Single sign-on (SSO)? on page 38.</td>
</tr>
<tr>
<td>Security Assertion Markup Language (SAML) is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service</td>
<td>See SAML SSO on page 136.</td>
</tr>
</tbody>
</table>
and is delivered in the Genero Web Services package under $FGLDIR/web_utilities/services/saml.

OpenID is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under $FGLDIR/web_utilities/services/openid.

Delegate the start of a GWC (/wa/r) or GDC (/ja/r) application or a GWS (/ws/r) service to another REST service in order to perform some controls (such as authentication, authorization, or monitoring) in a single and centralized Genero program.

Table 36: Web Services and the GAS, Version 2.50

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for sticky Web services.</td>
<td>See Configure sticky Web services on page 254.</td>
</tr>
<tr>
<td>Service invalidation provides notification when a Genero Web Service</td>
<td>See Service invalidation on page 254.</td>
</tr>
<tr>
<td>configuration is invalid. Once identified as having invalid</td>
<td></td>
</tr>
<tr>
<td>configuration, the dispatcher will not attempt to start the gwsproxy</td>
<td></td>
</tr>
<tr>
<td>for the service until the configuration is modified.</td>
<td></td>
</tr>
</tbody>
</table>

Table 37: Engine and Architecture, Version 2.50

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better internal/temporary data organization.</td>
<td>See GAS directories on page 39.</td>
</tr>
<tr>
<td>The gasadmin command has been updated to handle multiple dispatchers.</td>
<td>See The gasadmin command on page 310.</td>
</tr>
<tr>
<td>You can target the dispatcher with the -d option of the gasadmin command.</td>
<td></td>
</tr>
<tr>
<td>DVM standard error and standard output are directed to dedicated log</td>
<td>See Logging on page 167.</td>
</tr>
<tr>
<td>files.</td>
<td></td>
</tr>
<tr>
<td>The Genero Application Server on Linux®/UNIX™ will use domain sockets</td>
<td>See SOCKET_FAMILY on page 408 and SOCKET_PATH</td>
</tr>
<tr>
<td>to communicate between the dispatcher and the proxies.</td>
<td>on page 409.</td>
</tr>
<tr>
<td>Enable and disable resource compression in the int.cfg file. Compress</td>
<td>See Compression in Genero Application Server</td>
</tr>
<tr>
<td>resources (static files) using the -z option of the gasadmin command.</td>
<td>on page 154 and The gasadmin command on page</td>
</tr>
<tr>
<td></td>
<td>310.</td>
</tr>
</tbody>
</table>

Table 38: Deployment, Version 2.50

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genero Archive provides a means for packaging and deploying applications.</td>
<td>See Deploying with Genero Archive on page 256.</td>
</tr>
</tbody>
</table>
Table 39: Miscellaneous, Version 2.50

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genero Front Call ActiveX is deprecated.</td>
<td>No additional reference.</td>
</tr>
</tbody>
</table>

**Genero Application Server 2.41 New Features**

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.41.

**Table 40: Genero Application Server Version 2.41 New Features**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>When using GWC for Silverlight, if the toolbar is larger than the form, the default behavior is to have a left and right button on the toolbar (horizontal scrolling).</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.41 User Guide.</td>
</tr>
<tr>
<td>GWC for HTML5 is a supported theme. It is no longer considered a &quot;preview version&quot;.</td>
<td>The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the topic, Html5 Theme, in the Genero Application Server 2.41 User Guide.</td>
</tr>
<tr>
<td>Important: The default theme for the next major release of Genero will be GWC for HTML5. Both the AJAX and iPhone themes will be deprecated in the next major release. If you are developing applications for the iPhone, it is recommended that you use the HTML5 theme, which is fully supported on iOS.</td>
<td></td>
</tr>
<tr>
<td>The Basic theme is deprecated. There will be no further development on the Basic theme. It is recommended that you use the HTML5 theme for future development.</td>
<td>No additional reference.</td>
</tr>
<tr>
<td>The GWC hybrid mode takes Genero applications and delivers them as native applications for Android or iOS mobile platforms.</td>
<td>The supporting topic has been removed, as GWC hybrid mode is deprecated. Please see the topic, Gnero Web Client hybrid mode, in the Genero Application Server 2.41 User Guide.</td>
</tr>
<tr>
<td>New template paths: Application hierarchy, Window hierarchy, Layout hierarchies, and Widgets hierarchies have been added.</td>
<td>The supporting topics have been removed, as the template paths theme is deprecated. Please see the topics:</td>
</tr>
<tr>
<td></td>
<td>• Template Paths - Application hierarchy</td>
</tr>
<tr>
<td></td>
<td>• Template Paths - Window hierarchy</td>
</tr>
<tr>
<td></td>
<td>• Template Paths - Layout hierarchies</td>
</tr>
<tr>
<td></td>
<td>• Template Paths - Widgets hierarchies</td>
</tr>
<tr>
<td></td>
<td>in the Genero Application Server 2.41 User Guide.</td>
</tr>
</tbody>
</table>

**Genero Application Server 2.40 New Features**

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.40.

**Table 41: 2.40 New Features: Common**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support to add summary lines on TABLEs defining AGGREGATE form fields for the Genero Web Client.</td>
<td>Please see the topic, Features and limitations, in the Genero Application Server 2.40 User Guide.</td>
</tr>
</tbody>
</table>
The Genero Application Server can handle submit parameters with the POST method.

Performance improvements on the GAS have been made to provide users with a better experience using Genero Web Applications.

- Less memory consumed for GWC rendering
- Less data traffic for tables and windows
- Better stack management in threads

---

### Table 42: 2.40 New Features: Silverlight theme

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>Reference</td>
</tr>
<tr>
<td>The Silverlight theme layout is improved to closely match the layout proportions (widgets size, widgets redesign after window resizing, etc.) found in Genero Desktop Client.</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
<tr>
<td>In the Silverlight theme, user preferences are saved in the stored settings and frozen columns for tables are supported</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
<tr>
<td>In the Silverlight theme, most decoration common style attributes described in the Genero Business Development Language User Guide are supported. Some decoration common style attributes not supported are:</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
<tr>
<td>- border</td>
<td></td>
</tr>
<tr>
<td>- localAccelerators</td>
<td></td>
</tr>
<tr>
<td>- imageCache</td>
<td></td>
</tr>
<tr>
<td>- showAcceleratorInToolTip</td>
<td></td>
</tr>
<tr>
<td>In the Silverlight theme, rich text editing is supported for TextEdit fields with textFormat html. An embedded toolbox with classic editing actions (bold, italic, underline, font size, etc.) is provided for text editing.</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
<tr>
<td>Text editing limitations compared to GDC / GWC for Ajax</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
<tr>
<td>- No list (ordered or unordered)</td>
<td></td>
</tr>
<tr>
<td>- No text indentation</td>
<td></td>
</tr>
<tr>
<td>- No local action binding</td>
<td></td>
</tr>
<tr>
<td>- xHTML is mandatory</td>
<td></td>
</tr>
<tr>
<td>- New lines in paragraph are replaced by spaces</td>
<td></td>
</tr>
<tr>
<td>The Silverlight theme provides a user-friendly file download window.</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
<tr>
<td>The Silverlight theme provides support for type ahead inputs.</td>
<td>The supporting topic for Silverlight has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide. For generic information about the Type Ahead input mechanism, see Type Ahead.</td>
</tr>
</tbody>
</table>
Table 43: 2.40 New Features: HTML5 theme

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to the HTML5 theme (preview version) for the Genero Web Client. The DUA_HTML5 output map is provided as a preview set. It is designed to run on modern desktop browsers as well as on mobile browsers.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the Genero Application Server 2.40 User Guide.</td>
</tr>
</tbody>
</table>

**Important:** The next major release of Genero will find the default theme changed to the HTML5 theme. The AJAX theme will be desupported in the subsequent release (the second major release after the current 2.40 release).

Genero Application Server 2.32 New Features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.32.

Table 44: Genero Application Server Version 2.32 New Features

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Genero Application Server administration tool gasadmin is provided. This tool is a command to list, ping and kill applications sessions.</td>
<td>See The gasadmin command on page 310.</td>
</tr>
<tr>
<td>MAX_REQUESTS_PER_DVM is a new entry for GWS pool configuration. It allows a limited number of requests to be processed before the DVM is stopped.</td>
<td>See MAX_REQUESTS_PER_DVM on page 385.</td>
</tr>
<tr>
<td>A new log output allows logging both in a directory and to the console.</td>
<td>See LOG on page 383.</td>
</tr>
</tbody>
</table>

Genero Application Server 2.30 New Features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.30.

Table 45: 2.30 New Features: New architecture

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAS 2.30 introduces dispatchers and proxies for more reliability, better performances and better integration in web servers.</td>
<td>See Overview and What is a dispatcher? on page 36</td>
</tr>
<tr>
<td>Java Application Server integration: The GAS server can be fully integrated into any J2EE Servlet container using the GWC dispatcher j2eedispatch.</td>
<td>The supporting topic has been removed, as dispatcher j2eedispatch is no longer used. Please see Java Servlet Installation and Web Server Configuration topics in the Genero Application Server 2.30 User Guide.</td>
</tr>
<tr>
<td>New monitoring information: The monitor has been revised to display information that is relevant to the new architecture. In addition to application-specific information, there are also statistics shown for the dispatchers and proxies.</td>
<td>See Monitoring on page 161.</td>
</tr>
</tbody>
</table>
New LOG system: The log format and categories have been adapted to the new architecture with log files created for each dispatcher, proxy, and DVM started.

See LOG on page 383.

The web services pool management has been enhanced to explicitly limit the number of DVMs that can be started for a specific Web service. The Timeout configuration has changed to prevent the GWS proxy and DVMs from running indefinitely if the GAS dispatcher or the web server crashes.

See Services Pool (GWS Only) on page 33

Table 46: 2.30 New Features: GWC themes

GWC for Silverlight is based on Microsoft™ Silverlight technology. It is a fully-implemented Genero front-end, like the Genero Desktop Client, with the ability to be customized, like GWC for AJAX.

The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.30 User Guide.

GWC for iPhone provides a dedicated Genero Web Client template and set of snippets to render Genero applications as close as possible to native iPhone applications.

The supporting topic has been removed, as the iPhone theme is deprecated. Please see the Genero Application Server 2.30 User Guide.

GWC for AJAX provides support for iPhone and iPad web browsers. With Safari on iPhone providing the interface for all web content on iPhone, most of the feature set of a desktop browser is made available to mobile users. When it comes to displaying GWC applications on the iPhone or iPod Touch, the AJAX mode is fully functional with Safari® on iPhone.

The supporting topic has been removed, as the AJAX theme is deprecated. Please see the Genero Application Server 2.30 User Guide.

Table 47: 2.30 New Features: Miscellaneous

GWC for AJAX and GWC for Silverlight support drag-and-drop.

The supporting topics have been removed, as the AJAX and Silverlight themes are deprecated. Please see the Genero Application Server 2.30 User Guide.

GAS provides a URI to launch GDC applications without having to configure a shortcut in the GDC monitor. Shortcuts can be exported as .gdc. GAS delivers these shortcuts through a URI for any application.

See URI Examples on page 57

Note:

• On Mac, you need to manually add the .gdc association for Safari® or save the .gdc and double click to launch the application.

• Currently, any access to /da/r url is handled by DUA_GDC map, no matter which OutputMap argument is set in the url.
### Overview

The ACCESS_CONTROL element specifies which IP addresses are allowed to execute an application or web service. By default, all IP addresses are allowed.

The new resource `configuration/filepath` provides the absolute path to the GAS configuration file.

New topics have been introduced to the documentation about each of the following themes (snippet sets):

- AJAX theme
- Silverlight theme
- Basic theme
- iPhone theme

New topics have been introduced to the documentation about ISAPI, FastCGI, and JAVA GAS dispatcher configuration, installation and integration to their dedicated Web servers. New topic added explaining GDC to GWC migration.

### Genero Application Server 2.22 New Features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.22.

**Table 48: 2.22 New Features**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications started by a <code>StartMenu</code> can be rendered in folder tabs.</td>
<td>See Enable the <code>StartMenu</code> and applications in folder tabs.</td>
</tr>
<tr>
<td>Textedit with <code>textFormat html</code> display a toolbox with classic editing actions (bold, italic, fontsize, and so on). Local actions are also created so you can create your own action views (global toolbar, etc).</td>
<td>See Rich text editing.</td>
</tr>
</tbody>
</table>

### Genero Application Server 2.21 New Features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.21.

**Table 49: 2.21 New Features: Backwards compatibility**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAS 2.21 is compatible with the Genero 2.11 product suite (FGL, GWS).</td>
<td>No additional reference.</td>
</tr>
</tbody>
</table>
### Table 50: 2.21 New Features: Genero Web Client New Features

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The layout of the AJAX output driver has been enhanced in several ways.</td>
<td>The supporting topic has been removed, as the AJAX theme is desupported. Please see the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>Picture deployment enhancements</td>
<td>The supporting topic has been removed, as the template path theme is desupported. Please see the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>• The list of file system paths searched for images can be configured according to the user agent type.</td>
<td></td>
</tr>
<tr>
<td>• Pictures that are not found on the GAS file system are requested from the DVM. The ResourceURI has been enhanced to support these requests.</td>
<td></td>
</tr>
<tr>
<td>Internet Explorer 8 is supported.</td>
<td>The supporting topic has been removed as GWC for HTML5v1 is deprecated. Please see the topic, Features / browsers supported, in the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>The authenticated user and the remote IP address from the client is accessible to the Genero application through environment variables.</td>
<td>See Application environment on page 43</td>
</tr>
<tr>
<td>The list of supported snippet-based rendering engine (SBRE) template paths is accessible through a specific URL.</td>
<td>The supporting topic has been removed, as the template path theme is desupported. Please see the topic, Template Paths Overview, in the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>The Template Element Identifiers feature provides a means to customize the template element identifier name used to reference elements during the incremental update of the page, according to the target markup language.</td>
<td>The supporting topic has been removed, as the template path theme is desupported. Please see the topic, Template Element Identifiers, in the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>The application/ui/decimalSeparator path gets access to the currently active decimal separator character.</td>
<td>The supporting topic has been removed, as the template path theme is desupported. Please see the topic, Template Paths - Application hierarchy, in the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>The SpinEdit/minValue and SpinEdit/maxValue paths get access to the corresponding SpinEdit attributes.</td>
<td>The supporting topic has been removed, as the template path theme is desupported. Please see the topics:</td>
</tr>
</tbody>
</table>
| Applications in configuration files support both a short description and a long description. | • SpinEdit/minValue  
• SpinEdit/maxValue  
See DEscription on page 366 |

### Genero Application Server 2.20 New Features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.20.

**Note:** Links to outdated documentation pages have been removed. If you are using version 2.20, you may wish to view the documentation created for version 2.20.
Table 51: 2.20 New Features: Application Server

Overview | Reference
---|---
Multi-threaded capabilities adapt the Genero Application Server daemon (gasd) capabilities to the Application Server demand. One gasd process will be able to adapt itself to the number of pending requests, providing better efficiency. | The architecture of the Genero Application Server has since changed. See Architecture of the Genero Application Server on page 29.

The following improvements have been made in GAS 2.20:

- New plug-able components (GAS Isapi and GAS FastCGI) are available for each targeted application server, providing a new connection architecture that adapts the external world model to the GAS model.
- A new generic resourceURI() function associated to the new PATH element definition allows you to specify multiple resource locations accessible from the browser on an application-level basis.

The GAS gives access to server and applications details through the “/monitor” URL. The monitor page exposes details about DVMs, Web applications, Web services, GDC applications, and GAS activity. | See Monitoring on page 161.

Table 52: 2.20 New Features: GDCAX management

Overview | Reference
---|---
Protocol between GDC and the application server is no longer character-oriented; it is now binary-oriented. As such, the connection is more efficient. They communicate on a unique URL, which improves the authenticated and secured connections. | The supporting topic has been removed, as the Genero Desktop Client ActiveX feature is deprecated. Please see the Genero Application Server 2.20 User Guide.

Table 53: 2.20 New Features: Web application management (Genero Web Client)

Overview | Reference
---|---
Tree views are fully supported by the Genero Web Client. | The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of what the HTML5 theme supports.

The new XUL snippet set offers a desktop-like rendering. The XUL snippet set will be activated if you use a Prism user agent. You can also activate XUL snippet sets in any Firefox® browser by using the DUA_XUL output map.

A new GAS 2.20 framework allows JavaScript functions to handle the BDL frontCall function on the browser.

BDL 2.20 also introduces two new frontcall functions (launchUrl and feinfo/screenresolution) which are fully supported by the GWC. | The supporting topic has been removed, as the XUL theme is deprecated. Please see the Genero Application Server 2.20 User Guide.

The new frontcall supporting topic has been removed, as the HTML5 theme is deprecated. Please see the JavaScript topic in the Genero Application Server 2.20 User Guide.
### Overview

Modern browsers fully support SVG and SWF. Image widget snippets create the correct HTML code so the browser will render images with the extension `.svg` or `.swf`.

Select multiple rows using the usual key and mouse combination to enhance your Display Array.

Enable Sorting in Input Array is possible.

Hyperlinks in Label / TextEdit are supported.

Button type `<link>`: A classic Genero BDL button, but displayed as an hyperlink.

### Reference

The supporting topic has been removed, as the HTML5v1 theme is deprecated. Please see the Genero Application Server 2.20 User Guide.

The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of what the HTML5 theme supports.

See Display a widget as a hyperlink

No additional reference.

---

### Upgrade Guides for the Genero Application Server

Each upgrade guide is an incremental upgrade guide that covers only topics related to a specific version of Genero. It is important that you read all of the upgrade guides that sit between your existing version and the desired version.

- GAS 3.00 upgrade guide on page 288
- GAS 2.50 upgrade guide on page 291
- GAS 2.41 upgrade guide on page 292
- GAS 2.40 upgrade guide on page 292
- GAS 2.30 upgrade guide on page 293
- GAS 2.22 upgrade guide on page 296
- GAS 2.21 upgrade guide on page 297
- GAS 2.20 upgrade guide on page 297
- Upgrading from GAS 2.10.x or GWC 2.10.x on page 298
- GAS (GWC) 2.10 upgrade guide on page 301
- GAS 2.00 upgrade guide on page 301

### GAS 3.00 (Maintenance Releases) upgrade guide

Complete these tasks when migrating to Genero Application Server 3.00 Maintenance Releases (MRs) of the GAS from version 3.00. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

#### Genero Report Viewer URLs

Starting with GAS version 3.00.12, the report viewer URL prefix `/ua/grv` that was provided to the Genero Report Engine (GRE) is no longer used.

New URL prefixes provide access to the HTML report viewer implementation and its applications’ public and private resources. Different URLs are provided for when the Genero Report Engine is operating in **local mode**, on the same server as the GAS, or in **distributed mode** on another server.
The URLs are very similar in structure in each case.

**Local /Local Server Mode**
- /ua/report/viewer
- /ua/report/public
- /ua/report/private/$(session.id)/reports

**Distributed Mode**
- /ua/report-r/viewer
- /ua/report-r/public
- /ua/report-r/private/$(session.id)/reports

These URLs are used for the same purpose in local mode as distributed mode, where:

- **viewer** loads the HTML report viewer implementation.
- **public** provides access to shared resources.
- **private** provides access to resources that are private to the application.

For more information see [Application URIs on page 51](#) and [REPORT_VIEWER_DIRECTORY on page 398](#).

**Debug mode**

Support of "gwc-js.debug" in a GAS configuration file has been dropped. If you have previously used `gwc-js.debug` to launch GBC applications in debug mode, you must now use an alternate method. See [GWC-JS application in debug mode on page 197](#).

**GAS 3.00 upgrade guide**

Complete these tasks when migrating to Genero Application Server 3.00 from version 2.50. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

**Client scope (protocols)**

The FGLGWS package includes both Genero Business Development Language and Genero Web Services.

**Table 54: FGLGWS 3.00 and client protocols on page 288** shows the protocols and front-ends that are supported with FGLGWS 3.00.

**Table 54: FGLGWS 3.00 and client protocols**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>ua</td>
<td>GDC 3.00, GWC-JS, GMA, GMI</td>
</tr>
<tr>
<td>ws</td>
<td>GWS</td>
</tr>
</tbody>
</table>

**Table 55: FGLGWS 2.50 and client protocols on page 288** shows the protocols and front-ends that are supported with FGLGWS 2.50.

**Table 55: FGLGWS 2.50 and client protocols**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>ja</td>
<td>GDC 2.50</td>
</tr>
<tr>
<td>wa</td>
<td>GWC-HTML5</td>
</tr>
</tbody>
</table>
**Important:** Web services (ws) with GAS 3.00 are supported with FGLGWS 3.00 DVM only.

**Important:** If you wish to use GWC-HTML5 or GDC 2.50 with GAS 3.00, you must work with FGLGWS DVM 2.50.

What does this mean for your upgrade to 3.00? Links that previously used "wa" should now use "ua" instead. See Application URIs on page 51 for more information on including the scope in a URI.

**IIS configuration no longer automatic**

The IIS automatic configuration has been removed from the installation process. Manual configuration is required.

**Genero Web Client**

Genero Web Client for JavaScript (GWC-JS) will be the default rendering for Genero Web Client applications, replacing Genero Web Client for HTML5 (GWC-HTML5). New development should use GWC-JS.

Document your existing customization under GWC-HTML5, and determine what needs to be reimplemented with GWC-JS.

**Genero Desktop Client ActiveX (GDCAX) is desupported**

For new development, use GWC-JS.

**Single Sign-On Authentication**

The Kerberos authentication mechanism for Single Sign-on authentication is deprecated. Any new development requiring Single Sign-on should plan to use OpenID Connect, SAML or OpenID. See How to implement Single sign-on (SSO) on page 122. For alternative solutions, please contact your Four Js support center.

**Java Dispatcher**

**Note:** The Java dispatcher is no longer used. Please see Java Servlet Installation and Web Server Configuration topics in the Genero Application Server 3.00 User Guide.

For Genero 3.0, the Java Dispatcher, requires at least version 3.0 of the Java servlet API. This is supported on the following Java web servers:

- Glassfish (since version 3.0). See [http://glassfish.java.net/](http://glassfish.java.net/)

**Web Server side Resources**

In GAS 2.50, using gwcproxy and html5proxy proxies, the path to the image directory is configured to fetch resources on the Web server side in the PATH element, see PATH with Type WEBSERVER on page 390.

```xml
<PICTURE>
  <PATH type="WEBSERVER">a_uri</PATH>
</PICTURE>
```

This is usually configured to improve performance. The Web server delivers static files or images instead of the GAS.
With GAS 3.00, for uaproxy, all the resources are delivered by the GAS. By default the application’s public directory is defined by the `PUBLIC_IMAGEPATH` in `FGLASDIR/etc/as.xcf` configuration file:

```xml
<PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
```

**Note:** `${res.public.resources}` defaults to the path `appdata/public/common`, see `PUBLIC_IMAGEPATH` on page 396.

If you use Genero Archive you can specify public resources for your applications by adding a `RESOURCES` element in the Genero Archive manifest. Then the resources are copied in `FGLASDIR/appdata/public/deployment/deployappname`, where "deployappname" is the name given to the deployed application directory by the Genero Archive. See `Resource deployment overview` on page 257.

**Web Components**

In GAS 2.50, web components are deployed under `$FGLASDIR/web/components` directory, see `WEB_COMPONENT_DIRECTORY` on page 422.

Starting from GAS 3.00, with uaproxy, the default path for a web component is `appdir/webcomponents`, where "appdir" is the application directory. See the `WEB_COMPONENT_DIRECTORY` element in your `FGLASDIR/etc/as.xcf` configuration file:

```xml
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
    ...
    <DVM>${res.dvm.wa}</DVM>
    <WEB_COMPONENT_DIRECTORY>${application.path}/webcomponents</WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

You can change the default web components location by configuring a `WEB_COMPONENT_DIRECTORY` element in your application's configuration. In this example, the web component is no longer located in `appdir/webcomponents` but in `appdir/mycomponents`.

```xml
<APPLICATION Parent="defaultgwc" ...>
    <EXECUTION>
        <PATH>/home/myapp</PATH>
        <MODULE>myapp</MODULE>
        <WEB_COMPONENT_DIRECTORY>/home/myapp/mycomponents</WEB_COMPONENT_DIRECTORY>
    </EXECUTION>
</APPLICATION>
```

For more details on Web component usage, see the *Genero Business Development Language User Guide*.

**Genero Web Client hybrid mode (deprecated)**

The GWC hybrid mode feature has been deprecated. Applications for Android™ or iOS mobile platforms which used the GWC hybrid mode will need to use the more-featured GMA or GMI instead. If you do not have equivalent features in GM, contact your local Four Js support center.

**GWC-JS LOOKUP PATH**

The `GWC_JS_LOOKUP_PATH` on page 376 is a new mandatory element in `as.xcf`. If you are upgrading to GAS versions 3.00.11 or later and wish to keep your existing `as.xcf` configuration file, you need to add this element to the `INTERFACE_TO_CONNECTOR` on page 381 element. Otherwise, you will see the following error message at GAS startup:

```
Application Server startup . ...........................................
httpdispatch
```
The following is a diagnostic message indicating a configuration error:

"Configuration ERROR" Code:1871 Message:Element 'TEMPORARY_DIRECTORY': This element is not expected. Expected is ( GWC_JS_LOOKUP_PATH ).

The error message is a reference to a configuration file issue. The expected element is not found, which is causing a build failure for the main configuration.

To find your GAS version, at the command line run:

gasadmin -V

For more information on the gasadmin tool, see The gasadmin command on page 310 topic.

**GAS 2.50 upgrade guide**

Complete these tasks when migrating to Genero Application Server 2.50 from version 2.40. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

- **GWC for HTML5** will be the default rendering for Genero Web Client applications, replacing GWC for AJAX. Any new development should plan to use GWC for HTML5.
- Customization using CSS has changed to provide a flat design to fit all platforms (desktop, tablet, and smartphone) using the HTML5 theme. All CSS customization is contained within `css_customization.css`. This should ease both customization and migration between versions of the Genero Application Server. As the HTML5 theme is deprecated, please see the Cascading Style Sheets and HTML5 theme topics in the Genero Application Server 2.50 User Guide.
- The **Genero Administration Application (GAD)** is no longer supported.
- The `ForwardDVMStdout` attribute for the `MAP` element is no longer permitted in the configuration file. The DVM output and errors are logged to a separate log file. Remove this attribute from your Genero Application Server configuration file.
- The `CONNECTOR_PREFIX` element is no longer valid in the Genero Application Server configuration file. Remove it from your configuration file, if present.
- The behavior of a simple `RUN` command has changed. Previously, when an child application was launched using a simple `RUN` command, the child application replaced the main application in the same browser window. Only the child application was visible. When you exited the child application, the main application resumed. Throughout, there was a single browser window.

With version 2.50, an application launched by a simple `RUN` command opens a second window (or browser) containing the child application, as it also does for `RUN WITHOUT WAITING`. The child application runs normally. The main application is frozen, waiting for the child application to terminate. If you attempt to leave or close the main application when a child application is still running, a prompt asks you to confirm that you want to leave.

- Genero Front Call ActiveX is deprecated.
- Logging for the Genero Application Server has changed. Logs are created for the dispatchers, the proxies, and the virtual machines. DVM logs are redirected to files when `DAILYFILE` is set for the log output type.
- The default location of the Genero Application Server log files has moved. If you set logrotate or another tool to manage your GAS log files, you will need to reset. See GAS directories on page 39 for details on the location of log files.
- Some Windows® directories have restricted permissions. Depending on where the Genero Application Server is installed, you might not have access to log, tmp and sessions directories. These directories are located at `C:\ProgramData\FourJs\gas\gas_version` where `gas_version` is the version of the Genero Application Server. You can configure their location by modifying "res.appdata.path" in the Genero Application Server configuration file.
**GAS 2.41 upgrade guide**

There are no migration tasks specific for the Genero Application Server 2.41 release.

While there are no migration tasks specific for the Genero Application Server 2.41 release, you should be aware that starting with the next major release (tentatively 2.50):

- GWC for HTML5 will be the default rendering for Genero Web Client applications, replacing GWC for AJAX. Any new development should plan to use GWC for HTML5.
- The Genero Administration Application (GAD) will be de-supported.

**GAS 2.40 upgrade guide**

Complete these tasks when migrating to Genero Application Server 2.40 from version 2.30. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

- **Template and snippets** on page 292
- **Template paths** on page 293

**Template and snippets**

**Namespaces**

To optimize template and snippets rendering, all namespaces need to be declared in the main template. You need to move all the namespaces used in your snippets to the main template. The changes have to be applied on all snippet sets.

Example in 2.3x:

```
$FGLASDIR/tpl/SetAjax/main.xhtml
```

```html
<html xmlns:gwc="http://www.4js.com/GWC" xmlns="http://www.w3.org/1999/xhtml"
     xmlns:svg="http://www.w3.org/2000/svg"
     viewBox="0 0 1000 1000"
     preserveAspectRatio="none">
    <svg:rect x="0" y="0" width="1000" height="1000" fill="lightgray"
             stroke="black"/>
    <span gwc:repeat="i items" gwc:replace="i" />
</svg:svg>
```

Example in 2.40:

```
$FGLASDIR/tpl/SetAjax/main.xhtml
```

```html
<html xmlns:gwc="http://www.4js.com/GWC" xmlns="http://www.w3.org/1999/xhtml"
     xmlns:svg="http://www.w3.org/2000/svg" ...
```

**GWC for Silverlight behaviors**

Starting with GAS 2.40, GWC for Silverlight follows the Model-View-ViewModel (MVVM) pattern to enable even more designers to customize the view of their applications with tools like Microsoft® Expression Blend.
This change leads to the replacement of the GWC.Behaviors module by a view models layer. Therefore all the XAML markup that used the bhv prefix needs to be updated. Here is an example of the migration of an action view.

In 2.32:

```
<sr:Button
  bhv:Action.Observer="{action & action/isActive ?
  [action/IDID, 'Activate'] : null}"
  bhv:Media.URI="[ '{resourceURI("images/close.png", "SetSL")}', 'SmallImage']"
  IsEnabled="{action/isActive || false}"
  IsTabStop="False" />
```

In 2.40:

```
<sr:Button
  Command="{{Binding ClickCommand,Mode=OneTime}"
  SmallImage="{{Binding Image.Source}"
  IsEnabled="{action/isActive || false}"
  IsTabStop="False">
  <sr:Button.DataContext>
    <vm:ActionView Image="{resourceURI('images/close.png','SetSL')}">
      <vm:ActionView.ServerActions
        gwc:condition="action && action/isActive">
        <vmsa:Action ServerID="{action/IDID}" IsEnabled="True" Event="Click" />
      </vm:ActionView.ServerActions>
    </vm:ActionView>
  </sr:Button.DataContext>
</sr:Button>
```

The bhv:Action.Observer and bhv:Media.URI behaviors have been replaced by a more conventional mechanism based on Data bindings that gets their properties from the ActionView view model.

**GWC for Silverlight template snippet splitting**

To ease the customization, some template snippets have been splitted into more parts:

- The user interface part of the main.xaml template has been put into the UserInterface.xaml snippet.
- The ending part of the main.xaml template has been put into the EndingPage.xaml snippet.
- The toolbar and the top menu parts of the WindowContent.xaml snippet have been put into the ToolMenu.xaml snippet.

**Template paths**

application/interrupt/did and application/interrupt/xdid template paths are replaced by application/interrupt/url. For more details on the new template path usage see the main templates.

**GAS 2.30 upgrade guide**

Complete these tasks when migrating to Genero Application Server 2.30 from version 2.22. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

- Starting the GAS on page 294
- Configuration on page 294
- Support on page 295
- Template functions on page 296
- XPathConfig migration on page 296
- Legacy connectors on page 296
Starting the GAS

The gasd command no longer exists, it is replaced by dispatchers. To run the GAS in standalone mode use `httpdispatch` command. On windows, from the start menu, there is a shortcut to start the dispatcher in standalone mode.

Configuration

Starting with version 2.30, the configuration file has been simplified. Some configuration parameters are no longer needed due to the new architecture, and others are now handled transparently (such as the web services pool and load).

--development

To start a GWC application in development mode, you must add the option `--development` to the `gwcproxy` entry in the appropriate xcf file.

Example:

```xml
<RESOURCE Id="res.gwcproxy.param" Source="INTERNAL">--development</RESOURCE>
```

This resource `res.gwcproxy.param` is used as parameter when launching the GWC proxy

```xml
<RESOURCE Id="res.gwcproxy" Source="INTERNAL">
  "$\{res.path.as\}/bin/gwcproxy\" $\{res.gwcproxy.param\}
</RESOURCE>
```

Configuration file (xcf) inheritance

Prior to version 2.30, when an application was started with a RUN, the child application inherited the configuration of the parent unless the child application has its own configuration file. If the child application had its own configuration file (where the .xcf file shares the same name as the child application), then the child configuration took priority over the parent configuration, and was used for the child application.

Starting with version 2.30, this is no longer true. A 4gl application started with RUN or RUN WITHOUT WAITING will inherit the configuration of the parent application, and this cannot be changed. The configuration used to start the first application (the first FGLRUN) will be used for all child applications (child FGLRUNs).

While you might want to review your parent/child applications, this will likely not have an impact, as a customer survey determined that most had only provided a configuration file for the parent application and had not provided a configuration file for the child applications.

Resource of type FILE

Starting with version 2.30, resources of type FILE are no longer supported.

Example:

```xml
<RESOURCE Id="res.theme.default.gdc.template" Source="FILE">
  "$\{res.path.tpl\}/fglgdcdefault.html"
</RESOURCE>
```

ALIAS entry removed

Starting with version 2.30, the entry ALIAS has been removed. You can configure PICTURE element instead.
THREAD_POOL entry removed
Starting with version 2.30, the entry THREAD_POOL has been removed.

LOG output of type PATH de-supported
Starting with version 2.30, you cannot set a log output of type PATH.

REQUEST_QUEUE and REQUEST_RESULT for Web Services are removed
Starting with version 2.30, Web Service REQUEST_QUEUE and REQUEST_RESULT have been removed.
Instead of these timeouts, the web server one is used. For example, the fastcgi idle timeout.

DVM_FREE removed
Starting with version 2.30, Web Service DVM_FREE has been removed.
It is no more needed as the Web Service pool management uses statistics of previous requests to decide whether to stop a DVM or not.

Hot Restart no longer necessary
There is no longer a need for a hot restart, as changes in an external XCF file are immediately taken into account at:
• Application startup for a web application
• GWS DVM startup for web services that could be at each new request
As a result, you should check the changes you have made to your XCF files and ensure they are correct before you save the file. For example, you could create a test.xcf file and validate that the test.xcf file is correct; then replace the production xcf file (assume the file is named prod.xcf for this discussion) by archiving prod.xcf and renaming test.xcf to prod.xcf.

Socket port selection
The following three entries of the INTERFACE_TO_DVM configuration are deprecated and will be removed in the next release.
The selection of a free socket port will be in charge of the operating system for performance issues.
• TCP_BASE_PORT
• TCP_PORT_RANGE
• EXCLUDED_PORT

License Consumption and Web Service Applications
Prior to this release, when it came to consuming licenses, you were able to go over the setting of MAX_AVAILABLE, up to the limit specified for the application by the MaxLicenseConsumption attribute. This is no longer the case. MaxLicenseConsumption is no longer available as an application attribute and will be ignored by the Genero Application Server. The maximum limit for licenses is given by MAX_AVAILABLE. You will not be allowed to go over this limit.

Support

XUL snippets set is no longer supported
Starting with version 2.30, the XUL snippet set is no longer supported. Instead, there is GWC for Silverlight, which covers the same kind of usage as the XUL snippet set, but with a more powerful technology.
Following OS are no longer supported: osf0510, sco0507 and uxw0711

These Operating Systems (HP Tru64 unix V5.1B, SCO OpenServer 5.0.7 and SCO UnixWare 7.1.3 - 7.1.4 & OpenServer 6.00) are very old and do not fit with the minimal OS requirements needed for the GAS redesign (specifically multi-threading).

Template functions

XPathConfig migration

Starting with version 2.30, the configuration provided in XML format to the GWC has changed; you have to adapt all of your customized snippet files containing XPathConfig expressions. Only the XML configuration needed by the GWC is loaded, so all the XPathConfig expressions must be simplified as follows:

- **Remove APPLICATION node**

  ```
  XPathConfig('/APPLICATION/TIMEOUT/USER_AGENT/text()')
  ```

  to

  ```
  XPathConfig('/OUTPUT/MAP/TIMEOUT/USER_AGENT/text()')
  ```

- **Remove DUA Id attribute**

  ```
  XPathConfig('APPLICATION/OUTPUT/MAP[@Id='DUA_AJAX']/RENDERING/MIME_TYPE/text()')
  ```

  to

  ```
  XPathConfig('/OUTPUT/MAP/RENDERING/MIME_TYPE/text()')
  ```

It is no longer possible to access all XPATH with function XPathConfig(). We can only access the node OUTPUT and its descendants.

Legacy connectors

Starting with version 2.30, due to the new multi-threaded architecture, legacy connectors are no longer provided in the package.

Mod_fcgid

Mod_fcgid is no more supported. Mod_fcgid architecture did not fit GAS 2.2x stateful process. Mod_fcgid has not been reconsidered in GAS 2.30.

GAS 2.22 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.22 from version 2.21. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Template path application/meta/variable

`application/meta/variable[name]` no longer returns the value of the variable.

To get the variable value, use the new path: `application/meta/variable[name]/value`.

If the new path is not used, you will get an error such as this:

```
Rendering error...
Template snippet: _default, style '_default', line 107
Message: The 'variable' path element cannot be rendered automatically; please use its attributes
```
GAS 2.21 upgrade guide
Complete these tasks when migrating to Genero Application Server 2.21 from version 2.20. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Topics
• Configuration

Configuration
• Picture
  For web applications, the picture configuration has moved from APPLICATION/PICTURE to APPLICATION/OUTPUT/MAP/PICTURE.

See PICTURE on page 392 configuration for more details.

GAS 2.20 upgrade guide
Complete these tasks when migrating to Genero Application Server 2.20 from version 2.10.x. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

• Configuration on page 297
• Snippets sets on page 298
• Legacy connectors installation on page 298
• Templates and snippets on page 298

Configuration

Timeout
For web applications, the timeout configuration has moved from APPLICATION/TIMEOUT to APPLICATION/OUTPUT/MAP/TIMEOUT.

See TIMEOUT configuration for more details.

Session Variables and Cookies
The cookie configuration has moved from APPLICATION/OUTPUT/MAP to APPLICATION/OUTPUT.

See HTTP_COOKIES configuration for more details.

Blob URLs
If you are using blob urls (Note: for more information please see the Template Paths - Document hierarchy topic in the Genero Application Server 2.20 User Guide), beginning with 2.20, you need to allow the access to the resources, by default the access is disabled. See WEB_APPLICATION_EXECUTION_COMPONENT on page 417 for more details.

DVM ping timeout
Since 2.20.09, the DVM ping timeout is configurable. DVM_PINGTIMEOUT

Error document
INTERFACE_TO_CONNECTOR/ERROR_DOCUMENT is no longer available. This should now be configured and handled by the Web server.
Snippets sets

The snippets sets are renamed with more explicit names. set1 is renamed into setAjax. set2 and set3 are merged and becomes setBasic. A new set, setXul, for output map DUA_XUL is added.

These changes imply some migrations for your customized snippets.

• setAjax (formely known as set1)
  Adapt main.xhtml to reflect the last enhancement on resource deployment.

• setBasic (formely known as set2 and set3)
  As setBasic is the result of the merging of set2 and set3, it is recommended that you rework your customization from this new set.

Legacy connectors installation

Such an architecture is no longer recommended. It is provided to ease the migration to 2.20. The installation of the connector assumes that you have the rights to install the product in the web server directories.

To install the legacy connectors:

• On IIS, you have to create a virtual directory named "gas" for example and assign execution permission to this virtual directory
• On Apache, configure your own ScriptAlias named "gas" for example or use the default ScriptAlias directory "/cgi-bin/".
• Then copy the content of $FGLASDIR/legacy_connectors to the directory.

Validate the installation by accessing demo program with a URL like:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(with IIS)</td>
<td><a href="http://myWebServer/gas/fglcisapi.dll/ua/r/gwc-demo">http://myWebServer/gas/fglcisapi.dll/ua/r/gwc-demo</a></td>
</tr>
<tr>
<td>(with Apache)</td>
<td><a href="http://myWebServer/gas/fglccgi.exe/ua/r/gwc-demo">http://myWebServer/gas/fglccgi.exe/ua/r/gwc-demo</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://myWebServer/gas/fglccgi/ua/r/gwc-demo">http://myWebServer/gas/fglccgi/ua/r/gwc-demo</a></td>
</tr>
</tbody>
</table>

If you encounter any issues or need to configure connector.xcf, please refer to GAS manual prior to 2.20.

Templates and snippets

• double the left curly brace "{{" to escape the embedded expression processing.

Upgrading from GAS 2.10.x or GWC 2.10.x

Complete these tasks when migrating from Genero Application Server 2.10.x to a later version.

• Application configuration on page 298
• Template and snippets on page 300
• Deprecated functions and paths on page 300

Application configuration

Add noNamespaceSchemaLocation attribute in external application configuration file

All external application configuration files must be updated by adding the noNamespaceSchemaLocation attribute. When defining external application files, the "noNamespaceSchemaLocation" attribute should have this value:

• For web applications: xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.xsd"
• For web services applications: xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextws.xsd"

For example, consider the following examples of the Edit.xcf web application configuration file:
The old Edit.xcf:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
  </EXECUTION>
</APPLICATION>
```

The new Edit.xcf:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
  </EXECUTION>
</APPLICATION>
```

If this attribute is missing, the corresponding application will fail to start, and the following message will be written to the log file:

```
Can't find 'noNamespaceSchemaLocation' attribute in external application file '/home/f4gl/gwc/app/Edit.xcf'.
```

Output drivers for Internet Explorer

Specific output drivers DUA_AJAX_HTML and DUA_PAGE_HTML have been added to support certain features (such as the Canvas widget) on Internet Explorer. As a result, all customized snippets specified for DUA_AJAX will also need to be specified for DUA_AJAX_HTML.

The original CardStep1.xcf:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_AJAX">
      <THEME>
        <SNIPPET Id="Image" Style="Picture">
          $(res.path.demo.app)/card/tpl/set1/Image.xhtml
        </SNIPPET>
      </THEME>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

The new CardStep1.xcf:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_AJAX">
      <THEME>
        <SNIPPET Id="Image" Style="Picture">
          $(res.path.demo.app)/card/tpl/set1/Image.xhtml
        </SNIPPET>
      </THEME>
    </MAP>
  </OUTPUT>
</APPLICATION>
```
Likewise, all customized snippets specified for DUA_PAGE will also need to be specified for DUA_PAGE_HTML.

To change output drivers default behaviors, see Automatic Discovery of User Agent.

**URL parameters**

By default, parameters in the URL are not taken into account. They are not transmitted to the DVM. Only the parameters defined in the configuration files are transmitted.

To use URL parameters, in the EXECUTION tag, you have to set AllowUrlParameters to TRUE.

Caution, parameters are transmitted to the DVM in this order: configured parameters in PARAMETERS on page 389 tag followed by the URL parameters.

**Template and snippets**

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

**Deprecated functions and paths**

Some functions have been renamed due to enhancements on the front-end protocol. The default logging includes the DEPRECATED category that displays warnings. If any deprecated functions are used, this kind of warning is logged:

```
[ TASK=1808 VM=1860 WA=115128484 TEMPLATE ]
Event(Time='7.481526', Type='Using deprecated function')
/ function(Name='makescrollpagesizedid')
```

**Deprecated template paths**

- DID becomes IDID
Deprecated template functions
- makeCompoundRowSelectionDID becomes makeRowSelectionIDID
- makeScrollPageSizeDID becomes makePageSizeIDID
- makeScrollOffsetDID becomes makeScrollOffsetIDID

GAS (GWC) 2.10 upgrade guide
If you have been working with the Genero Web Client prior to the release of GWC 2.10, you will have already done some configuration and possibly customization to deliver your Genero applications as Web applications using the initial built-in rendering engine (no longer supported).

All new development and advances is focused on the snippet-based rendering engine introduced with GWC 2.10.
- Migrating to the snippet-based rendering engine on page 301

Migrating to the snippet-based rendering engine
To take full advantage of the snippet-based rendering engine, you must follow the procedures outlined in this manual, regardless of whether or not you have previously deployed your application using the pre-2.10 GWC.

If you have previously deployed the application with the pre-2.10 GWC, ensure you revisit the following:
- Customization includes presentation styles and snippet sets; previous customization of the template files are no longer valid with the snippet-based rendering engine. Customization previously implemented using JavaScript will likely be implemented using HTML.
- The limitations of the GWC prior to 2.10 included lack of accelerator key support, StartMenus, ProgressBars, ON IDLE, StatusBars, and Genero Presentation Styles. You may have modified your application to work around these limitations. (Please see the topic, Features and limitations, in the Genero Application Server 2.10 User Guide.)

Note: Links to pages made obsolete by more current releases have been removed from this page.
If you are migrating to GWC 2.10, we recommend you use the documentation released with version 2.10.

GAS 2.00 upgrade guide
Complete these tasks when migrating to Genero Application Server 2.00 from any earlier version of the Genero Application Server.
- fglxslp migration tool on page 301
- fglxmlp XML preprocessor on page 302

fglxslp migration tool
Starting with Genero 3.00, the fglxslp tool is deprecated.

When migrating from Genero Application Server (GAS) 1.3x to 2.00, it is necessary to update your GAS configuration file to conform to the XML specifications of GAS 2.00. A migration tool, fglxslp, has been added to assist you in this migration.

Usage:

```
$FGLASDIR/bin/fglxslp $FGLASDIR/etc/gasxcf1xxto200.xsl
$FGLASDIR/etc/as-132.xcf > $FGLASDIR/etc/myas.xcf
```

Note:
- fglxslp is the migration tool.
- gasxcf1xxto200.xsl is the XSL style sheet that describes the GAS 2.00 XML configuration file
- as-132.xcf is the configuration file to migrate (GAS 1.3x).
- myas.xcf is the result (new configuration file for GAS 2.00).
**fglxmlp XML preprocessor**

The XML Preprocessor can be used as part of the BDL development process. It fetches data in a XML resource file to "fill" the content of a source file that contains the dollar tag expression.

**Usage:**

```bash
$FGLASDIR/bin/fglxmlp -i src1.4gx -o src1.4gl -r resource.xrf
```

**Note:**

- *src1.4gx* is the file to be processed through the XML Preprocessor.
- *src1.4gl* is the output file.
- *resource.xrf* is the XML resource file containing the definition of a complex 4GL record.

**Using the XML Preprocessor**

In this example, two source files will be "expanded" through the XML resource file. The resource file contains the definition of a complex 4GL record. The extension of files to be processed through the XML Preprocessor is .4gx. The extension for the resource file is .xrf (XML Resource File).

```bash
fglxmlp -i src1.4gx -o src1.4gl -r resource.xrf

fglxmlp -i src2.4gx -o src2.4gl -r resource.xrf
```

The resulting .4gl files are compiled and link as usual:

```bash
fglcomp -c src1.4gl

fglcomp -c src2.4gl

fgllink -o project.42r src1.42m src2.42m
```

**Files used in the example**

**src1.4gx**:

```plaintext
01 FUNCTION useRecord (myRecord)
02 DEFINE myRecord $(record)
...
06 END FUNCTION
```

**resource.xrf**:

```plaintext
01 <?xml version="1.0" ?>
02 03 <RESOURCE_FILE>
04 <RESOURCE_LIST>
05 <RESOURCE Name="record"><![CDATA[
06     RECORD
07         nb_columns INTEGER,
08         nb_lines INTEGER,
09         name CHAR (8)
10     END RECORD
11 ]]></RESOURCE>
12 03 </RESOURCE_LIST>
13 </RESOURCE_FILE>
```
The output file src1.4gl:

```
01 FUNCTION useRecord (myRecord)
02    DEFINE myRecord
03        RECORD
04            nb_columns INTEGER
05            nb_lines INTEGER,
06            name CHAR (8)
07        END RECORD
...
15 END FUNCTION
```

### Migrating Templates and Snippets Customizations

Starting with Genero 3.00, templates and snippets are deprecated

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the `wa` protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use Genero Browser Client, see *Genero Web Client for JavaScript (GWC-JS)* on page 184.
Reference

Genero Application Server reference.

- **Tools and Commands** on page 304
- **Automatic discovery of User Agent (adua.xrd)** on page 335
- **GAS Predefined resources** on page 340
- **GAS Configuration Reference** on page 341

Tools and Commands

Information about the dispatchers, proxies, command line utilities, and the **Ghost Client testing tool**.

GAS Dispatchers

GAS dispatchers refer to the connectors in charge of dispatching a GAS request to the appropriate proxy.

- `httpdispatch` (Standalone)
- `fastcgidispatch` (Web Server)
- `java-j2eedispatch` (Java Servlet)
- `isapidispatch`, delivered as a DLL; dedicated to IIS (Windows® platforms)

Proxy

Proxies refer to binaries started by a dispatcher to serve a type of request.

- `uaproxy` (Genero Web Client for JavaScript (GWC-JS), Genero Desktop Client)
- `gwsproxy` (GWS)

Utility

Binaries delivered to ease XML files processing, such as configuration migration.

- `fglgar`
- `gasadmin`
- `Dispatcher: httpdispatch` on page 304
- `Dispatcher: fastcgidispatch` on page 306
- `Dispatcher: java-j2eedispatch` on page 307
- `Proxy: uaproxy` on page 307
- `Proxy: gwsproxy` on page 308
- `Proxy: html5proxy` on page 308
- `The fglxsip command` on page 308
- `The fglgar command` on page 308
- `The gasadmin command` on page 310
- `gas_ghost_client.ditamap`

Dispatcher: `httpdispatch`

`httpdispatch` is the standalone dispatcher that starts the Genero Application Server (GAS) in command line. No web server is needed. It is only used in development mode.

**Important:** The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.
Note: The GAS is configured through the GAS configuration file. This configuration file can be the default configuration file ($FGLASDIR/etc/as.xcf) or a custom configuration file that is specified when the Genero Application Server is started. For more information on configuration parameters, see GAS configuration file on page 342.

Syntax

```
httpdispatch [options]
```

Options

Table 56: httpdispatch options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h --help</td>
<td>Displays help information.</td>
</tr>
<tr>
<td>-p directory --as-directory directory</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>-f configuration_file --configuration-file configuration_file</td>
<td>Specify which configuration file to use when starting the Genero Application Server dispatcher. If not specified, the default configuration file, $FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>-k --no-keepalive</td>
<td>Disable keep alive for http connections. For debug purpose only.</td>
</tr>
<tr>
<td>-E name=value --resource-overwrite name=value</td>
<td>Overwrites the resource defined in the configuration file or creates a new one. Example:</td>
</tr>
<tr>
<td></td>
<td>httpdispatch -E res.dvm.wa=$FGLDIR/bin/myrun</td>
</tr>
<tr>
<td></td>
<td>If in the configuration file &quot;res.dvm.wa&quot; has another value it is now set to myrun. The final value is the one set in the option.</td>
</tr>
<tr>
<td>-V --version</td>
<td>Displays version information.</td>
</tr>
</tbody>
</table>

What does "address already in use" mean?

The message "address already in use" means that an application server (dispatcher) has already been started on the same port. Check the GAS configuration file (default $FGLASDIR/etc/as.xcf) to identify the port where the application server (dispatcher) started. The port number is identified in the following section:

```
<INTERFACE_TO_CONNECTOR>
<TCP_BASE_PORT>6300</TCP_BASE_PORT>
<TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
```
The default port specified is 6394 - derived by adding the base port (6300) to the port offset (94). Set the values to a port which is not used by another application, see INTERFACE_TO_CONNECTOR on page 381.

**Restarting the standalone GAS**

To restart httpdispatcher, use:

```
kill -9
```

Once the Web server restarts the dispatcher, the dispatcher uses the session table to reconnect to the various proxies. The applications are still maintained by proxies, are still running, and once the dispatcher is relaunched, the user can continue his or her work.

Ctrl + C or sending SIGTERM will stop the standalone dispatcher, and in both cases the dispatcher will request all proxies to stop. With `kill -9` the dispatcher process is killed yet the sessions remain alive and untouched. When the dispatcher is restarted, the sessions continue to be active. Notice, the fastcgi dispatcher will stop sessions on Ctrl-C too if started in standalone mode. But not on SIGTERM.

**Dispatcher: fastcgidispatch**

`fastcgidispatch` is the dispatcher for web server supporting FastCgi protocol.

It can be started by the Web server or in command line. See FastCGI Installation and Web Server Configuration on page 89 for more details.

**Syntax**

```
fastcgidispatch _options_
```

**Options**

**Table 57: fastcgidispatch options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p directory</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--as-directory directory</td>
<td></td>
</tr>
<tr>
<td>-f configuration_file</td>
<td>Specify which configuration file to use when starting the Genero Application Server dispatcher. If not specified, the default configuration file, $FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>--configuration-file configuration_file</td>
<td></td>
</tr>
<tr>
<td>-E name=value</td>
<td>Overwrites the resource defined in the configuration file or creates a new one.</td>
</tr>
<tr>
<td>--resource-overwrite name=value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>fastcgidispatch</td>
</tr>
<tr>
<td></td>
<td>-E res.dvm.wa=$FGLDIR/bin/myrun</td>
</tr>
<tr>
<td></td>
<td>If in the configuration file &quot;res.dvm.wa&quot; has another value it is now set to myrun. The final value is the one set in the option.</td>
</tr>
</tbody>
</table>
### Option Description

- **-s**
  --standalone
  Start the fastcgi dispatcher in standalone mode, where the dispatcher is started prior to any actual request from an application or web server. The web server's FastCGI extension must be configured to connect to an external GAS.

- **-h**
  --help
  Displays help information.

- **-v**
  --version
  Displays version information.

For information on restarting the fastcgidispatch, see Restarting the FastCGI dispatcher on page 96.

### Dispatcher: java-j2eedispatch

The java-j2eedispatch is a Java servlet that manages the Genero Application Server in a Java Web container, such as Tomcat™ or JBoss.

The Genero Application Server provides a Java servlet dispatcher called java-j2eedispatch by default, which can be deployed on any Java server supporting servlets. It offers the same functionality as Dispatcher: httpdispatch on page 304 and Dispatcher: fastcgidispatch on page 306 but has the advantage of opening the Genero technology to the Java web servers such as Apache Tomcat®, WebSphere®, GlassFish or JBoss.

The java-j2eedispatch dispatcher requires at least version 3.0 of the Java servlet API. This is supported on the following Java web servers:

- Glassfish (since version 3.0). See [http://glassfish.java.net/](http://glassfish.java.net/)

### Proxy: uaproxy

The uaproxy manages a Genero Web Client for JavaScript (GWC-JS) or Genero Desktop Client (GDC) application session.

It must be started by a dispatcher. If you attempt to run the uaproxy from the command line, you will likely get an error. The proxies are not intended to be run from the command line, but started by a dispatcher in the proper environment.

### Syntax

```
uaproxy [options]
```

### Options

**Table 58: uaproxy options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Display help.</td>
</tr>
<tr>
<td>-V, --version</td>
<td>Show version.</td>
</tr>
</tbody>
</table>
### Proxy: gwsproxy

The gwsproxy manages all services for a single Genero web application.

The `gwsproxy` must be started by a dispatcher.

If you run the `gwsproxy` from the command line, you will get the message “ERROR: FGL_VMPROXY_SESSION_ID not set.”

### Proxy: html5proxy

Starting with Genero 3.00, the `html5proxy` proxy is deprecated.

It must be started by a dispatcher. If you attempt to run the `html5proxy` from the command line, you will likely get an error. The proxies are not intended to be run from the command line, but started by a dispatcher in the proper environment.

### Syntax

```
html5proxy [options]
```

### Options

**Table 59: html5proxy options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--development</code></td>
<td>Enable the development mode that gives access to the application AUI tree and logs. See Configure development environment on page 156</td>
</tr>
<tr>
<td><code>--dump-io</code></td>
<td>Enable scenario dump. Log internal information for debugging purpose. Do not set it unless directed by your support center.</td>
</tr>
</tbody>
</table>

### The fglxslp command

Starting with Genero 3.00, the `fglxslp` tool is deprecated.

The `fglxslp` command is the same tool as the `xsltproc` tool (a command line tool for applying XSLT stylesheets to XML documents), mainly used to have a XSLT processor on every system. For more details on the `fglxslp` command, enter `fglxslp -h` from the command line.

**Important:** This utility is not supported.

### The fglgar command

The Genero Archiver tool (`fglgar`) enables you to build a java Web Archive (`.war`) to deploy Genero Application Server (GAS) on Java Web servers or to build a Genero archive (`.gar`) to deploy Genero applications on GAS.

It configures the Java™ Servlet to use the GAS directory and GAS configuration file. It also provides a deployment framework for managing application resources and deploying public images on the GAS.
**Syntax**

```
fglgar [options]
```

**Options**

**Table 60: fglgar options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-V</td>
<td>Displays version information.</td>
</tr>
<tr>
<td>--version</td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>Displays help information.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>-o</td>
<td>Name of archive file to create.</td>
</tr>
<tr>
<td>--output</td>
<td></td>
</tr>
<tr>
<td>-v</td>
<td>Displays the verbose output of additional information.</td>
</tr>
<tr>
<td>--verbose</td>
<td></td>
</tr>
<tr>
<td>-q</td>
<td>Silent mode</td>
</tr>
<tr>
<td>--quiet</td>
<td></td>
</tr>
<tr>
<td>-d</td>
<td>Directory where to save the archive file.</td>
</tr>
<tr>
<td>--output-directory</td>
<td></td>
</tr>
<tr>
<td>-s directory</td>
<td>Directory to archive.</td>
</tr>
<tr>
<td>--input-source</td>
<td></td>
</tr>
<tr>
<td>directory</td>
<td></td>
</tr>
<tr>
<td>-w</td>
<td>Option to generate the Java™ Web Archive.</td>
</tr>
<tr>
<td>--war</td>
<td>To run fglgar --war on a Windows or Mac, the user running the command must have write permission on the FGLASDIR directory. If you cannot grant write permissions on the FGLASDIR directory, move all of the FGLASDIR/war directory to another directory with write permission. See Building the Java Web Archive (WAR) on page 97</td>
</tr>
<tr>
<td>-g</td>
<td>Option to generate the Genero Web Archive.</td>
</tr>
<tr>
<td>--gar</td>
<td></td>
</tr>
<tr>
<td>-p directory</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--as-directory</td>
<td></td>
</tr>
<tr>
<td>directory</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--asdir directory</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>-x config_file</td>
<td>Specify which configuration file to use when starting the Java™ dispatcher. If not specified, the default configuration file, $FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>--asxcf config_file</td>
<td>Specify which configuration file to use when starting the Java™ dispatcher. If not specified, the default configuration file, $FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>-f config_file</td>
<td>Specify which configuration file to use when starting the Java™ dispatcher. If not specified, the default configuration file, $FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>--configuration-file config_file</td>
<td>Specify which configuration file to use when starting the Java™ dispatcher. If not specified, the default configuration file, $FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>--resource directory</td>
<td>Specify the Genero archive resource directory. See Resource deployment overview on page 257</td>
</tr>
<tr>
<td>--trigger-component</td>
<td>Genero archive trigger execution component, see TRIGGERS (for manifest) on page 263.</td>
</tr>
<tr>
<td>--deploy-trigger command</td>
<td>Genero archive deploy trigger command. See TRIGGERS (for manifest) on page 263</td>
</tr>
<tr>
<td>--undeploy-trigger command</td>
<td>Genero archive undeploy trigger command. See TRIGGERS (for manifest) on page 263</td>
</tr>
<tr>
<td>--application config_file</td>
<td>Genero archive application configuration file.</td>
</tr>
<tr>
<td>--service config_file</td>
<td>Genero archive service configuration file.</td>
</tr>
</tbody>
</table>

Example building a Web archive

fglgar --war --asdir C:\usr\gas\3.00 --asxcf C:\conf\as.xcf --output java-j2eedispatch

Generates a Java™ Servlet called java-j2eedispatch.war using the Genero Application Server located at C:\usr\gas\3.00 and with the configuration file located at C:\conf\as.xcf, and accessible via URL of type http://host:port/java-j2eedispatch/demos.html.

fglgar --war --asdir C:\usr\gas\3.00 --output connector

Generates a Java™ Servlet called connector.war using the Genero Application Server located at C:\usr\gas\3.00 and accessible via URL of type http://host:port/connector/demos.html.

The gasadmin command

The gasadmin tool is provided as an administrative command for the Genero Application Server.

With this tool, you can:

- Display GAS version information, see Display GAS version.
- Specify the GAS application directory, see Specify GAS directory
- List all sessions of a specified dispatcher, see List sessions
- Stop dispatcher sessions, see Kill sessions
- Validate the GAS configuration, see Validating with the gasadmin tool on page 99
- Explode the GAS configuration, see Explode configuration file
- Ping dispatchers active sessions, see Ping active sessions
- Manage archive files, see Genero Archive procedures on page 265
- Reset the log output for a session. See Reset logs on page 315
- Retrieve information to monitor a session. See Monitor session on page 315
Syntax

```
gasadmin [options]
```

Syntax with reset-log command

The `reset-log` command reconfigures the logs for one or more sessions. Options are described in Table 62: `gasadmin reset-log options` on page 313. This feature was added in the version 3.00 (MR) maintenance release.

```
gasadmin reset-log [options] [session-id ...]
```

Options

Table 61: `gasadmin options`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-p directory</code></td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td><code>--as-directory directory</code></td>
<td></td>
</tr>
<tr>
<td><code>-f configuration-file</code></td>
<td>Specify the configuration file to use. If not specified, the default configuration file, <code>FGLASDIR/etc/as.xcf</code>, is used.</td>
</tr>
<tr>
<td><code>--configuration-file configuration-file</code></td>
<td></td>
</tr>
<tr>
<td><code>-E resourceName=value</code></td>
<td>Define or overwrite a resource.</td>
</tr>
<tr>
<td><code>--resource-overwrite resourceName=value</code></td>
<td></td>
</tr>
<tr>
<td><code>-c</code></td>
<td>Check the configuration file, then exit.</td>
</tr>
<tr>
<td><code>--configuration-check</code></td>
<td></td>
</tr>
<tr>
<td><code>-e</code></td>
<td>Explode the GAS configuration into files, one for each application.</td>
</tr>
<tr>
<td><code>--configuration-explode</code></td>
<td></td>
</tr>
</tbody>
</table>

**Important**: If you start the dispatcher with the option `(-E)` to override the `$(res.appdata.path)` location, you must also override the resource when using the `gasadmin` command to deploy an archive.

For example, specify the same option with both commands:

- Starting the dispatcher:
  ```
  httpdispatch -E res.appdata.path=/work/tmp/gas/appdata
  ```
- Deploying the application:
  ```
  gasadmin -E res.appdata.path=/work/tmp/gas/appdata --deploy-archive myapp.gar
  ```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t --configuration-explode-external</td>
<td>Explode the given external configuration file in current directory. See Explode configuration file into an XML file on page 314</td>
</tr>
<tr>
<td>-r --configuration-expand-resources</td>
<td>Expand resources and replace with real values. Used with --configuration-explode or --configuration-explode-external. See Explode configuration file into XML files on page 314</td>
</tr>
<tr>
<td>-d dispatcher-name --dispatcher dispatcher-name</td>
<td>Used by session-related options to select the target dispatcher.</td>
</tr>
<tr>
<td>-K -d dispatcher-name --kill-all-sessions</td>
<td>Terminate (kill) all sessions of a specified dispatcher. Note: The option -d &lt;dispatcher_name&gt; is optional. If omitted, the option kills all sessions of all dispatchers.</td>
</tr>
<tr>
<td>-k session-id -d dispatcher-name --kill-session session-id --dispatcher dispatcher-name</td>
<td>Terminate (kill) the requested session id of the specified dispatcher. Important: The option -d &lt;dispatcher_name&gt; is required, as the same session identifier can exist for the various dispatchers. See Kill dispatcher sessions on page 315</td>
</tr>
<tr>
<td>-X</td>
<td>Ping all active sessions of the specified dispatcher. Note: The option -d &lt;dispatcher_name&gt; is optional. If omitted, the option pings all active sessions of all dispatchers.</td>
</tr>
<tr>
<td>-x session-id -d dispatcher-name --ping-session session-id --dispatcher dispatcher-name</td>
<td>Ping the request session id of the specified dispatcher.</td>
</tr>
<tr>
<td>-l --list-sessions --dispatcher dispatcher-name</td>
<td>List all sessions of the specified dispatcher. Note: The option -d &lt;dispatcher_name&gt; is optional. If omitted, the option lists all sessions of all dispatchers. See List dispatcher sessions on page 315</td>
</tr>
<tr>
<td>--dispatcher dispatcher-name --monitor session-id</td>
<td>Retrieve monitor information for a session. Information is displayed in XML format on the standard output. See Monitor session on page 315. Note: Monitor feature was added in 3.00 (MR) maintenance release.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>-z paths, [,...]</td>
<td>Compress the resources located in specified paths. The path separator is a comma (,). See Compress resources on page 315.</td>
</tr>
<tr>
<td>--compress-resources paths, [,...]</td>
<td>Compress the resources located in specified paths. The path separator is a comma (,). See Compress resources on page 315.</td>
</tr>
<tr>
<td>-C</td>
<td>Clear remaining Linux®/UNIX™ domain sockets.</td>
</tr>
<tr>
<td>--session-cleanup</td>
<td>Clear remaining Linux®/UNIX™ domain sockets.</td>
</tr>
<tr>
<td>-h</td>
<td>Display help information.</td>
</tr>
<tr>
<td>--help</td>
<td>Display help information.</td>
</tr>
<tr>
<td>-V</td>
<td>Display GAS version information. See Display GAS version information on page 315</td>
</tr>
<tr>
<td>--version</td>
<td>Display GAS version information. See Display GAS version information on page 315</td>
</tr>
<tr>
<td>-q</td>
<td>Silent mode.</td>
</tr>
<tr>
<td>--quiet</td>
<td>Silent mode.</td>
</tr>
<tr>
<td>--deploy-archive archive_file</td>
<td>Unpack the specified archive content into the deployment directory.</td>
</tr>
<tr>
<td>--undeploy-archive archive_file</td>
<td>Undeploy the specified archive.</td>
</tr>
<tr>
<td>--enable-archive archive_file</td>
<td>Expose all services and applications contained in the specified archive.</td>
</tr>
<tr>
<td>--disable-archive archive_file</td>
<td>Unexpose all services and applications contained in the specified archive.</td>
</tr>
<tr>
<td>--list-archives</td>
<td>List all archives deployed on the Genero Application Server.</td>
</tr>
<tr>
<td>--clean-archives</td>
<td>Clean up all undeployed archives.</td>
</tr>
<tr>
<td>--xml-output</td>
<td>List all archives and applications deployed on the Genero Application Server. Only compatible with archive options.</td>
</tr>
<tr>
<td>--monitor</td>
<td>Retrieve monitor information for a session. Information is displayed in XML format on the standard output. See Monitor session on page 315.</td>
</tr>
</tbody>
</table>

Table 62: gasadmin reset-log options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays help for the reset-log command.</td>
</tr>
<tr>
<td>--help</td>
<td>Displays help for the reset-log command.</td>
</tr>
<tr>
<td>-q</td>
<td>Operates in silent mode</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>--quiet</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>-p directory_name</td>
<td>Specify the configuration file to use. If not specified, the default configuration file, FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>-f configuration-file</td>
<td>Specify the configuration file to use. If not specified, the default configuration file, FGLASDIR/etc/as.xcf, is used.</td>
</tr>
<tr>
<td>-E resource_name=value</td>
<td>Define or overwrite a resource.</td>
</tr>
<tr>
<td>-d dispatcher-name</td>
<td>Target the dispatcher - used by session-related options to select the target dispatcher.</td>
</tr>
<tr>
<td>-O output-type &lt;arg&gt;</td>
<td>Log type (CONSOLE or DAILYFILE), default is DAILYFILE.</td>
</tr>
<tr>
<td>-O output-path &lt;arg&gt;</td>
<td>Log output directory for DAILYFILE, default is the current directory.</td>
</tr>
<tr>
<td>-O raw-data-max-length &lt;arg&gt;</td>
<td>Data max length, default is -1.</td>
</tr>
<tr>
<td>-O format &lt;arg&gt;</td>
<td>Log format, columns to output, the default is: time event-type event-params.</td>
</tr>
<tr>
<td>-O categories &lt;arg&gt;</td>
<td>Log categories to enable, the default is: GAS ACCESS PROCESS ERROR</td>
</tr>
</tbody>
</table>

**Note:** The reset-log feature is not enabled with java-j2eedispatch, as it does not store sessions on disk.

**Usage Examples**

**Explode configuration file into an XML file**

Explode external application configuration file demo/Card.xcf and expand its resources and its parent's resources into an XML file.

```
gasadmin -t demo/Card
```

**Explode configuration file into XML files**

Explode external application configuration file demo/Card.xcf. Replace its resources and its parent resources with real values, and expand them into individual XML files.

```
gasadmin -r -t demo/Card
```
## List dispatcher sessions

List all sessions having `as1.xcf` as configuration file for dispatcher `httpdispatch`

```
gasadmin -X -f as1.xcf -d httpdispatch
```

## Kill dispatcher sessions

Kill the session with the id "d98290172c8f7c0d861db329f1ce6597" that has `as1.xcf` as its configuration file, and `isapidispatch` as its dispatcher.

```
gasadmin -k d98290172c8f7c0d861db329f1ce6597 -f as1.xcf -d isapidispatch
```

## Monitor session

This example shows how to use the `--monitor` option to retrieve information to monitor a specified session. Information on the current status of the dispatcher is sent to the standard output in XML format during the session.

```
gasadmin --dispatcher <dispatcher> --monitor <session-id>
```

The `-d` or `--dispatcher` option is required to specify the dispatcher.

## Compress resources

Compress the resources located in the listed paths.

```
gasadmin -z FGLASDIR/app, FGLASDIR/services, FGLASDIR/web, FGLASDIR/tpl
```

## Reset logs

```
gasadmin reset-log --output-type CONSOLE --categories "ALL DEBUG" 1170f560ca4d03fd3aa4bbac75da97e9
```

The example command uses options for `--output_type` and `--categories`. The output type effects where the logs are sent, either the console or the daily log file (DAILYFILE). The categories effects the type of logs output. These changes only effect the specified session.

**Tip:** You can specify multiple sessions by listing the session ids, separated by spaces.

If an option (for example, `--output_path`) is not specified, `gasadmin` tries to use the LOG configuration from the `as.xcf`. If not found in the `as.xcf` (for example, CONSOLE may not be configured in `as.xcf`), default values are used.

## Display GAS version information

```
gasadmin -V
```
Ghost Client and Testing Tools

The Genero Ghost Client (GGC) is a Java framework that allows you to test your applications. You can create different scenarios and user interaction on applications that the GGC can run automatically on a Genero server; that is the Genero Application Server (GAS) and runtime. It can also be used without the GAS in direct connection mode using a stand-alone Dynamic Virtual Machine (DVM).

It allows you to test the business logic and size the infrastructural needs of your applications before release. The GGC acts as a “ghost client” because it does not render a graphical user interface. Therefore you can use it to test applications for different front-ends: Genero Web Client for JavaScript (GWC-JS), GWC-HTML5, GDC, GMA, or GMI. When you want to do load or performance testing, you can specify it to run tests acting as one or more concurrent users from one machine.

You can use the Ghost Client to automate the following tests:

- Unit tests
- Load tests
- Performance tests

Test scenarios can be developed in two ways:

- Tests can be written in both Java and Genero BDL and compiled to Java for testing by the Ghost Client.
- Test scenarios can be generated from the behavior described in a log file recorded by GDC or GWC-JS clients. The Ghost Generator feature generates Java or BDL files automatically from the log file data, which allows you to replay the resulting scenarios with the Ghost Client. See Generate tests from Sample log files on page 330.

One of the key features of testing with the Ghost Client is that you do not need to modify the original application code to write test scenarios.

The Ghost Client allows you to develop working test case scenarios that can be run as required to test the stability of your applications before release.

- Installing Ghost Client on page 317
- Configure your environment for Ghost Client on page 317
- How Ghost Client works on page 318
- Unit testing on page 324
- Load testing on page 325
- Explore Java demos on page 328
- Explore BDL demos on page 329
- How to compile and run Java tests on page 325
- Generate tests from Sample log files on page 330
Installing Ghost Client
This topic provides information about how to install your Ghost Client API package.

About this task
The procedure in this topic shows you how to carry out an installation of Ghost Client.

Before you begin:
- Ensure Genero Ghost Client is supported for your operating system. For a list of supported operating systems, refer to the download page (available on the Four Js Web site) or contact your support center.
- To use Genero Ghost Client API, you need a Java Development Kit (JDK). Make sure that your JDK version is at least version 1.7 or greater.

Once the JDK has been installed, you are ready to install the Ghost Client as described in the next step.
1. Download the package appropriate for your operating system.
2. Unzip the GGC package to a directory on your server.

What to do next:
Configure your environment for the Ghost Client as detailed in Configure your environment for Ghost Client on page 317.

Configure your environment for Ghost Client
This topic provides information about how to configure your environment to use the Ghost Client.

About this task
The procedure in this topic shows you how to configure your environment so that Ghost Client can use Java.

Before you begin:
- You must have the Java Development Kit installed. See Installing Ghost Client on page 317.
  
  Important: Before you can run scenarios with the GGC, you need to set up your PATH and CLASSPATH environment variables as described in the next steps.

1. Set your PATH environment variable to include the Java Development Kit bin directory.
   On Linux®/UNIX™/MAC®:
   ```sh
   export PATH= /path-to-jdk/bin:$PATH
   
   On Windows®:
   ```
   ```
   set PATH = C:\path-to-jdk\bin;%PATH%
   
   2. To be able to compile and run test scenarios from any directory on your disk, set your CLASSPATH environment variable to the absolute path to the ggc.jar and fgl.jar files.
      Note: You can also specify the path to jar libraries directly at the command line using the Java -classpath (-cp ) option, see Example using Java -cp option below.

      On Windows®:
      ```
      set CLASSPATH = C:\path-to-the-ggcdir\ggc.jar;C:\path-to-fgldir\fgl.jar
      On Linux®/UNIX™/Mac®:
      ```
      ```
      export CLASSPATH =/path-to-the-ggcdir/ggc.jar:/path-to-fgldir/fgl.jar
      
      Note: CLASSPATH references the jars needed by the GGC. You can compile and run your scenarios as shown in these examples:
      - Compiling: javac userWorkspace/path/to/generated/*.java
• **Running:**

An alternative to setting the *CLASSPATH* environment variable is to specify the absolute path to the *ggc.jar* file using the *java -classpath (-cp)* option at runtime. Below are some examples.

**On Linux®/UNIX™/Mac®:**

- **Compiling:**
  javac -cp /absolute/path/to/userWorkspace:/absolute/path/to/ggc.jar userWorkspace/path/to/generated/.*java

- **Running:**

**On Windows®:**

- **Compiling:**
  javac -cp C:\absolute\path\to\userWorkspace;C:\absolute\path\to\ggc.jar userWorkspace\path\to\generated\.*java

- **Running:**
  java -cp C:\absolute\path\to\userWorkspace;C:\absolute\path\to\ggc.jar com.fourjs.ggc.Launcher -s userWorkspace.path.to.generated.UserClass -u http://application/url

For more information on `-classpath (-cp)` option, see the Java documentation.

---

**How Ghost Client works**

The **Ghost Client** is a Java framework that provides you with the API structure of interfaces, classes and methods for building application tests.

**The Ghost Client infrastructure overview**

The **Ghost Client** infrastructure consists of three components:

- **Launcher**
  The **Launcher** is the **Ghost Client** Java program which starts a testing session for an application.

- **SessionManager**
  The **SessionManager** is the interface that manages the runtime and the test **Scenario**.

- **Scenario**
  The **Scenario** is the interface that contains the sequence of tests run during a given session.

**Ghost Client classes**

The methods of these public classes are called in the **SessionManager** and **Scenario** classes you define for your test. The **GhostRunner** class, for example, is the entry point for test case scenarios.
Table 63: Ghost Client classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GhostRunner</td>
<td>Provides methods to control a Genero application running in a Genero Application Server. It contains methods for a set of possible actions that an end-user might do on the running application, e.g. set focus on a field, set a value in a field, etc.</td>
</tr>
<tr>
<td>Log</td>
<td>Implements the logging mechanism used to displays errors, warnings, and information as output from the tests.</td>
</tr>
<tr>
<td>Session</td>
<td>Exposes the application object to your BDL program so that it can be retrieved from the session.</td>
</tr>
<tr>
<td>Application</td>
<td>Exposes the application user interface to your BDL program so that form objects such as field values can be retrieved and set.</td>
</tr>
</tbody>
</table>

The complete details of the packages that make up the Ghost Client and the classes and interfaces it uses can be found in the /doc directory of your GGC package. For more information please see the help file by launching the /doc/index.html file in your browser.

Developing tests with Ghost Client

When you write tests to be run by the Ghost Client, you need to implement its two main interfaces: the SessionManager and the Scenario. These classes function together to implement the GGC tests.

Table 64: Ghost Client interfaces

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionManager</td>
<td>Instantiates and manages Scenario instances according to incoming VM connections and new runtime launches.</td>
</tr>
<tr>
<td>Scenario</td>
<td>A Scenario describes the action sequence played during the GhostRunner session to simulate user actions on a Genero application.</td>
</tr>
</tbody>
</table>

For more detailed information on developing tests with Ghost Client see Testing with Ghost Client on page 322.

Ghost Client options

The Ghost Client allows you to run tests in different ways using various command line options.

- Display Ghost Client help on page 320
- Display Version Number on page 321
- Command line examples on page 321

This section provides you with a reference guide to the available command line options and includes some examples to help you with using the Ghost Client to generate application tests.

Syntax

```
java com.fourjs.ggc.Launcher [options]
```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-u URL</td>
<td>Specify the URL to use to test your application, see Launch tests to different front-ends on page 321.</td>
</tr>
<tr>
<td>--command &quot;fglrun &lt;myapp&gt;&quot; (-cd)</td>
<td>Specify a test from a stand-alone DVM, using the direct connection mode. (Note: Not used with the URL -u option), see Launch tests with direct connection on page 322</td>
</tr>
<tr>
<td>--manager (-s) path.to.mySessionManager</td>
<td>Specify the SessionManager java file to use.</td>
</tr>
<tr>
<td>--thread_number (-t) value</td>
<td>Specify the number of thread instances to launch to simulate the number of users using an application.</td>
</tr>
<tr>
<td>--thread_delay (-td ) value</td>
<td>Define the delay in seconds between the launch of each thread instance. For thread and thread delay options usage examples, see Load testing on page 325</td>
</tr>
<tr>
<td>--no_x509_check -x</td>
<td>Prevent the server X.509 certificates checking.</td>
</tr>
<tr>
<td>--all_mode</td>
<td>Launch the test for all protocols, /wa/r/, /ja/r/, /ua/r/.</td>
</tr>
<tr>
<td>--write (-w) path/to/generate/java_files</td>
<td>Write Java files generated from a log file to specified path</td>
</tr>
<tr>
<td>--log (-l) path/to/log_file</td>
<td>Specify a log file to use to generate a test set, see Generate Java test scenarios on page 330</td>
</tr>
<tr>
<td>--BDL</td>
<td>Command used with the --write and --log options to have the Ghost Generator write 4gl files instead of Java files generated from a log file.</td>
</tr>
</tbody>
</table>

### Display Ghost Client help

To display the help for the Ghost Client, execute the command:

```
java com.fourjs.ggc.Launcher --help
```
Display Version Number

To display the version of the Ghost Client, execute the command:

```
java com.fourjs.ggc.Launcher --version
```

Command line examples

Ghost Client command line examples showing the use of different options.

Note: The GGC command line is a string without line breaks. Line breaks may have been added to
the command line examples in this section to support the printed version of our documentation.

- Launch tests to different front-ends on page 321
- Launch tests with direct connection on page 322
- Generate test sets from log file on page 335

Launch tests to different front-ends

Run tests behaving as different front-ends.

In the syntax of the following command line examples it is assumed that the GGC package was unzipped
in the same directory where you are running the Launcher, and that your environment is configured to run
Java and the GGC, see Configure your environment for Ghost Client on page 317.

Depending on how you want to test, run the command option in example 1 to run your application tests
against different front-end clients, or run the command option in example 2 to test all clients at once:

1. `java com.fourjs.ggc.Launcher -u http://localhost:6394/ua/r/myapp -s path.to.mySessionManager.mySessionManager`
The **Launcher** is the entry point of the Ghost Client Java program which starts a testing session for an application. This command is run by launching Java to run the **Launcher** class followed by the `-u` option and the URL of your application. The `-s` option is used followed by the path to the session manager.

**Tip:** Depending on the user agent protocol you provide in the URL, you can specify what front-end the test is to be run as:

- `/ua/r/`; for version 3 clients: Genero Web Client for JavaScript (GWC-JS), GDC 3.00, GMA, or GMI - UAProxy.
- `/ja/r/`; for Genero Desktop Client (GDC 2.50) - GDCProxy.
- `/wa/r/`; for Genero Web Client for HTML5 (GWC-HTML5) - HTML5Proxy.

2. `java com.fourjs.ggc.Launcher -u http://<host>:<port>/gas/ua/r/<group>/<myapp> -s path.to.mySessionManager.mySessionManager --all_mode`

This example is basically the same instruction but with the addition of the `--all_mode` option at the end to specify that the test is to be run for all clients. Note, the front-end you specify in the URL will be the first one to be run.

**Launch tests with direct connection**

Run the integrity tests in direct connection mode.

In the syntax of the following command line examples it is assumed that the GGC package was unzipped in the same directory where you are running the **Launcher**, and that your environment is configured to run Java and the **GGC**, see [Configure your environment for Ghost Client](#) on page 317.

To run the integrity tests for the demo application from a stand-alone DVM, using the direct connection mode, you run this command:

```
java com.fourjs.ggc.Launcher -s com.fourjs.ggc.testcases.IntegrityTestsSessionManager --command "fglrun demo"
```

In this example a URL is not used to launch the application. Therefore instead of the `-u` option, the **Launcher** is run with just the `-s` option and the path to the session manager.

The `--command` option is added to call the DVM to launch the application for testing.

**Note:** The DVM command line to launch the application needs to be passed in double quotes ("fglrun demo").

**Testing with Ghost Client**

Ghost Client tests are developed separate to your actual 4gl application source code via interfaces that you write.

Typically, you write two separate classes for a test written in Java; the **SessionManager** and the **Scenario**.

For tests written in BDL, there will be only one file containing both the **SessionManager** and **Scenario**.

Both Java and BDL can be used to write these tests but they must be compiled to Java to be run by the **GGC**.

**Note:** In Genero Studio you can write tests using BDL and compile them to Java using the Ghost Client infrastructure through the Java Bridge. The Java Bridge allows you to import **GGC** classes, which opens up their methods for use. For more information about using BDL, please see the Genero Studio project, **GGC_Sample.4pw**, available in the samples/BDL directory.

- [Write a Java session manager](#) on page 323
- [Write a Java scenario](#) on page 323
- [Unit testing](#) on page 324
- [Load testing](#) on page 325
- [How to compile and run Java tests](#) on page 325
• Using the logging mechanism on page 326

Write a Java session manager

The `SessionManager` is a program written in Java that manages the whole test session life cycle.

To instantiate the `Scenario` or the sequence of scenarios required to run your tests, include a `getScenario()` function in your `SessionManager`.

Example: Java Session Manager

The `Scenario` instance created is given to the main application (i.e. the first one launched), which it needs to get to run the tests.

If child applications are run by the main application, the `SessionManager` will also get each of them a `Scenario` and instantiate it, as shown in the example.

```java
import com.fourjs.ggc.Scenario;
import com.fourjs.ggc.SessionManager;

public class mySessionManager
    implements SessionManager
{
    boolean mStarted = false;
    public Scenario getScenario()
    {
        if (!mStarted) {
            mStarted = true;
            return new FirstScenario(); << will be given to the first 4gl application, the mother
        } else {
            return new SecondScenario(); << will be given to the second 4gl application, the child launched via run or run without waiting by the mother
        }
    }
}
```

Write a Java scenario

A `Scenario` is a program written in Java that tests different functions of your application.

Your `Scenario` requires these methods:

• A `play(GhostRunner runner)` method, which describes the user interaction sequence to play; that is the sequence of tests it plays.
• An `InvokeFrontcall(String module, String name, String[] args)` method, which describes how to handle the call to the front-end client when and if it is required by the application.
• A `getScenarioListener()` method. This method is called by the GhostRunner framework to perform runtime monitoring.

You can decide what tests are to be carried out by providing within the `play(GhostRunner runner)` function of your scenario the required user interactions.

**Note:**

The complete details of the packages that make up the Ghost Client and the classes and interfaces it uses can be found in the `/doc` directory of your GGC package. For more information please see the help file by launching the `/doc/index.html` file in your browser.

Any interaction a user would normally do on an application can be played by a Ghost Client API method:

• **Introspecting the application AUI Tree**
  • List available actions
- List available fields
- Get value of a given field
- Get field that has the focus

**Executing a user action**

- Send an action
- Input a value in a field
- Scroll through a table
- Open a tree view node

```java
/**
 * The Scenario to test the "CustOrders" application from the FGLGWS demo
 */
public void play(GhostRunner runner) {
    mRunner = runner;
    try {
        // at least, we expect these actions
        String[] expected = { "zoom_city", "cust_query", "cust_next", "cust_last", "dialogtouched", "cust_append", "cust_delete", "ord_append", "ord_modify", "ord_delete", "close" };
        ArrayList<ActionInfo> actions = mRunner.getActions();
        for (String act : expected) {
            boolean present = false;
            for (ActionInfo action : actions) {
                if (act.equals(action.getName())) {
                    present = true;
                    break;
                }
            }
            if (!present) {
                runner.log().error("CustomerOrderScenario: action [", act, "] is missing");
            }
            ... 
        }
    }
}
```

In this code sample from the `CustomerOrderScenario` class sample, the CustOrders application from the FGLGWS demo is being tested for integrity.

The `mRunner.getActions()` method passes details about the form's menu actions into an array and checks these against an expected list of actions. If an action menu is missing, the `runner.log().error` method outputs the information to the standard output.

**Unit testing**

The aim of unit testing is to test each feature of your application in isolation to make sure it works as expected.

The Ghost Client provides a framework for unit tests. You can develop these tests via a Scenario interface.

**What is unit testing?**

A good unit test should provide you with the correct responses to a given set of anticipated user input, showing that the feature is able to handle correct as well as incorrect input. As a developer approaching unit testing for the first time, the following is recommended as good practice when designing unit tests:

- It is recommended that each 4gl application should have its own Scenario.
• Make a complete list of the application features, from the smallest (for example, displaying the About... screen), to the biggest and/or the most important ones.
• For each feature in this list, write a single test that will test one (and only one) feature.
• If a primary feature implies a secondary feature (for example, to register a new customer, you need to fill out a form), the secondary feature's test should take place before the primary feature's test in the test sequence.

Load testing
This testing strategy consists of simulating a specified number of user, all of them using your application at the same time and at normal human speed.

What is load testing?
You can use the GGC to test your application during development to see how it would behave in conditions in a production environment. This can help you identify, for example, server (e.g. how many servers, how much memory, etc.) and network requirements based on the number of anticipated users. Observations made under these test conditions will also show up the application's weaknesses and will allow you to fix them before release.

Load testing with Ghost Client
The GGC allows you to specify several options through the Launcher class for load testing:
• The thread_number option allows you to set the number of users to simulate, one thread representing one user.
• The thread_delay option allows you to set a delay between each user's connection. In a real life situation where, for example, a thousand users could potentially launch an application, the likelihood of them all connecting at the exact same time is quite remote, so the thread delay option helps you to recreate a more real life situation.

You can specify the number of threads (users) and the thread delay using the command line switches when launching tests. As shown in the example below, the GGC will launch the IntegrityTestsSessionManager with 3 users, allowing a 3 second delay between each launch.


Developing Scenario for load testing
The Scenario for load testing needs to reflect how a real user would use an application. For example, the setDelay() method can be set to make the GhostRunner instance wait for a specified number of seconds between each action. This allows you to try to reproduce the human speed of interaction with an application, which is usually much slower than computer speed. Apart from that any interaction a user would normally do on an application can be played.

Performance testing
Performance testing is similar to load testing as it also consists of simulating a significant number of users but the aim is to determine when the system's performance will start to degrade. So, in this case you would specify the number of simulated user as high as possible so as to observe when the request/response delay begins to be affected by the number of users.

How to compile and run Java tests
This topic provides information about how to compile and run your own tests.

About this task
The procedure in this topic shows you how to compile and run Java tests you have developed.

Before you begin:
- It is assumed that you have built two new test java files:
  - One SessionManager implementation named, for example, mySessionManager.java that will manage the testing session
  - One Scenario implementation named, for example, myScenario.java that will instantiate real scenarios based on the application launched by the GAS
- It is assumed that your new test java files are in the package path.to.myTests and therefore you have placed them in the directory path/to/myTests.
- It is assumed your environment is configured to run Java and the GGC, see Configure your environment for Ghost Client on page 317.
- If you are using the GAS:
  - Make sure your GAS version is at least version 2.50.34 or greater.
  - Make sure that the standalone dispatcher httpdispatch see Dispatcher: httpdispatch on page 304 is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser.

1. To compile your newly created tests, type the following:
   ```
   javac path/to/myTests/*.java
   ```

2. To run the tests through the GAS using the application URL, type the command:
   ```
   ```
   **Tip:** Depending on the user agent protocol you provide in the URL, you can specify what front-end the test is to be run as:
   - /ua/r/; for version 3 clients: Genero Web Client for JavaScript (GWC-JS), GDC 3.00, GMA, or GMI - UAProxy.
   - /ja/r/; for Genero Desktop Client (GDC 2.50) - GDCProxy.

Or alternatively you can run the test in direct connection mode without using the GAS.

3. To run the tests using a stand-alone DVM, type the command:
   ```
   java com.fourjs.ggc.Launcher -s path.to.myTests.mySessionManager --command "fglrun myApplication"
   ```
   **Note:** The DVM command line to launch the application just needs to be passed in double quotes, for example "fglrun myApplication".

**Using the logging mechanism**

The Ghost Client implements a logging mechanism to displays errors, warnings, and information as output from the tests.

- GHOSTLOG on page 326
- Log test scenario output on page 327

**GHOSTLOG**

The environment variable that defines the logging level for the Ghost Client.

You can adjust the log level with a numeric or literal value. The GHOSTLOG setting defines what information is outputted as tests are run, based on the logging detail included in your test play method, see Log test scenario output on page 327.

Valid literal values include:

- INFO
- ERROR
- WARNING
- DEBUG
The Launcher automatically sets GHOSTLOG to INFO but if you want a different log level, you can set it at the command line before launching a test.

- On Linux®/UNIX™/MAC®
  
  ```
  export GHOSTLOG=ERROR
  ```

- On Windows®:
  
  ```
  set GHOSTLOG = ERROR
  ```

**Log test scenario output**

Use the Log class methods to output errors, warnings, and information from test scenarios as they play.

In the `play` method of your Java test scenarios, for example, to have information about errors, warnings, or debug information displayed as the test is running, add calls to the log class with the required log method, for example:

```java
runner.log().error("This is bad..."); // for the error
runner.log().warning("Beware !!! "); // for the warning
runner.log().info("Just to tell you..."); // for the info
runner.log().debug("Bug to report..."); // for debug info
```

```java
public void play(GhostRunner runner)
{
    try {
        runner.sendAction("INPUT");
        ArrayList<FieldInfo> fields = runner.getFields();
        for (FieldInfo field : fields) {
            runner.log().info(field.getId(), " / ", field.getName(), " / ", field.getType());
        }
        runner.sendAction("close");
        runner.close();
    } catch (GhostException e) {
        runner.log().error("ButtonEditDelegatedScenario: exception raised!", e);}
    catch (Exception e) {
        runner.log().error(e);}
    ...
    runner.log().info(this.getClass().getName(), " ended successfully");
}
```

In this code sample from the ButtonEditScenario class sample, the ButtonEdit application from the FGLGWS demo is being tested. The `runner.sendaction("INPUT")` method selects the INPUT button of the application. This opens a form for user input.

The `runner.getFields()` method gets details about the form fields; their id, name, and type. Using the `runner.log().info` method outputs this information.

In the `catch` block, any exception raised by the `GhostException` method can also be logged as an error by the `runner.log().error` method.

![Figure 71: ButtonEdit scenario log output](image_url)
You can adjust the log level with the GHOSTLOG environment variable. This will effect what is outputted to the display, but bear in mind it depends on how detailed the logging is in your test play method.

If GHOSTLOG is set to ERROR, only errors will be displayed on the output.
If set to WARNING, errors and warnings are displayed.
If set to INFO, errors, warnings, and info are displayed.
If set to DEBUG, errors, warnings, info, and debug information, basically everything is displayed.

**Exploring Ghost Client demos**
The Genero Ghost Client installation includes tests samples for you to explore in the `samples` directory of your GGC package.

The topics in this section shows you how to run integrity tests on the `gwc-demo` applications using the BDL and Java demos provided with your Ghost Client installation.

- Ghost Client samples directory on page 328
- Explore Java demos on page 328
- Explore BDL demos on page 329

**Ghost Client samples directory**
The `samples` directory contains BDL and Java demo files, programs, and recorded log files.

The default location for Genero Ghost Client sample files is the `samples` directory in your GGC package installation.

- **java**
  Contains the `IntegrityTestsSessionManager` and various `Scenario` implementations that are invoked through the `IntegrityTestsSessionManager` to test the `gwc_demo` applications. All are written in Java.

- **BDL**
  Contains the `GGC_Sample.4pw` project and associated (4gl) files for running GGC tests. All are written in Genero Business Development Language (BDL).

The `samples` directory contains a set of recorded log files that can be used to automatically generate valid test sets in Java or BDL, see Generate tests from Sample log files on page 330.

**Explore Java demos**
This topic provides information about how to view the Ghost Client Java demos provided with the installation.

**About this task**
The procedure in this topic shows you how to use the Java demos provided with the Ghost Client installation to run integrity tests on the `gwc-demo` applications via the Ghost Client.

Before you begin:

- Make sure that the directory where your GGC package is installed has the following samples included in its `/samples/java` directory:
  - One `SessionManager` implementation named `IntegrityTestsSessionManager.java` that will manage the testing session
  - One `Scenario` implementation named `GenericScenario.java` that will instantiate real scenarios depending on the application launched by the GAS
  - Several simple `Scenario` implementations that will be invoked through the `GenericScenario`
  - Make sure your environment is configured to run Java and the GGC, see Configure your environment for Ghost Client on page 317.
• If you are using the GAS:
  • Make sure your GAS version is at least version 2.50.34 or greater.
  • Make sure that the standalone dispatcher httpdispatch see Dispatcher: httpdispatch on page 304 is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser.

1. To run the test simulating a GDC 2.50 over HTTP client, type the command as shown below:

   ```java
   ```

   Below is sample output logged by the GGC to the console during the running of integrity tests for CustomerOrderScenario.

   ![Sample Output]

2. To run the test simulation on GDC, GWC-JS, GMA, or GMI clients, type the command shown below:

   ```java
   ```

   Or alternatively you can run the test in direct connection mode without using the GAS.

3. To run the tests using a stand-alone DVM, type the command:

   ```java
   java com.fourjs.ggc.Launcher -s path.to.myTests.mySessionManager --command "fglrun demo"
   ```

   **Note:** The DVM command line to launch the application needs to be passed in double quotes, for example "fglrun demo".

**Explore BDL demos**

This topic provides information about how to view the BDL demos provided with the Ghost Client installation.

**About this task**

The procedure in this topic shows you how to use the BDL demos provided with the Ghost Client installation to test the integrity of the gwc-demo applications in simulation mode via the Ghost Client.
Before you begin:

- Make sure that the `samples/BDL` directory where your GGC package is installed has the `DemoTests.4gl` file included.
- If you are using the GAS:
  - Make sure your GAS version is at least version 2.50.34 or greater.
  - Make sure that the standalone dispatcher `httpdispatch` see Dispatcher: `httpdispatch` on page 304 is started and that you can access the GAS demos welcome page, `http://localhost:6394/demos.html`, from your browser.
- To compile and run the test you need to have your environment configured to load the Java Virtual Machine, see Configure your environment for Ghost Client on page 317.

1. To compile `DemoTests.4gl`, at the command line of the directory where the GGC is installed, type the following:

   ```
   cd samples/BDL ; fglcomp DemoTests.4gl
   ```

2. To run the tests through the GAS using the application URL, at the command line of the directory where the GGC is installed, type the command:

   ```
   cd samples/BDL ; fglrun DemoTests http://localhost:6394
   ```

3. To run the tests using a stand-alone DVM, type the command:

   ```
   cd samples/BDL ; fglrun DemoTests "fglrun demo"
   ```

   **Note:** The DVM command line to launch the application just needs to be passed in double quotes, for example "fglrun demo".

Generate tests from Sample log files

The topics in this section detail how to generate a Java or BDL file set of `SessionManager` and `Scenario` classes from log files found in the `samples` directory in your GGC package installation.

This section includes tasks to show you how to run integrity tests on the `gwc-demo` applications from the files sets generated in both Java and BDL.

- Generate Java test scenarios on page 330
- Generate BDL test scenarios on page 331
- Run the demo integrity tests with Java on page 332
- Run the demo integrity tests with BDL on page 333

Generate Java test scenarios

Generate a Java file set of `SessionManager` and `Scenario` classes from recorded log files.

Before you begin:

- Make sure that the `/samples` directory where your GGC package is installed has the following log files:

  - `StartMenu.log`
  - `DemoPanel.log`

- It is assumed your environment is configured to run Java and the GGC, see Configure your environment for Ghost Client on page 317.

1. Type the command:

   ```
   java com.fourjs.ggc.Launcher --write path/to/generate/DemoPanel --log samples/DemoPanel.log
   ```

   **Note:**
• The --write option writes the test file set to path/to/generate/. If some or all directories in the specified path, do not already exist, they will be created. If the path specified is not absolute, GGC will consider its own directory as root.

• The name of the generated class file is fully qualified. Therefore it will be made up of the series of directories from the CLASSPATH to the path to where the compiled classes are located (the package), with the file name (without extension) added at the end:

  - **Package name**
    
    path.to.generate

  - **Path to files**
    
    path/to/generate

• If Java files with the same name already exist at the same location, an error will be raised.

Once the Ghost Generator feature has been run, the generated Java files need to be compiled as described in the next step.

2. To compile your newly generated tests, type the following:

   javac path/to/generate/*.java

   The resulting file set can either be executed as is, see Run the demo integrity tests with Java on page 332, or can be modified to do additional tests.

Generate BDL test scenarios

Generate a BDL file set of SessionManager and Scenario classes from a recorded log file.

**Before you begin:**

• Make sure that the /samples directory where your GGC package is installed has the following log files:
  
  - StartMenu.log
  - DemoPanel.log

• It is assumed your environment is configured to run Java and the GGC, see Configure your environment for Ghost Client on page 317.

1. To generate a BDL file set using sample log files, type the command shown below:

   java com.fourjs.ggc.Launcher --write path/to/generate/DemoPanel --log samples/DemoPanel.log --BDL
   java com.fourjs.ggc.Launcher --write path/to/generate/StartMenu --log samples/StartMenu.log --BDL

   Where the --BDL option at the end of the command specifies that a 4gl file set, instead of Java files test set, is to be written.

**Note:**

• The --write option writes the test file set to path/to/generate/. If some or all directories in the specified path, do not already exist, they will be created. If the path specified is not absolute, GGC will consider its own directory as root.

• The name of the generated file is fully qualified. Therefore it will be made up of the series of directories from the CLASSPATH to the path to where the compiled files are located (the package), with the file name (without extension) added at the end:

  - **Package name**
    
    path.to.generate

  - **Path to files**
    
    path/to/generate

• If BDL files with the same name already exist at the same location, an error will be raised.
Once the Ghost Generator feature has been run, the generated 4gl files need to be compiled as described in the next step.

2. To compile your generated tests, type the following:

   cd path/to/generate/DemoPanel ; fglcomp DemoPanel.4gl  
   cd path/to/generate/StartMenu ; fglcomp StartMenu.4gl

   The resulting file set can either be executed as is, see Run the demo integrity tests with BDL on page 333, or can be modified to do additional tests.

**Run the demo integrity tests with Java**

Run an integrity test on the gwc-demo applications from scenarios generated from a log file as described in this task.

Before you begin:

- It is assumed you have generated the test classes from a recorded log file and compiled the files as detailed in Generate Java test scenarios on page 330.
- It is assumed that your new test Java files are in packages placed in the directory path/to/generate:
  - path.to.generate.StartMenuSessionManager
  - path.to.generate.DemoPanelSessionManager.
- If you are using the GAS:
  - Make sure your GAS version is at least version 2.50.34 or greater.
  - Make sure that the standalone dispatcher httpdispatch see Dispatcher: httpdispatch on page 304 is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser.

About this task: The integrity test on the gwc-demo application can be run on all three available protocols or using the direct connection mode.

1. To run the test simulation for Genero Web Client for JavaScript (GWC-JS) clients, type the commands shown below:


2. To run the test simulation for GWC-HTML5 clients, type the commands shown below:


3. To run the test simulation for GDC clients, type the commands shown below:


4. To run the tests using a stand-alone DVM, type the commands:

   cd FGLDIR\demo ; java com.fourjs.ggc.Launcher -s path.to.generate.StartMenuSessionManager --command "fglrun demo"  
   cd FGLDIR\demo ; java com.fourjs.ggc.Launcher -s path.to.generate.DemoPanelSessionManager --command "fglrun demo"
Important:

It is assume that the all the Java session and scenario class files; StartMenuSessionManager.class and DemoPanelSessionManager.class, etc, have been placed in the same directory as the tested application, FGLDIR/demo.

As file names must be fully qualified, for example path.to.generate.DemoPanelSessionManager, files must be located in the corresponding path, path/to/generate, from where the DVM command is run.

Run the demo integrity tests with BDL

Run an integrity test on the gwc-demo applications from scenarios generated in Genero BDL from a log file as described in this task.

Before you begin:

- It is assumed you have generated the test files from a recorded log file and compiled them as detailed in Generate Java test scenarios on page 330.
- It is assumed that your new test Genero BDL files are in the directory path/to/generate:
  - StartMenu
  - DemoPanel
- If you are using the GAS:
  - Make sure your GAS version is at least version 2.50.34 or greater.
  - Make sure that the standalone dispatcher httpdispatch see Dispatcher: httpdispatch on page 304 is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser.

About this task: The integrity test on the gwc-demo application can be run on all three available protocols or using the direct connection mode.

1. To run the test simulation for Genero Web Client for JavaScript (GWC-JS) clients, type the commands shown below:

```bash
cd path/to/generate ; fglrun StartMenu http://localhost:6394/ua/r/gwc-demo
cd path/to/generate ; fglrun DemoPanel http://localhost:6394/ua/r/gwc-demo
```

2. To run the test simulation for GWC-HTML5 clients, type the commands shown below:

```bash
cd path/to/generate ; fglrun StartMenu http://localhost:6394/wa/r/gwc-demo
cd path/to/generate ; fglrun DemoPanel http://localhost:6394/wa/r/gwc-demo
```

3. To run the test simulation for GDC clients, type the commands shown below:

```bash
cd path/to/generate ; fglrun StartMenu http://localhost:6394/ja/r/gwc-demo
cd path/to/generate ; fglrun DemoPanel http://localhost:6394/ja/r/gwc-demo
```

4. To run the tests using a stand-alone DVM, type the command:

```bash
Important:

It is assume that the BDL scenario files; StartMenu and DemoPanel, have been placed in the same directory as the tested application, for example FGLDIR\demo.

```bash
cd FGLDIR\demo ; fglrun StartMenu "fglrun demo"
cd FGLDIR\demo ; fglrun DemoPanel "fglrun demo"
```
**Recording log files**
A recorded log file can be used by the Ghost Client to store the scenario for testing. You can record logs for the purpose of generating scenarios for testing.

The topic in this section details how to record a log file from both the Dynamic Virtual Machine (DVM) or the Genero Desktop Client (GDC).

- Record a scenario log on page 334
- Generate test sets from log file on page 335

**Record a scenario log**

**About this task:** You can record a log file of a user's interaction with an application's graphical user interface (GUI) from either the runtime Dynamic Virtual Machine (DVM) end or the front-end.

**To record the logs from the DVM**

1. To record a log from the DVM end, type the commands shown below:

   ```
   cd $FGLDIR/demo; fglrun demo --start-guilog=/path-to-the-ggcdir/samples/test.log
   ```

   Where: the `demo` program is run with the option to start recording a log file and the output is saved to the `Ghost Client samples` directory.

**To record a log from the Genero Desktop Client (GDC) end**

2. Start the GDC client in admin and debug mode:

   ```
   cd path-to-the-gdcdir/bin; gdc.exe -a -D
   ```

   Where: the `-a` option starts the GDC in administration mode and `-D` starts debug mode (debug Tree and debug Console are active).

   The GDC opens.

3. To start recording to the log file, in the **Debug** panel first select a log file to store the scenario. Then, click on the **Record** button to start recording.
4. After starting recording to the log file, start the application you wish to record.

```
cd $FGLDIR/demo; fglrun demo
```

The `gwc-demo` demo program is opened by the GDC.

5. To stop recording to the log file, click on the **Stop** button.

When you have completed recording the log, your next task is to generate test sets from the recorded log file. See **Generate test sets from log file** on page 335.

### Generate test sets from log file

Generate test sets from recorded log files.

In the syntax of the following command line example it is assumed that the GGC package was unzipped in the same directory where you are running the *Launcher*, and that your environment is configured to run Java and the GGC, see **Configure your environment for Ghost Client** on page 317.

If you want to generate Java and BDL test sets from a log file that has previously been recorded from either the Genero Desktop Client or the Genero Web Client, run the command in example one for Java and example 2 for BDL.

1. `java com.fourjs.ggc.Launcher -w path/to/generate/myapp_tests -l path/to/myapp.log`

   This example is running the *Launcher* with the write `-w` option to generate a Java test set from a log file. The `-l` option is used followed by the path to the log file.

2. `java com.fourjs.ggc.Launcher -w path/to/generate/myapp_tests -l path/to/myapp.log --BDL`

   This example is basically the same instruction but with the addition of the `--BDL` option at the end to specify that a `4gl` file set, instead of Java files test set, is to be written.

   The samples directory in your GGC installation contains a set of recorded log files that can be used to automatically generate valid test sets in Java or BDL, see **Generate tests from Sample log files** on page 330.

### Automatic discovery of User Agent (adua.xrd)

`adua.xrd` is the configuration file used by the Genero Application Server (GAS) to determine which Output Map to use, based on the User Agent that submits the request for a specific application.

The `adua.xrd` file is located in the `$FGLASDIR/etc` directory.

**Tip:** Under most circumstances, modification of this file is not necessary.

- What is an Output Map? on page 336
- How an Output Map is chosen on page 336
- Modify the adua.xrd file to specify custom Output Maps on page 337
- Specify the Output Map in the application URI on page 337
- ADUA Syntax Diagrams on page 338
- adua.xrd usage example on page 339
What is an Output Map?

When the application server needs to render an application, it relies on the application having one or more MAP components defined in its configuration. Each MAP component specifies a RENDERING engine to be used. These MAP elements are child elements of the OUTPUT element, thus the name "Output Map".

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

How an Output Map is chosen

The Output Map chosen for an application is based on the detected browser type, as specified in the adua.xrd file.

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

The Genero Application Server first identifies the value of the RULE element for the application. The RULE element is defined in (or inherited by) the application configuration.

For example, if the value of the RULE element is UseGWC, then drop into that element. Once inside that element, the type of browser being used to display the application determines which Output Map is used to render the application.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RULE_LIST PUBLIC "-//Four Js//DTD Gas 2.50 Rule List//EN" "http://www.4js.com/ns/gas/2.50/rule.dtd">
<!-- Output Driver Determination (XRD - XML Rule Definition) -->
<RULE Id="UseGDC">
  <TABLE Id="1" Key="User-Agent">
    <ROW>
      <IN>MSIE</IN>
      <ACTION Type="RESULT">DUA_GDC</ACTION>
    </ROW>
    <ROW>
      <IN>GDC</IN>
      <ACTION Type="RESULT">DUA_GDC</ACTION>
    </ROW>
  </TABLE>
</RULE>
```
Based on this sample adua.xrd file:

If the RULE is UseGDC:
- For MSIE or GDC, the DUA_GDC Output Map is chosen.

If the RULE is UseGWC (for most desktop browsers):
- For the GDC, the DUA_GDC Output Map is chosen.
- Otherwise the DUA_HTML5 Output Map is chosen.

Modify the adua.xrd file to specify custom Output Maps

If you create a custom Output Map, you can modify the adua.xrd file to reference your new Output Map.

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

For example, you create an Output Map named DUA_HTML5_CUSTOM. You would modify the adua.xrd file and identify those browsers where that application should use your custom Output Map. The decision is still based on the browser type that is displaying the application.

```
...  
  <ACTION Type="RESULT">DUA_HTML5_CUSTOM</ACTION>
  ...
```

Specify the Output Map in the application URI

You can force an application to use a specific Output Map by providing the Output Map as an argument in the application URL.

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.
ADUA Syntax Diagrams

Define valid syntax for the `auda.xrd` file.

- **RULE_LIST**
- **RULE**
- **TABLE**
- **ROW**

**Syntax**

```
<RULE_LIST>
  <RULE Id="useId">
    <TABLE Id="numId" Key="keyType"/>
    <ROW>
      <IN>inType</IN>
      <OUT>outType</OUT>
      <ACTION Type="actionType">actionName</ACTION>
    </ROW>
    [...]
  </TABLE>
  <RULE> [...]
</RULE_LIST>
```

**Example**

```
<RULE_LIST>
  <RULE Id="UseGDC">
    <TABLE Id="1" Key="User-Agent">
      <ROW>
        <IN>MSIE</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
      <ROW>
        <IN>GDC</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
    </TABLE>
  </RULE>
</RULE_LIST>
```

- **RULE_LIST** on page 338
- **RULE** on page 338
- **TABLE** on page 339
- **ROW** on page 339

**RULE_LIST**
The `RULE_LIST` element is the main element of an XRD (XML Rule Definition) used by the GAS.

The `RULE_LIST` element contains the following child element:

1. One or more **RULE** elements.

**RULE**
The `RULE` element defines a unique rule.

The `RULE` element must specify an `Id` attribute; this required attribute takes a string value. The identifier (Id) of the rule defines its name, as it is going to be used later, in files such as the GAS configuration file.

Valid values for the `Id` attribute include:
• UseGDC
• UseGWC
• UseGWCMobile (deprecated)

The RULE element contains the following child element:

1. One or more TABLE elements. Each rule uses tables, which can be linked in order to have a complete process.

TABLE

The TABLE element must specify two attributes, an Id attribute and a Key attribute.

• The required Id attribute takes a string value. This attribute provides the table with a unique identifier (Id), which is necessary for linking tables.
• The required Key attribute takes a string value (NMToken). The Key attribute defines what is going to be analyzed. Currently, only two values are supported: Accept and User-Agent.

The TABLE element contains the following child element:

1. One or more ROW elements. Each table contains one or more rows. Rows are processed sequentially in order of appearance in the XRD file; therefore rows are not named.

ROW

The ROW element contains the required ACTION element, along with the optional IN and OUT elements.

The ROW element may contain the following child elements:

1. Zero or more IN elements (optional). The IN element takes a string value, and specifies a string or substring that must be in the HTTP header referenced by the TABLE key attribute.
2. Zero or more OUT elements (optional). The OUT element takes a string value, and specifies a string or substring that must not be in the HTTP header referenced by the TABLE key attribute (they must be OUT).
3. One ACTION element (required). The ACTION element must specify a Type attribute and takes a required string (NMToken) value. If the string matches the IN and OUT rules (i.e., the IN and OUT conditions are met), this element defines the action to perform. Valid values for this element type are:
   • GOTO_TABLE - Jumps to the specified table.
   • RESULT - Sends the result.

adua.xrd usage example

This topic explains an automatic user agent discovery configuration example.

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

Suppose you want to use the DUA_GDC output map for Internet Explorer browsers and DUA_XXX for all other user agents. To achieve this, you have to configure your application to support both Output Maps.

In your GAS configuration file (as.xcf), add an OUTPUT element with a rule identified by UseAllOutputDriver, defining the rendering and theme for two MAP elements identified by the DUA_GDC and DUA_XXX names:

```
<APPLICATION Id="test" Parent="defaultwa">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```
In the adua.xrd file, define the UseAllOutputDriver rule to associate a User-Agent type to a MAP element defined in the as.xcf file:

```
<RULE Id="UseAllOutputDriver">
  <TABLE Id="1" Key="User-Agent">
    <ROW>
      <IN>MSIE</IN>
      <ACTION Type="RESULT">DUA_GDC</ACTION>
    </ROW>
    <ROW>
      <ACTION Type="RESULT">DUA_XXX</ACTION>
    </ROW>
  </TABLE>
</RULE>
```

With this example, if the User-Agent value contains MSIE, GDCAX will be used (identified by DUA_GDC); otherwise the mapping that corresponds to the DUA_XXX will be used.

## GAS Predefined resources

Predefined resources for the Genero Application Server can be general or front-end specific.

- GAS predefined resources overview on page 340
- Common GAS predefined resources on page 341

### GAS predefined resources overview

This topic describes the syntax used in the GAS configuration file for predefined resources (i.e. variables). While most resources are defined in the GAS configuration file, predefined resources can not be explicitly defined as they are automatically replaced depending on the path of the application you are executing. Therefore, in order that they are available for use, they are predefined as resources using an XML tagging mechanism, which is described in this topic.

#### Syntax 1

```
$(resource-name)
```

1. resource-name is the name of a resource defined in the GAS configuration file.

#### Syntax 2

```
<tag gwc:tpl-attribute="tpl-value" [...]>...</tag>
```

1. tag is an HTML tag.
2. tpl-attribute is an attribute.
3. **tpl-value** is the value of the attribute.

**Usage example**

For example, we can define the path to an application's web component directory as follows:

```xml
<WEB_COMPONENT_DIRECTORY>${application.path}/webcomponents</WEB_COMPONENT_DIRECTORY>
```

**Common GAS predefined resources**

These Genero Application Server predefined resources are available for any front-end.

These resources include:

**Table 66: Common GAS predefined resources**

<table>
<thead>
<tr>
<th>Predefined Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application.id</td>
<td>Application identifier in as.xcf or, in the case where an external application configuration file is used, the name of the .xcf file.</td>
</tr>
<tr>
<td>application.group</td>
<td>Application Group name.</td>
</tr>
<tr>
<td>application.name</td>
<td>Application name.</td>
</tr>
<tr>
<td>application.path</td>
<td>Handles the path of the application (the path defined in the /APPLICATION/EXECUTION/PATH element). For example, we can define the resource domain &quot;Image&quot; as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;WEB_APPLICATION_PICTURE_COMPONENT Id=&quot;cpn.gwc.picture&quot;&gt;...&lt;PATH Id=&quot;Image&quot; Type=&quot;APPSERVER&quot;&gt;$(res.path.pic);${application.path}&lt;/PATH&gt;...&lt;/WEB_APPLICATION_PICTURE_COMPONENT&gt;</code></td>
</tr>
<tr>
<td>connector.uri</td>
<td>Will be replaced by the URL path to the connector. For more information on application URIs See Table 6: Explanation of URI syntax options on page 52.</td>
</tr>
<tr>
<td></td>
<td>For example, if the URL is:</td>
</tr>
<tr>
<td></td>
<td><a href="http://localhost/gas/ua/r/demo?Arg=val1&amp;Arg=val2">http://localhost/gas/ua/r/demo?Arg=val1&amp;Arg=val2</a></td>
</tr>
<tr>
<td></td>
<td>The value of connector.uri will be /gas.</td>
</tr>
<tr>
<td>server.version</td>
<td>Will be replaced by GAS software version.</td>
</tr>
<tr>
<td>configuration.filepath</td>
<td>The absolute path of the configuration file used to start the GAS.</td>
</tr>
</tbody>
</table>

**GAS Configuration Reference**

These topics provide reference details for Genero Application Server configuration.

- GAS configuration file on page 342
- Application configuration files on page 342
GAS configuration file
The Genero Application Server is configured through a configuration file. The default configuration file is `as.xcf`, located in the `$FGLASDIR/etc` directory.

To create a new Genero Application Server configuration file, create a copy of the default and make modifications in the copy. Use the `-f` option to specify which configuration file to use when starting the Genero Application Server. If you do not specify a specific file, the default (`as.xcf`) is used.

The configuration file is an XML file. The root element in the Application Server configuration file is the `CONFIGURATION` element. A configuration file will have one `CONFIGURATION` element which serves as the global configuration element. There are no attributes available for the `CONFIGURATION` element. The `CONFIGURATION` element contains a single child element, the `APPLICATION_SERVER` element. The Application Server configuration starts with this element. See GAS configuration file hierarchy on page 343 for a full list of valid elements and their place in the file hierarchy.

When you make a change to the GAS configuration file, you must restart the dispatcher before the changes take effect. Depending on the dispatcher, you may or may not have the applications (e.g. proxies) stopped; some of the dispatchers can recover the application after the restart and continue where the application left off. For more information on restarting the GAS dispatcher, refer to the dispatcher-specific help topics:

- Restarting the standalone dispatcher
- Restarting the ISAPI dispatcher
- Restarting the FastCgi dispatcher

Application configuration files
Configure Genero Web applications and services using a subset of the elements provided for the Genero Application Server configuration.

The application configuration details can either be:

- added to the Genero Application Server configuration file.
- provided in application-specific configuration files. Each application would have its own configuration file, written in XML.

Elements in an application’s configuration
The top-most element for application configuration is the `APPLICATION` element. The elements allowed, and how they are used, depend on whether the application is a web application (GWC) or a web service (GWS). For a full list of valid elements and their place in the file hierarchy, see the appropriate topic:

- GWC configuration file hierarchy on page 347
- GWS configuration file hierarchy on page 348

Use application-specific configuration files
When you add application configuration details to the Genero Application Server configuration file, you must restart the dispatcher before the changes take effect. You must also take care during upgrades, that the GAS configuration file is not overwritten and you lose your changes.

When you create a new application configuration file, or make changes to an existing application configuration file, you do not need to restart any dispatchers. All applications started after you save your changes use the changed settings. The files are easily archived apart from the GAS configuration file, and not overwritten by newer versions during upgrades.

For these reasons, it is recommended that you use application-specific configuration files.
Configuration file hierarchies

Configuration files define the initial configuration settings for the GAS and for individual applications. Each configuration file is made up of elements in a predefined hierarchy.

- GAS configuration file hierarchy on page 343
- GWC configuration file hierarchy on page 347
- GWS configuration file hierarchy on page 348

GAS configuration file hierarchy

A listing of elements valid in the Genero Application Server configuration file, shown hierarchically.

Select an element name to be taken to the topic discussing that element.

- CONFIGURATION
  - APPLICATION_SERVER
    - RESOURCE_LIST
      - PLATFORM_INDEPENDENT
        - RESOURCE
      - WNT
        - RESOURCE
      - UNX
        - RESOURCE
    - COMPONENT_LIST
      - WEB_APPLICATION_EXECUTION_COMPONENT
        - ENVIRONMENT_VARIABLE
        - PATH
        - DVM
        - WEB_COMPONENT_DIRECTORY on page 422
      - MODULE
      - PARAMETERS
        - PARAMETER
      - ACCESS_CONTROL
        - ALLOW_FROM
      - DELEGATE
    - SERVICE_APPLICATION_EXECUTION_COMPONENT
      - ENVIRONMENT_VARIABLE
      - PATH
      - DVM
      - MODULE
      - PARAMETERS
        - PARAMETER
      - ACCESS_CONTROL
        - ALLOW_FROM
      - DELEGATE
      - POOL
        - START
        - MIN_AVAILABLE
        - MAX_AVAILABLE
• MAX_REQUESTS_PER_DVM
• WEB_APPLICATION_TIMEOUT_COMPONENT
  • USER_AGENT
  • REQUEST_RESULT (for an application) on page 399
  • DVM_AVAILABLE
  • DVM_PINGTIMEOUT
• AUTO_LOGOUT_COMPONENT on page 359
  • TIMEOUT (for auto logout) on page 413
  • COMMAND (for auto logout) on page 364
• SERVICE_APPLICATION_TIMEOUT_COMPONENT
  • DVM_AVAILABLE
  • KEEP_ALIVE
  • REQUEST_RESULT (for a service) on page 399
• WEB_APPLICATION_PICTURE_COMPONENT
  • PATH
• WEB_APPLICATION_RENDERING_COMPONENT
  • OUTPUT_DRIVER
  • XML_DECLARATION
  • HTTP_RESPONSE_ENCODING
  • HTTP_REQUEST_ENCODING
  • MIME_TYPE
  • DOC_TYPE
    • NAME
    • EXTERNAL_ID
    • SYSTEM_ID
• WEB_APPLICATION_HTTP_COOKIES_COMPONENT
  • HTTP_COOKIE
    • CONSTANT
    • VARIABLE
• WEB_APPLICATION_THEME_COMPONENT
  • BOOTSTRAP
  • TEMPLATE
  • SHORTCUT
  • SNIPPET
• INTERFACE_TOCONNECTOR
  • ROOT_URL_PREFIX on page 402
  • TCP_BASE_PORT
  • TCP_PORT_OFFSET
  • DOCUMENT_ROOT
  • GWC_JS_LOOKUP_PATH on page 376
  • TEMPORARY_DIRECTORY
  • SESSION_DIRECTORY
  • REPORT_VIEWER_DIRECTORY on page 398
  • SOCKET_FAMILY
  • SOCKET_PATH
  • HTTP on page 378
• SESSION_COOKIE on page 406
• APPLICATION (for HTTP) on page 355
  • HEADER on page 377
  • SERVICE (for HTTP) on page 404
  • HEADER on page 377
• INTERFACE_TO_DVM
• ADDRESS
• LOG
  • OUTPUT
  • FORMAT
  • CATEGORIES_FILTER
  • RAW_DATA
• MONITOR
  • ALLOW_FROM
• FILE_TRANSFER
  • TIMEOUT
• APPLICATION_LIST
• GROUP
• APPLICATION
  • DESCRIPTION
    • SHORT
    • LONG
  • RESOURCE
  • EXECUTION
    • ENVIRONMENT_VARIABLE
    • PATH
    • DVM
    • MODULE
    • PARAMETERS
      • PARAMETER
      • ACCESS_CONTROL
        • ALLOW_FROM
        • DELEGATE
        • WEB_COMPONENT_DIRECTORY on page 422
• UA_OUTPUT
  • PROXY
  • PUBLIC_IMAGEPATH
  • GWC-JS on page 374
• TIMEOUT
  • USER_AGENT
  • REQUEST_RESULT
  • DVM_AVAILABLE
  • DVM_PINGTIMEOUT
• OUTPUT
- HTTP_HEADER
- HTTP_COOKIES
  - HTTP_COOKIE
    - CONSTANT
    - VARIABLE
- MAP
  - PROXY
  - PICTURE
  - PATH
- TIMEOUT
  - USER_AGENT
  - REQUEST_RESULT
  - DVM_AVAILABLE
  - DVM_PINGTIMEOUT
- RENDERING
  - OUTPUT_DRIVER
  - XML_DECLARATION
  - HTTP_RESPONSE_ENCODING
  - HTTP_REQUEST_ENCODING
  - MIME_TYPE
  - DOC_TYPE
    - NAME
    - EXTERNAL_ID
    - SYSTEM_ID
- THEME
  - BOOTSTRAP
  - TEMPLATE
  - SHORTCUT
  - SNIPPET
- SERVICE_LIST
- GROUP
- APPLICATION
  - DESCRIPTION
  - SHORT
  - LONG
  - RESOURCE
  - PROXY
  - EXECUTION
    - ENVIRONMENT_VARIABLE
  - PATH
  - DVM
  - MODULE
  - WEB_COMPONENT_DIRECTORY on page 422
- PARAMETERS
  - PARAMETER
GWC configuration file hierarchy
A listing of the elements valid for a Genero Web Client (GWC) application configuration file, shown hierarchically.

This is a complete listing of all of the elements available for a Genero Web Client external application configuration file. It is a subset of the elements available for the Genero Application Server configuration file. The **APPLICATION** element is the root element for that application's configuration. For a Web application, the **APPLICATION** element is the child of the **APPLICATION_LIST** element in the Genero Application Server configuration file.

The elements are described in detail; select an element name to be taken to the topic discussing that element.

- **APPLICATION**
  - **DESCRIPTION**
    - **SHORT**
    - **LONG**
  - **RESOURCE**
  - **EXECUTION**
    - **ENVIRONMENT_VARIABLE**
    - **PATH**
    - **DVM**
    - **MODULE**
    - **PARAMETERS**
      - **PARAMETER**
      - **ACCESS_CONTROL**
        - **ALLOW_FROM**
        - **DELEGATE**
        - **WEB_COMPONENT_DIRECTORY** on page 422
  - **UA_OUTPUT**
    - **PROXY**
    - **PUBLIC_IMAGEPATH**
    - **GWC-JS** on page 374
  - **TIMEOUT**
    - **USER_AGENT**
    - **REQUEST_RESULT**
    - **DVM_AVAILABLE**
    - **DVM_PINGTIMEOUT**
GWS configuration file hierarchy
A listing of available elements valid for a Genero Web Services (GWS) application configuration file, shown hierarchically.

This is a complete listing of all of the elements available for a Genero Web Services external application configuration file. It is a subset of the elements available for the Genero Application Server configuration file. The APPLICATION element is the root element for that application's configuration. For a Web service, the APPLICATION element is the child of the SERVICE_LIST element in the Genero Application Server configuration file.

The elements are described in detail; select an element name to be taken to the topic discussing that element.

- APPLICATION
  - DESCRIPTION
    - SHORT
    - LONG
  - RESOURCE
• PROXY
• EXECUTION
  • ENVIRONMENT_VARIABLE
  • PATH
  • DVM
  • MODULE
  • WEB_COMPONENT_DIRECTORY on page 422
• PARAMETERS
  • PARAMETER
• ACCESS_CONTROL
  • ALLOW_FROM
• DELEGATE
• POOL
  • START
  • MIN_AVAILABLE
  • MAX_AVAILABLE
  • MAX_REQUESTS_PER_DVM
• TIMEOUT
  • DVM_AVAILABLE
  • KEEP_ALIVE

Configuration file elements
An alphabetical listing of the elements used across the Genero Application Server configuration files.

• ACCESS_CONTROL on page 351
• ADDRESS on page 353
• ALLOW_FROM on page 353
• APPLICATION (for an application) on page 354
• APPLICATION (for HTTP) on page 355
• APPLICATION (for a service) on page 356
• APPLICATION_LIST on page 356
• APPLICATION_SERVER on page 357
• AUTO_LOGOUT on page 358
• AUTO_LOGOUT_COMPONENT on page 359
• BOOTSTRAP (GWC-HTML5) on page 360
• CATEGORIES_FILTER on page 361
• COMPONENT_LIST on page 363
• COMMAND (for auto logout) on page 364
• CONFIGURATION on page 364
• CONSTANT on page 365
• DELEGATE on page 365
• DESCRIPTION on page 366
• DOC_TYPE on page 366
• DOCUMENT_ROOT on page 367
• DVM_AVAILABLE on page 368
• DVM_PINGTIMEOUT on page 368
• DVM on page 369
• ENVIRONMENT_VARIABLE on page 369
• EXECUTION (for an application) on page 369
• EXECUTION (for a service) on page 371
• EXTERNAL_ID on page 371
• FILE_TRANSFER on page 372
• FORMAT on page 372
• GROUP (for an application) on page 373
• GROUP (for a service) on page 374
• GWC-JS on page 374
• GWC_JS_LOOKUP_PATH on page 376
• HEADER on page 377
• HTTP on page 378
• HTTP_COOKIE on page 378
• HTTP_COOKIES on page 379
• HTTP_HEADER on page 380
• HTTP_REQUEST_ENCODING on page 380
• HTTP_RESPONSE_ENCODING on page 381
• INTERFACE_TO_CONNECTOR on page 381
• INTERFACE_TO_DVM on page 382
• KEEP_ALIVE on page 383
• LONG on page 383
• LOG on page 383
• MAP on page 384
• MAX_AVAILABLE on page 385
• MAX_REQUESTS_PER_DVM on page 385
• MONITOR on page 385
• MIME_TYPE on page 386
• MIN_AVAILABLE on page 387
• MODULE on page 387
• NAME on page 387
• OUTPUT (under LOG) on page 388
• OUTPUT (under APPLICATION) on page 388
• OUTPUT_DRIVER on page 389
• PARAMETERS on page 389
• PATH (under EXECUTION) on page 390
• PATH (under PICTURE) on page 390
• PATH with Type WEBSERVER on page 390
• PATH with Type APPSERVER on page 391
• PICTURE on page 392
• PLATFORM_INDEPENDENT on page 395
• POOL on page 395
• PROXY (for an application) on page 396
• PROXY (for a service) on page 396
• RAW_DATA on page 397
• RENDERING on page 398
• REPORT_VIEWER_DIRECTORY on page 398
• REQUEST_RESULT (for an application) on page 399
• REQUEST_RESULT (for a service) on page 399
• RESOURCE on page 400
• RESOURCE (for a service) on page 401
• RESOURCE (for an application) on page 401
• RESOURCE_LIST on page 401
The **ACCESS_CONTROL** element specifies access from a list of IP allowed to access applications or services.

Access can be globally denied or allowed by keywords (NOBODY, ALL).

By default, an application or a service is not accessible by anyone. It needs to be explicitly configured with the **ALLOW_FROM** element.

**Syntax**

```xml
<ACCESS_CONTROL>
  [<ALLOW_FROM>"NOBODY|ALL|ip_address"]</ALLOW_FROM>[[...]]
</ACCESS_CONTROL>
```

The *ip_address* is a valid IPv4 or IPv6 address. For IPv4 it can be a complete IP address or a network address (ending with a dot).
**Important:** Depending on the network configuration, it is not always possible to get the actual client IP address. If there is a proxy server between the client and the server, for example, the client IP address seen by the GAS may be the address from the proxy server.

**Child elements**
- Zero or more `ALLOW_FROM` on page 353 elements.

**Example**

```xml
<ACCESS_CONTROL>
  <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
  <ALLOW_FROM>192.168.1.1</ALLOW_FROM>
  <ALLOW_FROM>192.168.2.1</ALLOW_FROM>
  <ALLOW_FROM>192.168.3.1</ALLOW_FROM>
  <ALLOW_FROM>192.168.4.1</ALLOW_FROM>
</ACCESS_CONTROL>
```

In this example, an application or a service is reachable from the localhost (127.0.0.1), and all IP addresses that begin with "192.168.". The consecutive colons (::) notation in "fdbd:2768:c176:1::323a" shows an example of a collapsed IPv6 address, where the colons represent four successive 16-bit blocks that contain zeros.

**Example configuring access control for demo applications**

The default deployment of the demo application is specified by the resource `res.access.control`, which is defined with the value `NOBODY` by default.

**Note:** Access control rules will be ignored by the standalone dispatcher (httpdispatch).

**Important:** The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.

To allow access from the localhost, in the GAS configuration file (default `FGLASDIR/etc/as.xcf`) you need to change the application element for `gwc-demo` from:

```xml
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```

To:

```xml
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```
**ADDRESS**

The ADDRESS element specifies the name or IP address of the machine where the Dynamic Virtual Machine (DVM) runs.

*The ADDRESS element specifies* the name or IP address of the machine where the Genero Application Server runs. This address is used by the DVM to set (build) the FGLSERVER environment variable.

**Usage examples**

```xml
<Address>app_server</Address>
<Address>192.127.45.17</Address>
<Address>zeus</Address>
```

**Note:** It is not recommended to use localhost or 127.0.0.1 because the license server requires the real address of the machine to check the licenses, and if your machine is not well configured, a bad address is returned to the license server that will then refuse to start a new DVM.

**Parent elements**

This element is a child of one of the following elements: SERVICE_APPLICATION_EXECUTION_COMPONENT on page 402

- WEB_APPLICATION_EXECUTION_COMPONENT on page 417
- EXECUTION (for an application) on page 369
- EXECUTION (for a service) on page 371

**ALLOW_FROM**

The ALLOW_FROM element specifies rules for access to GAS applications or services from specific hosts.

This can be specified by a host's IP address, or access can either be globally denied or allowed by the following keywords:

- NOBODY
- ALL

**Syntax**

\[<ALLOW_FROM>"NOBODY\|ALL\|ip_address"</ALLOW_FROM>\]

The ip_address is a valid IPv4 or IPv6 address. For IPv4 it can be a complete IP address or a network address (ending with a dot).

**Example usage**

```xml
<ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
```

The resource res.access.control is defined with the value NOBODY by default. The default GAS deployment for demo applications and MONITOR on page 385 references this resource. For more information on configuring access control, see Example configuring access control for demo applications on page 352.

**Note:** Access control rules will be ignored by the standalone dispatcher (httpdispatch).
Important: The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.

How access control rules are applied

If more than one access control rule (i.e. ALLOW_FROM element) is specified, the following describes the order rules are applied:

1. NOBODY always gets the highest priority regardless of other rules, which means that nobody gets access.
2. ALL gets priority over specific IP addresses when the keyword NOBODY is not provided, which means that access is allowed to all.
3. If neither the keyword NOBODY nor ALL is provided, the remaining access control rules are applied.

Parent elements

This element is a child of one of the following elements:

- ACCESS_CONTROL on page 351
- MONITOR on page 385

APPLICATION (for an application)

This APPLICATION element defines an application within the Genero Application Server configuration file or in an external application configuration file.

An APPLICATION element defines an application. Attributes for this element include:

- Id (required for applications defined within the Genero Application Server configuration file. For external configuration files, the Id attribute is ignored) - A string used to uniquely identify this application configuration element. The Id specified is compared to the application name in the request.
- Parent - A string that identifies the parent application, or the application from which this application will inherit its default configuration/settings.
- Abstract - Defines whether this application configuration element is an abstract application. It expects a boolean string; the valid values for this type are "TRUE" and "FALSE". An abstract application cannot instantiate Virtual Machines. Abstract configurations are used purely in the scope of future inheritance of the configuration for other Web applications. Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.
- mode - When set to "sticky", defines a Web service as a sticky Web service. See Configure sticky Web services on page 254.

Child elements

The APPLICATION element may contain the following elements:

1. Zero or one DESCRIPTION element.
2. Zero or more RESOURCE elements.
3. Zero or one EXECUTION elements.
4. Zero or one UA_OUTPUT on page 414 elements. This element is used for all UI applications. Use OUTPUT for GWC for HTML5 applications.
5. Zero or one OUTPUT elements. This element is deprecated; only used for UI applications using GWC for HTML5.

Caution: Use either UA_OUTPUT or OUTPUT, depending on which front-end your application is intended for; do not use both. Starting with Genero 3.00, all UI applications use UA_OUTPUT.
Example

```xml
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
  <UA_OUTPUT> </UA_OUTPUT>
</APPLICATION>
```

For more information, see Configuring applications on GAS on page 100.

Parent elements

When used in an application configuration file, it is the top-most element.

When used in the GAS configuration file, this element is a child of one of the following elements: APPLICATION_LIST on page 356.

APPLICATION (for HTTP)

This element contains a list of those HEADER elements defining the communication carried on the HTTP protocol between Web applications and the client.

Starting with GAS 3.00 you can set custom HTTP headers for Web applications and Web services. This configuration takes place in the HTTP element of the INTERFACE_TO_CONNECTOR element of the GAS configuration file.

Syntax

```xml
<APPLICATION>
  [<HEADER Name="headerName">value[; ...]</HEADER> | [...]]
</APPLICATION>
```

Child elements

The APPLICATION element may contain zero to many HEADER on page 377 elements.

Example

```xml
<INTERFACE_TO_CONNECTOR>
  <HTTP>
    <SESSION_COOKIE Secure="FALSE"> </SESSION_COOKIE>
    <APPLICATION>
      <HEADER Name="X-XSS-Protection">1; mode=block</HEADER>
      <HEADER Name="X-Content-Type-Options">nosniff</HEADER>
      <HEADER Name="X-Frame-Options">SAMEORIGIN</HEADER>
    </APPLICATION>
    <SERVICE>
      <HEADER/>
    </SERVICE>
  </HTTP>
</INTERFACE_TO_CONNECTOR>
```

Parent elements

This element is a child of the HTTP on page 378 element.
APPLICATION (for a service)

An APPLICATION element defines an application. For each Web Service you wish to make accessible through the Genero Application Server, you must create an APPLICATION element. Attributes for this element include:

- **Id** (required for Web services applications defined within the Genero Application Server configuration file; optional for applications defined in an external application configuration file) - A string to uniquely identify this Web service application configuration element. The Id specified is compared to the application name in the request.
- **Parent** - A string that identifies the parent application, or the application from which this application will inherit its default configuration/settings.
- **Abstract** - Defines whether this application configuration element is an abstract application. It expects a boolean string; the valid values for this type are "TRUE" and "FALSE". An Abstract application can not instantiate Virtual Machines. Abstract configurations are used purely in the scope of future inheritance of the configuration for other Web services applications. Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.
- **mode** - When set to "sticky", defines a Web service as a sticky Web service.

**Note**: With the release of Genero 2.0, Web services are named applications as they host several Web services in one DVM.

Child elements

When you define a Web service application, you can specify the following elements:

1. Zero or one DESCRIPTION element.
2. Zero or more RESOURCE elements.
3. Zero or one EXECUTION element.
4. Zero or more TIMEOUT elements.

Example

```xml
<Application Id="webapp" Parent="abswebapp">
  <Execution>
    <Path>${res.path.fgldir.demo}</Path>
    <Module>webapp.42r</Module>
  </Execution>
  <Timeout />
</Application>
```

Parent elements

When used in an application configuration file, it is the top-most element.

When used in the GAS configuration file, this element is a child of one of the following elements:

SERVICE_LIST on page 405

APPLICATION_LIST

The APPLICATION_LIST element provides a list of default groups and abstract application configurations.

This element is defined within the Genero Application Server configuration file.

For each application to be serviced by the Genero Application Server, you must provide the details for that application in either the Genero Application Server configuration file or in an external application configuration file (xcf).

For information about the general process of defining applications and groups, refer to the Configuring applications on GAS on page 100 section of this manual.
Syntax

```
<APPLICATION_LIST>
  [ <GROUP>...] </GROUP>
  [ <APPLICATION>...] </APPLICATION>
</APPLICATION_LIST>
```

Child elements

The `APPLICATION_LIST` element may contain the following child elements:

1. Zero or more `GROUP` elements.
2. Zero or more `APPLICATION` elements.

Example

```
<APPLICATION_LIST>
  <GROUP Id="appgroup"/>
  <APPLICATION Id="gwc-demo" Parent="defaultgwc">
    <EXECUTION>
      <PATH>$(res.path.fgldir.demo)</PATH>
      <MODULE>demo.42r</MODULE>
    </EXECUTION>
  </APPLICATION>
</APPLICATION_LIST>
```

Parent elements

This element is a child of the `APPLICATION_SERVER` on page 357 element.

`APPLICATION_SERVER`

This element acts as a parent container for all Genero Application Server configuration elements.

The `APPLICATION_SERVER` element does not support any attributes.

Syntax

```
<APPLICATION_SERVER>
  <RESOURCE_LIST>...</RESOURCE_LIST>
  <COMPONENT_LIST>...</COMPONENT_LIST>
  <INTERFACE_TO_CONNECTOR>...</INTERFACE_TO_CONNECTOR>
  <INTERFACE_TO_DVM>...</INTERFACE_TO_DVM>
  <LOG>...</LOG>
  <MONITOR>...</MONITOR>
  <FILE_TRANSFER>...</FILE_TRANSFER>
  <APPLICATION_LIST>...</APPLICATION_LIST>
  <SERVICE_LIST>...</SERVICE_LIST>
</APPLICATION_SERVER>
```

Child elements

The `APPLICATION_SERVER` element contains the following child elements:

- One `RESOURCE_LIST` element, containing a list of resources.
- One `COMPONENT_LIST` element, containing a list of components.
- One `INTERFACE_TO_CONNECTOR` element, specifying the interface between the Genero Application Server (GAS) and the GAS Connector. The connector is either the ISAPI, FastCGI, or J2EE extension, or the user agent through direct connection.
- One `INTERFACE_TO_DVM` element, specifying the interface to the Dynamic Virtual Machine.
• Zero or more **LOG** elements, specifying the type of information that is logged and where it is logged to.
• Zero or one **MONITOR** element, specifies from which machines the monitor URL is accessible.
• Zero or more **FILE_TRANSFER** elements, specifying the directory where files are stored while being transferred between the front-end machine and the DVM.
• Zero or one **APPLICATION_LIST** element, containing a list of applications.
• Zero or one **SERVICE_LIST** element, containing a list of Web Services.

**Example**

```xml
<CONFIGURATION
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfas.xsd">
    <APPLICATION_SERVER>
        ...
    </APPLICATION_SERVER>
</CONFIGURATION>
```

**Parent elements**

This element is a child of one of the following elements: [CONFIGURATION](#) on page 364

**AUTO_LOGOUT**

Selects the **AUTO_LOGOUT** mechanism to be used for an application. It takes the attribute **Using**, where you can reference a predefined **AUTO_LOGOUT_COMPONENT** on page 359 (to inherit the auto logout parameters of that component).

If the **AUTO_LOGOUT** is set, the application will get a log out message after the time in seconds specified by the **TIMEOUT** (for auto logout) on page 413 element has elapsed.

• For Genero Web Client for JavaScript (GWC-JS) applications, you will get a log out page.
• For GDC applications, you will get a pop-up window.

By default the ending page or pop-up window shows the following message when auto logout occurs: **You have been logged out.**

**Note:** The auto logout feature is not supported on the legacy Genero Web Client applications using the HTML5 theme.

**Syntax**

```xml
<AUTO_LOGOUT Using="auto_logout_component_Id"/>
</AUTO_LOGOUT>
```

**Usage example - using an auto logout component**

When your applications reference the default application **defaultwa**, they inherit its **AUTO_LOGOUT** settings, which use the default **AUTO_LOGOUT_COMPONENT** identified by the **Id** value "cpn.wa.autologout", as shown in the examples from the GAS configuration file.

```xml
<APPLICATION Id="defaultwa" Abstract="TRUE">
    <!-- This is the "default" application. It is not used directly: it is used for defining a "root" application. -->
    <EXECUTION Using="cpn.wa.execution.local"/>
    <AUTO_LOGOUT Using="cpn.wa.autologout"/>
    ...
</APPLICATION>
```

```xml
<COMPONENT_LIST>
```
Note: The default timeout duration set to 0 seconds means the auto logout is ignored and applications keep running.

Usage example - defining the auto logout directly
If you want to set auto logout directly in your application configuration file (xcf), you can add an AUTO_LOGOUT element.

Child elements
There are no child elements.

Parent elements
This element is a child of the following element: APPLICATION (for an application) on page 354

Auto logout and child applications
As long as the parent application or any of its child applications have user activity, the auto logout is not triggered.

Important: If an application has child applications running when the AUTO_LOGOUT is triggered, the following describes the logout process:

- An auto logout ending page or pop-up window will be generated and returned for all child applications.
- The proxy will stay alive until the last child application has shown its ending page before closing the fglrun application.

Note: The order that applications close can not be determined by the AUTO_LOGOUT process as it depends mainly on when the auto logout is initiated in the front-end. If the order your applications close at auto logout is important, you will need to handle this in your application's code.

AUTO_LOGOUT_COMPONENT
The AUTO_LOGOUT_COMPONENT element creates an application auto logout component, which defines a mechanism for triggering and handling auto-logout events. It takes an attribute Id, which specifies the unique identifier for a set of auto-logout definitions. It is this unique identifier that is referenced by an application, providing it with a base set of auto-logout values.
Syntax

```xml
<AUTO_LOGOUT_COMPONENT Id="component_unique_identifier">
  <TIMEOUT>timeoutSeconds</TIMEOUT>
  <!--COMMAND is an optional configuration element -->
  <COMMAND Timeout="timeoutSeconds">commandScript</COMMAND>
</AUTO_LOGOUT_COMPONENT>
```

Child elements

The `AUTO_LOGOUT` element may contain the following child elements:

1. Zero or one `TIMEOUT` (for auto logout) on page 413 element.
2. Zero or one `COMMAND` (for auto logout) on page 364 element (Optional).

Usage example default `AUTO_LOGOUT_COMPONENT` in the GAS configuration file

```xml
<AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
  <TIMEOUT>0</TIMEOUT>
</AUTO_LOGOUT_COMPONENT>
```

In this example, the `Id` value - `cpn.wa.autologout` - defines the default `AUTO_LOGOUT_COMPONENT` in the GAS configuration file. When your applications reference the default application, `defaultwa`, they inherit the settings defined by its `AUTO_LOGOUT` on page 358 element as shown in the example:

```xml
<AUTO_LOGOUT Using="cpn.wa.autologout"/>
```

Usage example 2

```xml
<AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
  <TIMEOUT>30</TIMEOUT>
  <COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND>
</AUTO_LOGOUT_COMPONENT>
```

In this example:

- The `TIMEOUT` element is set for 30 seconds. When no user activity is detected, the DVM waits for this timeout period to elapse before the auto logout task is performed.
- The `COMMAND` element's `Timeout` attribute is set to 20 seconds to allow a command to be run that checks if the auto logout is allowed, see `COMMAND` (for auto logout) on page 364

Parent elements

This element is a child of one of the following elements: `COMPONENT_LIST` on page 363.

**BOOTSTRAP (GWC-HTML5)**

Selects the bootstrap template to be used in this theme.

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the `wa` protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use Genero Browser Client, see *Genero Web Client for JavaScript (GWC-JS)* on page 184.
Syntax

<BOOTSTRAP Id="bootstrapId"> bootstrapPath </BOOTSTRAP>

Syntax notes

1. `bootstrapId` is the unique identifier for this element.
2. `bootstrapPath` is the path to the bootstrap template to be used in this theme.

Usage example

```
<WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.html5.gwc">
  <BOOTSTRAP Id="_default">$(res.path.tpl.html5)/bootstrap.xhtml</BOOTSTRAP>
  <TEMPLATE Id="_default">$(res.path.tpl.html5)/main.xhtml</TEMPLATE>
  <SNIPPET Id="UIFrame">$(res.path.tpl.html5)/UIFrame.xhtml</SNIPPET>
  ...
</WEB_APPLICATION_THEME_COMPONENT>
```

Parent elements

This element is a child of one of the following elements: `THEME` on page 411, `WEB_APPLICATION_THEME_COMPONENT` on page 420

CATEGORIES_FILTER

The `CATEGORIES_FILTER` element specifies the type of log messages to be captured.

This element is an optional element of the `LOG` element. If the `CATEGORIES_FILTER` element is omitted, no categorized messages are logged.

Syntax

```
<LOG>
  <OUTPUT></OUTPUT>
  <FORMAT></FORMAT>
  [ [<CATEGORIES_FILTER>filterName[...]</CATEGORIES_FILTER> ]
  <RAW_DATA></RAW_DATA>
</LOG>
```

Valid category filter names are shown in Table 67: Category filters on page 361.

Table 67: Category filters

<table>
<thead>
<tr>
<th>Filter Names</th>
<th>Description</th>
</tr>
</thead>
</table>
| DEBUG        | Log internal information for debugging.  
<pre><code>           | **Important**: Do not set unless requested by your support center. |
</code></pre>
<p>| WARNING      | Log warning messages. |
| ERROR        | Log error information. |
| MUTEX        | Log multithreading information. |
| SESSION      | Log session management information. |
| GAS          | Log information about the Genero Application Server version, build and package. |
| LOG          | Display log parameters. |</p>
<table>
<thead>
<tr>
<th>Filter Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGURATION</td>
<td>Log configuration management information.</td>
</tr>
<tr>
<td>DEPRECATED</td>
<td>Log warnings if a deprecated function is used in a template.</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Log Dynamic Virtual Machine data, error, any process handled by the Genero Application Server.</td>
</tr>
<tr>
<td>VM</td>
<td>Log communications with the Dynamic Virtual Machine.</td>
</tr>
<tr>
<td>FT</td>
<td>Log information about file transfer.</td>
</tr>
<tr>
<td>ACCESS</td>
<td>Log summary information about the requests handled by the server.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Log HTTP requests (interaction with the User Agent).</td>
</tr>
<tr>
<td>SOCKET</td>
<td>Log information concerning sockets (creation, connection, communication, and so on).</td>
</tr>
<tr>
<td>TASK</td>
<td>Extracts the time of all the things done by the GAS (a technical attribute).</td>
</tr>
<tr>
<td>TEMPLATE</td>
<td>Information about TEMPLATE template processing.</td>
</tr>
<tr>
<td>TIMER</td>
<td>Information about timers creation, destruction and expiration.</td>
</tr>
<tr>
<td>WA</td>
<td>Context of the Web applications.</td>
</tr>
<tr>
<td>ALL</td>
<td>Activate all categories except DEBUG.</td>
</tr>
</tbody>
</table>

**Syntax notes**

You can specify multiple category filter values by listing multiple filter names, separated by spaces.

**Usage example**

This example is valid for the `CATEGORIES_FILTER` element:

```
<CATEGORIES_FILTER>ERROR WARNING</CATEGORIES_FILTER>
```

This example generates error and warning messages in the logs.

**Generate a detailed daily log file**

If you encounter an issue, you can send a detailed log file to your support center. To create the detailed daily log file, use the ALL pseudo category as the category filter.

**Important:** Only use the default categories (GAS ACCESS PROCESS DEPRECATED ERROR WARNING) when in production. Categories such as VM or ALL are for debugging and should only be set for a short period of time, as they generate many log entries. Non-standard categories should be used with care. The support center will tell you when you should set ALL for your categories filter.

**Example - set category filter ALL for debugging**

```
<LOG>
  <OUTPUT Type="DAILYFILE">/work/tmp/gas</OUTPUT>
  <FORMAT>date time relative-time process-id thread-id contexts event-type event-params</FORMAT>
  <CATEGORIES_FILTER>ALL</CATEGORIES_FILTER>
</LOG>
```
This example generates various log files in the /work/tmp/gas directory, depending on the dispatcher and the application or service run.

Parent elements
This element is a child of the LOG on page 383 element.

COMPONENT_LIST
Within the Genero Application Server configuration file, you can define various application components. Components are sets of preset variables, and are used by applications that share common configurations. Within the COMPONENT_LIST element, you specify components to be available for use by applications. When defining an application, you can then reference the component by its unique Id.

Syntax

```
<CONFIGURATION>
  <APPLICATION_SERVER>
    ...
    <COMPONENT_LIST>
      EXECUTION (for an application) on page 369 [...]
      TIMEOUT (for an application) on page 412 [...]
      autoLogoutComponent [...]
      serviceTimeoutComponent [...]
      pictureComponent [...]
      renderingComponent [...]
      httpCookiesComponent [...]
      themeComponent [...]
    </COMPONENT_LIST>
    ...
  </APPLICATION_SERVER>
</CONFIGURATION>
```

Syntax notes
1. The COMPONENT_LIST element does not support any attributes.
2. Components are grouped by type. Each component type is discussed in its own section of this manual.

Child elements
A COMPONENT_LIST element may contain the following child elements:

- Zero to many WEB_APPLICATION_EXECUTION_COMPONENT elements.
- Zero to many SERVICE_APPLICATION_EXECUTION_COMPONENT elements.
- Zero to many WEB_APPLICATION_TIMEOUT_COMPONENT elements.
- Zero to many AUTO_LOGOUT_COMPONENT on page 359 elements.
- Zero to many SERVICE_APPLICATION_TIMEOUT_COMPONENT elements.
- Zero to many WEB_APPLICATION_PICTURE_COMPONENT elements.
- Zero to many WEB_APPLICATION_RENDERING_COMPONENT elements.
- Zero to many WEB_APPLICATION_HTTP_COOKIES_COMPONENT elements.
- Zero to many WEB_APPLICATION_THEME_COMPONENT elements.
**COMMAND (for auto logout)**

The `COMMAND` is an optional configuration element of `AUTO_LOGOUT_COMPONENT` on page 359 that provides a mechanism for the Genero Application Server to override an application's auto logout.

**Syntax**

```xml
<COMMAND Timeout="timeoutSeconds"> commandScript </COMMAND>
```

**Syntax notes**

1. `timeoutSeconds` is the number of seconds as the allowed time for a command to run.
2. `commandScript` is the name of the script or command to be run.

**Usage example**

```xml
<COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND>
```

The command determines whether the auto logout takes place based on the value of an exit code it returns, or whether it times out:

1. If the command's exit code is zero, the auto logout is ignored and the application keeps running.
2. If the command's exit code is not zero, the auto logout is performed.

**Note:** If the command times out, the auto logout process is also performed the same as if the exit code was not zero.

**Usage Scenario**

A typical use of the `COMMAND` option might be to check when the auto logout process is allowed. The external command could be used, for example, to only allow auto logout between the hours of 21:00 pm and 08:00 am, i.e. there would be no auto logout during work hours.

**Parent elements**

This element is a child of the following element: `APPLICATION_SERVER` on page 357

**CONFIGURATION**

This element is the starting point for the Genero Application Server configuration.

The `CONFIGURATION` element does not support any attributes.

**Syntax**

```xml
<CONFIGURATION>
  <APPLICATION_SERVER>...</APPLICATION_SERVER>
</CONFIGURATION>
```

**Child elements**

The `CONFIGURATION` element contains a single child element:

- One `APPLICATION_SERVER` element.
Example

```xml
<CONFIGURATION
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfas.xsd">
   <APPLICATION_SERVER>...</APPLICATION_SERVER>
</CONFIGURATION>
```

Parent elements
The **CONFIGURATION** element is the top-most element of the GAS configuration file. It has no parent element.

**CONSTANT**
Zero or one **CONSTANT**. Define a constant and optionally its value.

Syntax

```xml
<CONSTANT Id="cstId"> cst </CONSTANT>
```

Syntax notes
1. `cstId` is the constant name
2. `cst` is the constant value.

Example

```xml
<CONSTANT Id="constant1">A value</CONSTANT>
```

Parent elements
This element is a child of one of the following elements: **HTTP_COOKIE** on page 378

**DELEGATE**
The **DELEGATE** element specifies the Genero REST service in charge of all starting requests of the form /wa/r, /ja/r and /ws/r. It takes an attribute called **service** where you define the group and name of the Genero REST service.

Syntax

```xml
<DELEGATE service="serName">
   <!-- AnyTag1 --> anyValue </AnyTag1>
   <!-- AnyTag2 --> anyValue </AnyTag2>
</DELEGATE>
```

All child elements are optional and are passed as parameters to the REST service if present. See How to implement delegation on page 113.

When working with a single sign-on solution, child elements of the **DELEGATE** element will be specific to the identity provider (IdP). You will need to add the appropriate tags to work with your IdP. These tags should be documented by your IdP.

Parent elements
This element is a child of one of the following elements:

- **WEB_APPLICATION_EXECUTION_COMPONENT** on page 417
- **SERVICE_APPLICATION_EXECUTION_COMPONENT** on page 402
DESCRIPTION
The DESCRIPTION element allows a short and a long description to be associated with an application definition.

Syntax

```xml
<DESCRIPTION>
  [<SHORT>shortDescription</SHORT>]
  [<LONG>longDescription</LONG>]
</DESCRIPTION>
```

**Child elements**

The DESCRIPTION element may contain the following child elements:

1. Zero or one SHORT element.
2. Zero or one LONG element.

**Example**

```xml
<DESCRIPTION>
  <SHORT>A short description</SHORT>
  <LONG>A long description</LONG>
</DESCRIPTION>
```

**Parent elements**

This element is a child of one of the following elements:

- APPLICATION (for an application) on page 354
- APPLICATION (for a service) on page 356

DOC_TYPE

The DOC_TYPE element can be used to override the document type definition (DTD) in the main template.

Syntax

```xml
<DOC_TYPE>
  [<NAME>name</NAME>]
  [<EXTERNAL_ID>externalId</EXTERNAL_ID>]
  [<SYSTEM_ID>systemId</SYSTEM_ID>]
</DOC_TYPE>
```

**Child elements**

The DOC_TYPE element may contain the following child elements:

- One NAME element.
- Zero or one EXTERNAL_ID element.
- Zero or one SYSTEM_ID element.
Usage example

```xml
<DOC_TYPE>
  <NAME>HTML</NAME>
  <EXTERNAL_ID>-//W3C//DTD HTML 4.01//EN</EXTERNAL_ID>
  <SYSTEM_ID>http://www.w3.org/TR/html4/strict.dtd</SYSTEM_ID>
</DOC_TYPE>
```

In this example, the DTD in the main template would be overwritten with:

```xml
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

**Parent elements**

This element is a child of one of the following elements:

- `WEBAPPLICATION_RENDERING_COMPONENT` on page 419
- `RENDERING` on page 398

**DOCUMENT_ROOT**

The `DOCUMENT_ROOT` element specifies root directories that determine filesystem paths to serve files. The default document root directory (`res.path.docroot`) is `FGLASDIR/web`. This directory is the default bootstrap template path which is searched for CSS, JavaScript, demos.html, and other files served to the Genero Web Client for JavaScript (GWC-JS) front-end by the Application Server, see `GWC-JS` on page 374.

**Note:** When a Web server is included in the solution architecture, the GAS's document root directory is separate to the Web server's, which has its own document root - often called `htdocs` for the Apache Web server or `C:\Inetpub\wwwroot` for IIS. Files located in the Web server document root are served by the Web server, while files located in the GAS `DOCUMENT_ROOT` are served by the GAS, i.e. the dispatcher.

For the GAS, other root directories can be specified as required. For example, the resource path `res.path.docroot.user`, which is not defined in the GAS configuration file by default, can be set as required for files generated at runtime, such as reports generated by the Genero Report Engine (GRE).

**Syntax**

```xml
<DOCUMENT_ROOT> docroot; docrootuser </DOCUMENT_ROOT>
```

It allows for multiple document root paths to be specified, the separator used between resource paths is a semi-colon, `;`.

The `DOCUMENT_ROOT` element does not support any attributes or have any child elements.

**Usage example**

```xml
<DOCUMENT_ROOT>/usr/fgl2c/as/web</DOCUMENT_ROOT>
```

In this usage example, if you have the `demos.html` file in this directory and wish to access the file, use the URL: `http://<app_server>:<port>/demos.html` (where the file is on the host where the GAS resides) or `http://<web_server>/gas/demos.html` (where the file is on the Web server host), see `File serving URIs` on page 55.

**Parent elements**

This element is a child of one of the following elements: `INTERFACE_TO_CONNECTOR` on page 381
DVM_AVAILABLE

The DVM_AVAILABLE timeout specifies how long (in seconds) the Genero Application Server allows for the DVM to start. This parameter provides a delay for the DVM to start and be available.

The DVM_AVAILABLE timeout provides a mechanism for the Genero Application Server to handle the failure of the DVM to start. If the DVM has not started by the time the DVM_AVAILABLE timeout expires, the Genero Application Server sends an error message to the front-end and logs the message: "DVM_AVAILABLE timeout expired."

This element is optional. If the element is not specified, the default time out is 100 seconds.

Web applications

When used as a child of the WEB_APPLICATION_TIMEOUT_COMPONENT element, the DVM_AVAILABLE timeout is only applicable when you start an application or you launch sub process in interactive mode (IN LINE MODE). If you run the sub process in background (IN FORM MODE), the DVM_AVAILABLE timeout is not applicable.

Web services

For a Web service, there is no notion of a front-end client. An error message is therefore not sent. In addition, a Web service is not expected to perform RUNs commands with new DVMs connecting by themselves.

Usage example

```
<DVM_AVAILABLE>10</DVM_AVAILABLE>
```

In this usage example, the DVM Available timeout is set to 10 seconds.

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_TIMEOUT_COMPONENT on page 421, SERVICE_APPLICATION_EXECUTION_COMPONENT on page 402, TIMEOUT (for a service) on page 413, TIMEOUT (for an application) on page 412

DVM_PINGTIMEOUT

Time out (in seconds) before the Genero Application Server sends a PING (keep-alive) message to a running DVM.

The DVM Ping timeout specifies how long (in seconds) the GAS waits before send a PING (keep-alive) message to a running DVM.

A running DVM waits at a maximum gui.protocol.pingTimeout seconds for a front-end PING message when there is no user activity. After this timeout period elapses, the program stops with an error.

A correct configuration requires that the DVM_PINGTIMEOUT be set lower than the DVM gui.protocol.pingTimeout setting. By default, the DVM gui.protocol.pingTimeout is defined as 600 seconds and the GAS DVM_PINGTIMEOUT is defined as 300 seconds.

Usage example

```
<DVM_PINGTIMEOUT>300</DVM_PINGTIMEOUT>
```

In this example, the DVM PING timeout is set to 300 seconds.

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_TIMEOUT_COMPONENT on page 421, TIMEOUT (for an application) on page 412
**DVM**

The **DVM** element specifies the name of the Dynamic Virtual Machine you want to use to start and run the application.

This value is typically *fglrun* for Linux®/UNIX™ Systems (UNX) and *fglrun.exe* for Windows® Systems (WNT).

**Syntax**

```xml
[DVM>
  dvm
</DVM>
```

**Usage example**

```xml
<DVM>$ (res.dvm.wa)</DVM>
```

**Parent elements**

This element is a child of one of the following elements:

- [WEB_APPLICATION_EXECUTION_COMPONENT](#) on page 417
- [SERVICE_APPLICATION_EXECUTION_COMPONENT](#) on page 402
- [EXECUTION (for an application)](#) on page 369
- [EXECUTION (for a service)](#) on page 371

**ENVIRONMENT_VARIABLE**

The **ENVIRONMENT_VARIABLE** element provides the value to be set for an environment variable. It takes an attribute **Id**, which specifies the name of the environment variable. Prior to starting the application, the environment variable is set using this information.

**Usage example**

```xml
<ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
```

In this example, the environment variable FGLGUI is set to 1.

See also [Application environment](#) on page 43

**Parent elements**

This element is a child of one of the following elements:

- [WEB_APPLICATION_EXECUTION_COMPONENT](#) on page 417
- [SERVICE_APPLICATION_EXECUTION_COMPONENT](#) on page 402
- [EXECUTION (for an application)](#) on page 369
- [EXECUTION (for a service)](#) on page 371

**EXECUTION (for an application)**

This **EXECUTION** element sets the runtime environment for an application by specifying parameters for executing it.

It is defined by referencing in its **Using** attribute a predefined **WEB_APPLICATION_EXECUTION_COMPONENT** on page 417 to inherit runtime environment settings, and/or by setting individual execution elements specific to the application.

When settings are inherited from a **WEB_APPLICATION_EXECUTION_COMPONENT**, settings you define locally within the **EXECUTION** element override the component settings.
Syntax

```xml
<EXECUTION Using="web_application_execution_component_id" |
  AllowUrlParameters="[TRUE|FALSE]" |
  AllowUnsafeSession="[TRUE|FALSE]" |
  <![ENVIRONMENT_VARIABLE Id="envId">env</ENVIRONMENT_VARIABLE>]]...|
  <![PATH>path</PATH>]
  <![DVM>dvm</DVM>]]
  <![MODULE>module</MODULE>]
  <![PARAMETERS>...</PARAMETERS>]
  <![ACCESS_CONTROL>...</ACCESS_CONTROL>]
  <![DELEGATE>...</DELEGATE>]
  <![WEB_COMPONENT_DIRECTORY>webComponent1;</WEB_COMPONENT_DIRECTORY>]]...
</EXECUTION>
```

Syntax notes

Attributes for this element include:

1. **Using**: References an execution component identified by its `Id` attribute.
2. **AllowUrlParameters**: Defines whether parameters set in the application URL at the command line should be ignored ("FALSE", default value) or provided to the DVM ("TRUE").
3. **AllowUnsafeSession**: Defines whether safe session management is active ("FALSE", default value) or deactivated ("TRUE"). Safe session management is a transparent session check based on session cookies, to secure the application session tracking. You should deactivate (set to "TRUE") only if you encounter issues when migrating to version 2.50.20 or greater.

Child elements

**Important**: Element order. If child elements are present, they must be set in the order listed or as shown.

All child elements are optional and are passed as parameters to the REST service if present. See [How to implement delegation](#) on page 113.

When working with a single sign-on solution, child elements of the **DELEGATE** element will be specific to the identity provider (IdP). You will need to add the appropriate tags to work with your IdP. These tags are documented by your IdP.

The **EXECUTION** element may contain the following child elements:

1. Zero to many **ENVIRONMENT_VARIABLE** elements.
2. Zero or one **PATH** element.
3. Zero or one **DVM** element.
4. Zero or one **MODULE** element.
5. Zero or one **PARAMETERS** element.
6. Zero or one **ACCESS_CONTROL** element.
7. Zero or one **DELEGATE** element.
8. Zero or one **WEB_COMPONENT_DIRECTORY** element.

For more information on defining execution components, see [Web Application Execution Component](#) and [Service Application Execution Component](#).

Usage examples

```xml
<EXECUTION Using="cpn.wa.execution.local" />

<EXECUTION Using="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
</EXECUTION>
```
Parent elements
This element is a child of the **APPLICATION** (for an application) on page 354 element.

EXECUTION (for a service)
The **EXECUTION** element sets the runtime environment for the application by specifying the parameters for executing a Web application. You can reference a predefined **SERVICE_APPLICATION_EXECUTION_COMPONENT** to inherit the runtime environment settings of that component by including the **Using** attribute, specifying the unique identifier for that execution component, and/or you can set individual execution elements specific to the application.

Settings defined locally within the **EXECUTION** element override settings defined in included execution components.

Child elements
Possible execution elements include:

1. Zero or more **ENVIRONMENT_VARIABLE** elements.
2. Zero or one **PATH** element.
3. Zero or one **DVM** element.
4. Zero or one **MODULE** element.
5. Zero or one **PARAMETERS** element.
6. Zero or one **ACCESS_CONTROL** element.
7. Zero or one **DELEGATE** element.
8. Zero or one **WEB_COMPONENT_DIRECTORY** on page 422 element.
9. Zero or one **POOL** on page 395/xref> element.

Usage examples

```xml
<EXECUTION Using="cpn.ws.execution.local" />

<EXECUTION Using="cpn.ws.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
</EXECUTION>
```

Parent elements
This element is a child of one of the following elements: **APPLICATION** (for a service) on page 356

**EXTERNAL_ID**
The **EXTERNAL_ID** element is the public identifier of the DTD.

Usage example

```xml
<EXTERNAL_ID>-//W3C//DTD HTML 4.01//EN</EXTERNAL_ID>
```
Parent elements
This element is a child of one of the following elements: `DOC_TYPE` on page 366

FILE_TRANSFER
The `FILE_TRANSFER` element sets up parameters relevant for applications involved in transferring files between the front-end and the application server host. The `FILE_TRANSFER` element specifies when the transferred files should be deleted after application end. The files are removed from the temporary directory at the dispatcher shut down and after application end, when the timeout specified between the `TIMEOUT` tags expires.

Syntax

```
<FILE_TRANSFER>
  <TIMEOUT>duration</TIMEOUT>
</FILE_TRANSFER>
```

Child elements
The `FILE_TRANSFER` element must contain the following child element:

1. One `TIMEOUT` element.

Example

```
<FILE_TRANSFER>
  <TIMEOUT>600</TIMEOUT>
</FILE_TRANSFER>
```

Parent elements
This element is a child of one of the following elements: `APPLICATION_SERVER` on page 357

FORMAT
The `FORMAT` element specifies the format of each log message.

The `FORMAT` element takes an optional attribute `Type` (whose value can only be TEXT) and a list of field identifiers.

The field identifiers indicate the line format. The fields are separated by spaces. The order of the fields determines the order of the log output.

Table 68: Log message field identifiers

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Name of the category filter of the event.</td>
</tr>
<tr>
<td>component</td>
<td>Name of the component that generate the event.</td>
</tr>
<tr>
<td>date</td>
<td>Date of the event.</td>
</tr>
<tr>
<td>time</td>
<td>Time of the event.</td>
</tr>
<tr>
<td>relative-time</td>
<td>Time elapsed since the dispatcher has started.</td>
</tr>
<tr>
<td>process-id</td>
<td>Process identifier.</td>
</tr>
<tr>
<td>thread-id</td>
<td>Thread identifier.</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>location</td>
<td>The file name and line number where the event occurred. Replaces line and file Type attributes from earlier versions.</td>
</tr>
<tr>
<td>contexts</td>
<td>Internal data representing the successive context the process went through to reach the logged event. These data is formatted as type=value pairs separated by semicolons.</td>
</tr>
<tr>
<td>event-type</td>
<td>Event name.</td>
</tr>
<tr>
<td>event-params</td>
<td>Event details. XML fragment representing the structured information attached to the event. This XML fragment will be on a single line. If used this field should be the last one as it can contain data with spaces.</td>
</tr>
</tbody>
</table>

When parsing the content of log files:

- A log message ends with a LF/CR character.
- event-type values are enclosed in double quotes (").
- event-params values may contain any characters. This field should be put at the end of the log format string.
- All other field values do not contain spaces.
- A missing value is replaced by a hyphen (-).
- Within event-params values, well-known non-printable characters are replaced by standard C escape sequences; other non-printable characters use a hexadecimal encoding.
- Single event-params values, such as raw text, have the following structure: Data Size=size Content=value, where size is the size of the value.
- event-params contains the escaped log message.

Example

```xml
<FORMAT Type="TEXT">time event-type event-params</FORMAT>
<FORMAT>date time relative-time process-id thread-id contexts event-type event-params</FORMAT>
```

Parent elements

This element is a child of one of the following elements: LOG on page 383

GROUP (for an application)

This GROUP element allows you to specify a directory where external application configuration files are located.

Once a GROUP has been declared, an administrator can add an external application configuration file into the specified directory, and the Genero Application Server will be able to locate and use that file without having to be restarted.

It takes an attribute Id, which specifies the unique identifier for this group. When calling an application defined by an external application configuration file, you must provide the group Id and the name of the external application configuration file name (without the extension), which is typically the name of the application.

Syntax

```
[<GROUP Id="groupName">path</GROUP>][....]
```

Note:
1. *groupName* is a string that uniquely identifies the group.
2. *path* is the directory path where the external application configuration files are to be placed.

Usage examples

```
<GROUP id="tut-demo">$(res.path.as.demo)/tut/app</GROUP>
<GROUP id="mygroup">/home/myuser/config</GROUP>
```

Parent elements

This element is a child of the [APPLICATION_LIST](#) element.

**GROUP (for a service)**

The GROUP element allows you to specify a directory where external Web service application configuration files are located. Once a GROUP has been declared, an administrator can add an external Web service application configuration file into the specified directory, and the Genero Application Server will be able to locate and use that file without having to be restarted.

It takes an attribute *Id*, which specifies the unique identifier for this group. When calling an application defined by an external application configuration file, you must provide the group *Id* and the name of the external Web service application configuration file name (without the extension), which is typically the name of the Web service application.

Syntax

```
<GROUP Id="groupName"> path </GROUP>
```

Syntax notes

1. *groupName* is a string that uniquely identifies the group.
2. *path* is the directory path where the external Web service application configuration files are to be placed.

Usage example

```
<GROUP id="mygroup">/home/myuser/config</GROUP>
```

Parent elements

This element is a child of one of the following elements: [SERVICE_LIST](#) element.

**GWC-JS**

The GWC-JS element defines the customization directory for the Genero Web Client for JavaScript (GWC-JS) front-end.

Syntax

```
<GWC-JS>gwc_js_dir</GWC-JS>
```

1. Where *gwc_js_dir* is the name of a directory.

   The Genero Application Server (GAS) searches for this directory in [GWC_JS_LOOKUP_PATH](#) paths. If *gwc_js_dir* is found, it is expected to be a GWC-JS installation containing the bootstrap file (bootstrap.html).
Usage
You use this element to specify a customization directory for the GWC-JS. This directory contains a collection of HTML, CSS, JavaScript files, and all the resources required for rendering Web applications.

Usage example: default application
One or more customization directories are provided at GAS installation, and the default one is defined in the $(res.gwc-js) resource. The GWC-JS element in the defaultwa abstract application configuration in the GAS configuration file references this resource:

```
...<APPLICATION Id="defaultwa" Abstract="TRUE">
   <!-- This is the "default" application. It is not used directly: it is used for defining a "root" application. -->
   <EXECUTION Using="cpn.wa.execution.local"/>
   <UA_OUTPUT>
      <PROXY>${res.uaproxy.cmd}</PROXY>
      <PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
      <TIMEOUT Using="cpn.wa.timeout"/>
      <GWC-JS>${res.gwc-js}</GWC-JS>
   </UA_OUTPUT>
...<RESOURCE Id="res.gwc-js" Source="INTERNAL">gwc-js</RESOURCE>
```

The value of the $(res.gwc-js) resource is set by default in the as.xcf file to the "gwc-js" directory, which is located in the GAS installation directory, $(res.path.as).

Usage example: customization project directory
In this example, you create a new set of files to customize the look and feel of a Web application in a separate directory to the default bootstrap, for example, $(res.path.as)/gwc-js-custom. The GAS searches for the GWC-JS directory at paths defined by the GWC_JS_LOOKUP_PATH on page 376.

```
<GWC-JS>gwc-js-custom</GWC-JS>
```

URI gwc-js query string parameter
The customization directory can also be provided in the application URL using the gwc-js query string parameter. If the URI contains a gwc-js query string and value, this takes precedence over the configuration for the GWC-JS element in the application configuration file (xcf). See Set gwc-js customization with query string parameter on page 58.

Child elements
There are no child elements.

Parent elements
This element is a child of the following elements: UA_OUTPUT on page 414
**GWC_JS_LOOKUP_PATH**

Configure the `GWC_JS_LOOKUP_PATH` element to specify the location of your customized Genero Web Client for JavaScript (GWC-JS) front-end.

**Syntax**

```xml
<GWC_JS_LOOKUP_PATH>gwc_js_userdir;docroot</GWC_JS_LOOKUP_PATH>
```

1. The `GWC_JS_LOOKUP_PATH` allows for multiple directory paths to be specified. The separator between resource paths is a semi-colon (`;`).

**Usage**

You use this element to specify the path to the GWC-JS on page 374. Examples are shown of how to specify this using the `GWC_JS_LOOKUP_PATH` and with dispatcher overrides.

**Usage example**

```xml
<GWC-JS-LOOKUP-PATH>${res.path.gwcjs.user};${res.path.docroot}</GWC-JS-LOOKUP-PATH>
```

In this example, the GAS first searches the directory specified by the resource `res.path.gwcjs.user`. This resource is not defined in the GAS configuration file by default, however it can be set as required at runtime.

If the specified customization folder is not found, the GAS then looks for the GWC-JS directory at the default location defined by the resource `res.path.docroot`, which is defined by default in the GAS configuration file as `$FGLASDIR/web`.

**Note:** The GAS provides a dedicated URL as the document root where your customized GWC-JS directory is located (see `ua/w/` in Application URIs on page 51).

If the directory is found, the requested file is sent. If the file does not exist, no additional searching is performed. This is to avoid the risk of mixing up files in other GWC-JS versions that may be located in various directories in these paths.

**Set GWC_JS_LOOKUP_PATH at runtime**

```
httpdispatch -E res.path.gwcjs.user=<directory-of-your-choice>
```

This example sets the location of your customized GWC-JS files with the dispatcher switch (`-E`) by creating or overwriting the `res.path.gwcjs.user` resource with the path to the directory holding the customization folder.

**Child elements**

There are no child elements.

**Parent elements**

This element is a child of the following element: `INTERFACE_TO_CONNECTOR` on page 381
The HEADER element defines the request and response type communication carried on the HTTP protocol between Web applications, Web services and the client.

Starting with GAS 3.00 you can set custom HTTP headers for Web applications and Web services. This configuration takes place in the HTTP element of the INTERFACE_TO_CONNECTOR element of the GAS configuration file.

The HEADER element allows you to configure the launching of Web applications and Web services. HTTP headers can be configured for applications using the APPLICATION element and for Web services using the SERVICE element.

Syntax

```xml
[<HEADER Name="headerName">value[; ...]</HEADER>] [...]
```

Syntax notes

1. The NAME attribute defines the unique identifier for the HTTP header.
2. The value is a list of values separated by semi-colons.

Usage example - configure security headers

To set the HTTP security headers that comply with Open Web Application Security Project (OWASP) recommendations, configure the following headers shown highlighted in the example:

- The "X-XSS-Protection" header disables the cross-site scripting (XSS) filter built into most web browsers. This is usually enabled by default.
- The "X-Content-Type-Options" header prevents Internet Explorer and Google Chrome from sniffing a response away from the declared content-type. This helps reduce the danger of drive-by or unintended downloads.
- The "X-Frame-Options" header provides clickjacking protection by not allowing iframes to load on your site.

```xml
<INTERFACE_TO_CONNECTOR>
  ...
  <HTTP>
    <SESSION_COOKIE/>
    <APPLICATION>
      <HEADER Name="X-XSS-Protection">1; mode=block</HEADER>
      <HEADER Name="X-Content-Type-Options">nosniff</HEADER>
      <HEADER Name="X-Frame-Options">SAMEORIGIN</HEADER>
    </APPLICATION>
    <SERVICE>
      <HEADER/>
    </SERVICE>
  </HTTP>
</INTERFACE_TO_CONNECTOR>
```

Parent elements

This element is a child of the following elements:

- APPLICATION (for HTTP) on page 355
- SERVICE (for HTTP) on page 404
HTTP
The **HTTP** element contains elements defining the communication carried on the HTTP protocol between Web applications and the client.

The **HTTP** element defines a secure flag for cookies and the HTTP headers for applications and/or services. It therefore can contain **APPLICATION** and **SERVICE** elements.

**Syntax**

```xml
<HTTP>
<SESSION_COOKIE Secure="["TRUE|FALSE"]"></SESSION_COOKIE>
<APPLICATION>...</APPLICATION>
<SERVICE>...</SERVICE>
</HTTP>
```

**Child elements**
The **HTTP** element may contain the following child elements:

- One **SESSION_COOKIE** element
- One **APPLICATION** (for **HTTP**) element
- One **SERVICE** (for **HTTP**) element

**Example**

```xml
<INTERFACE_TO_CONNECTOR>
...
<HTTP>
<SESSION_COOKIE Secure="TRUE"> </SESSION_COOKIE>
<APPLICATION>
<HEADER/>
</APPLICATION>
<SERVICE>
<HEADER/>
</SERVICE>
</HTTP>
</INTERFACE_TO_CONNECTOR>
```

**Parent elements**
This element is a child of the following element:

- **INTERFACE_TO_CONNECTOR** element

**HTTP_COOKIE**
The **HTTP_COOKIE** element contains any HTTP cookie definitions for an application.

The main goal of cookies is to keep a state, through session variables, between two runs of an application by the same user. The number of cookies associated with an application should be constant.

**Syntax**

```xml
<HTTP_COOKIE Id="cid" [Expires="endTime" | Domain="mydomain" | Secure="TRUE|FALSE" | HttpOnly=""]>
<VARIABLE Id="varId">val</VARIABLE> [...]  
<CONSTANT Id="cstId">cst</CONSTANT> [...]  
<HTTP_COOKIE> [...]  
```
Syntax notes

The `HTTP_COOKIE` element takes a mandatory `Id` attribute and four optional attributes: `Expires`, `Domain`, `Secure` and `HttpOnly`.

1. `cid` is the cookie name.
2. The `Expires` attribute specifies the cookie expiration date. `endTime` is cookie expiration in "Wdy, DD-Mon-YYYY HH:MM:SS GMT" format. You can set a relative date with "+X" or "X", where X represent a number of seconds. "X" will fix the cookie date only at the creation time and "+X" will regenerate a new date for the cookie on each HTTP request. The `Expires` attribute is optional.
3. The `Domain` attribute restricts the cookie to a specified domain. `mydomain` is the domain name the cookie is restricted to. The `Domain` attribute is optional.
4. When set to `TRUE`, the `Secure` attribute restricts the cookie to secured connections (HTTPS) only. Valid values are `TRUE` or `FALSE`. The `Secure` attribute is optional.
5. When set to `TRUE`, `HttpOnly` attribute disables the cookie access from client-side scripting languages, such as JavaScript, running in a browser. Valid values are `TRUE` or `FALSE`. The `HttpOnly` attribute is optional.
6. `varId` is the variable name and `val` its value.
7. `cstId` is the constant name and `cst` its value.

Child elements

The `HTTP_COOKIE` element may contain the following child elements, defined by a mandatory identifier and an optional value:

1. Zero to many `CONSTANT` elements
2. Zero to many `VARIABLE` elements

Example

```xml
<!-- secure persistent cookie with default variable value and constant value -->
<HTTP_COOKIES>
  <HTTP_COOKIE Id="cookie3" Expires="Wdy, DD-Mon-YYYY HH:MM:SS GMT"
                Domain="www.domain.com" Secure="TRUE" HttpOnly="TRUE">
    <VARIABLE Id="var7" />
    <VARIABLE Id="var8">Initial value</VARIABLE>
    <CONSTANT Id="constant1">A value</CONSTANT>
  </HTTP_COOKIE>
</HTTP_COOKIES>
```

For more information on HTTP cookies, see the [Session Variables and Cookies](#) page.

Parent elements

This element is a child of one of the following elements:

- `HTTP_COOKIES` on page 379
- `WEB_APPLICATION_HTTP_COOKIES_COMPONENT` on page 418

`HTTP_COOKIES`

The `HTTP_COOKIES` element contains child elements that define persistent session variables or constants.

Child elements

The `HTTP_COOKIES` element may contain the following child elements:

1. Zero or more `HTTP_COOKIE` elements.
Syntax

```xml
<HTTP_COOKIES>
  <HTTP_COOKIE ...>...</HTTP_COOKIE>
[...]
</HTTP_COOKIES>
```

Example

```xml
<!-- secure persistent cookie with default variable value and constant value -->
<HTTP_COOKIES>
  <HTTP_COOKIE Id="cookie3" Expires="Wdy, DD-Mon-YYYY HH:MM:SS GMT"
     Domain="www.domain.com" Secure="TRUE" HttpOnly="TRUE">
    <VARIABLE Id="var7" />
    <VARIABLE Id="var8">Initial value</VARIABLE>
    <CONSTANT Id="constant1">A value</CONSTANT>
  </HTTP_COOKIE>
</HTTP_COOKIES>
```

For more information on HTTP cookies, see the Session Variables and Cookies page.

Parent elements

This element is a child of one of the OUTPUT (under APPLICATION) on page 388 element.

**HTTP_HEADER**

The HTTP_HEADER element specifies the request HTTP header for request response.

Usage example

```xml
<HTTP_HEADER Id="Cache-Control">no-cache</HTTP_HEADER>
```

Parent elements

This element is a child of one of the following elements: OUTPUT (under APPLICATION) on page 388

**HTTP_REQUEST_ENCODING**

- When "Source" attribute is "INLINE": use content of HTTP_REQUEST_ENCODING to determine encoding used in HTTP request.
- When "Source" attribute is "REQUEST": use "_charset_" form field or Content-Type HTTP request header sent by User-Agent to determine encoding used in HTTP request. Use UTF-8 if "_charset_" form field or Content-Type HTTP request are missing.

Usage example

```xml
<HTTP_REQUEST_ENCODING Source="INLINE">ISO-8859-1</HTTP_REQUEST_ENCODING>
```

or

```xml
<HTTP_REQUEST_ENCODING Source="REQUEST"/>
```
HTTP_RESPONSE_ENCODING

- When "Source" attribute is "INLINE": use content of HTTP_RESPONSE_ENCODING and force HTTP response encoding to this value.
- When "Source" attribute is "REQUEST": use Accept-Charset HTTP request header sent by User-Agent to determine HTTP response encoding. Use UTF-8 if Accept-Charset is missing.

Usage example

```xml
<HTTP_RESPONSE_ENCODING Source="INLINE">ISO-8859-1</HTTP_RESPONSE_ENCODING>
```

or

```xml
<HTTP_RESPONSE_ENCODING Source="REQUEST"/>
```

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_RENDERING_COMPONENT on page 419 RENDERING on page 398

INTERFACE_TO_CONNECTOR

The INTERFACE_TO_CONNECTOR element specifies the connection between the GAS and the GAS Connector located with the Web server.

The GAS Connector is either:

- The FastCGI extension
- The ISAPI extension
- User agent through direct connection

The INTERFACE_TO_CONNECTOR element defines the port on which the Genero Application Server listens for incoming requests. This applies for the httpdispatcher, (the stand-alone GAS, see Dispatcher: httpdispatch on page 304) as well as the GAS Connectors.

Syntax

```xml
<INTERFACE_TO_CONNECTOR>
  [...]
  <ROOT_URL_PREFIX>URL_Web_Server_behind_proxy</ROOT_URL_PREFIX>
  <TCP_BASE_PORT>base</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>offset</TCP_PORT_OFFSET>
  <DOCUMENT_ROOT>docroot</DOCUMENT_ROOT>
  <GBC_LOOKUP_PATH>path</GBC_LOOKUP_PATH>
  <TEMPORARY_DIRECTORY>dir</TEMPORARY_DIRECTORY>
  <SESSION_DIRECTORY>dir</SESSION_DIRECTORY>
  <REPORT_VIEWER_DIRECTORY>dir</REPORT_VIEWER_DIRECTORY>
  <REPORT_REMOTE_URL_PREFIX>URL_GRE_server</REPORT_REMOTE_URL_PREFIX>
  <SOCKET_FAMILY>dir</SOCKET_FAMILY>
  <SOCKET_PATH>dir</SOCKET_PATH>
  [...]
</INTERFACE_TO_CONNECTOR>
```

Child elements

The INTERFACE_TO_CONNECTOR element may contain the following child elements.
1. Zero or more **ROOT_URL_PREFIX** on page 402 element.
2. One **TCP_BASE_PORT** on page 410 element.
3. One **TCP_PORT_OFFSET** on page 410 element.
4. One **DOCUMENT_ROOT** on page 367 element.
5. One **GWC_JS_LOOKUP_PATH** on page 376 element.
6. One **TEMPORARY_DIRECTORY** on page 411.
7. One **SESSION_DIRECTORY** on page 405 element.
8. One **REPORT_VIEWER_DIRECTORY** on page 398 element.
9. One **SOCKET_FAMILY** on page 408 element.
10. One **SOCKET_PATH** on page 409 element.
11. Zero or one **HTTP** on page 378 element.

**Example**

```xml
<INTERFACE_TO_CONNECTOR>
  <ROOT_URL_PREFIX></ROOT_URL_PREFIX>
  <TCP_BASE_PORT>6300</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  <GWC_LOOKUP_PATH>$(res.path.gwcjs.user);$(res.path.docroot)</GWC_LOOKUP_PATH>
  <TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>
  <SESSION_DIRECTORY>$(res.appdata.path)/session</SESSION_DIRECTORY>
  <REPORT_VIEWER_DIRECTORY>$(res.gredir)/viewer</REPORT_VIEWER_DIRECTORY>
  <SOCKET_FAMILY>$(res.dispatcher.socket.family)</SOCKET_FAMILY>
  <SOCKET_PATH>$(res.dispatcher.socket.path)</SOCKET_PATH>
  <HTTP/>
</INTERFACE_TO_CONNECTOR>
```

In this example, the application server is listening on port 6394 (`TCP_BASE_PORT + TCP_PORT_OFFSET`), the application server Web site root is specified as the resource `$(res.path.docroot)`.

To have several instances of the Genero Application Server run concurrently on the same host, you create several GAS configuration files with different offsets. Once started, each application server listens at the offset specified.

**Important:** If you create multiple GAS configuration files (one for each instance of the application server), take care to ensure that the port values are unique for each application server started.

If two application server configuration files both specify the same `TCP_BASE_PORT` and `TCP_BASE_OFFSET`, a port conflict exists. The second application server will not start, an error message displays (Application Server startup........[fail]) and the message "Address already in use" is written to the log file.

**Parent elements**

This element is a child of the **APPLICATION_SERVER** on page 357 element.

**INTERFACE_TO_DVM**

The **INTERFACE_TO_DVM** element specifies the address of the host where the GAS dispatcher runs.

There can only be one **INTERFACE_TO_DVM** element specifying the connection between the GAS and the Dynamic Virtual Machine (DVM) in a given Genero Application Server configuration file.

**Syntax**

```xml
<INTERFACE_TO_DVM>
  <ADDRESS> address </ADDRESS>
</INTERFACE_TO_DVM>
```
Child elements

The `INTERFACE_TO_DVM` element contains a child element:

1. One `ADDRESS` element.

Example

```xml
<INTERFACE_TO_DVM>
  <ADDRESS>app_server</ADDRESS>
</INTERFACE_TO_DVM>
```

Parent elements

This element is a child of the following elements: `APPLICATION_SERVER` on page 357

**KEEP_ALIVE**

Time out (in seconds) before shutting down a proxy that is no longer serving requests.

The proxy `KEEP_ALIVE` timeout is used to shut down the proxy that hasn't served any requests during the specified time. The `KEEP_ALIVE` element specifies how long (in seconds) the proxy waits with no request to process before determining whether to shut down.

If the `KEEP_ALIVE` entry is missing, the proxy will never shutdown. It also prevents the DVMs from running indefinitely in the unlikely event that the GAS dispatcher or the web server crashes.

Usage example

```xml
<KEEP_ALIVE>10</KEEP_ALIVE>
```

In this usage example, the proxy timeout is set to 10 seconds.

Parent elements

This element is a child of one of the following elements: `SERVICE_APPLICATION_TIMEOUT_COMPONENT` on page 403, `TIMEOUT (for a service)` on page 413

**LONG**

The `LONG` element contains the long description to be associated with an application definition.

Usage example

```xml
[<LONG>longDescription</LONG>]
```

Parent elements

This element is a child of the `DESCRIPTION` on page 366 element.

**LOG**

The `LOG` element specifies details of the log files the Genero Application Server creates.

The GAS creates logs for each of its dispatchers, proxies, and DVMs. The `LOG` element and its child elements specify where the log files are created, the format of the log messages, the type of information logged, and the maximum size of a single log message.

It is possible to specify multiple `LOG` elements. For example, you may need to have separate log files capturing different categories of messages.
Syntax

```xml
<LOG>
  <OUTPUT Type="{DAILYFILE|CONSOLE|CONSOLE,DAILYFILE}" path/filename></OUTPUT>
  <FORMAT Type="TEXT">fields-id...\</FORMAT>
  <CATEGORIES_FILTER>filterName...\</CATEGORIES_FILTER>
  <RAW_DATA MaxLength="length" />
</LOG>
```

Child elements

The `LOG` element may contain the following child elements:

1. One `OUTPUT` element.
2. One `FORMAT` element.
3. One `CATEGORIES_FILTER` element (optional).
4. Zero or one `RAW_DATA` element.

Example

```xml
<LOG>
  <OUTPUT Type="DAILYFILE">$(res.log.output.path)</OUTPUT>
  <FORMAT Type="TEXT">time event-type event-params</FORMAT>
  <CATEGORIES_FILTER>GAS ACCESS PROCESS DEPRECATED ERROR WARNING</CATEGORIES_FILTER>
  <RAW_DATA MaxLength="-1" />
</LOG>
```

To support more than one GAS process using the same log configuration, if a required log file cannot be opened, the GAS process ID will be added to the log file name.

Parent elements

This element is a child of the `APPLICATION_SERVER` on page 357 element.

MAP

The `MAP` element is a combination of a rendering mechanism and a theme.

The `MAP` element takes a required attribute `Id`, which specifies the unique identifier for this component. The `Id` can be any value, but it is based on the result the user gets from `adua.xrd`. The default values include `DUA_GDC`, `DUA_UA`, (and `DUA_HTML5`, which is deprecated). The list can be extended by custom choices.

The `MAP` element may also specify optional attributes:

- `Allowed`, which specifies whether this map will be used in this context or not. Possible values are `TRUE` (allowed) or `FALSE` (not allowed). Default value is `FALSE` or the parent definition when inherited. For example, this can be used to forbid some applications to use PDA if they were not designed to.
- `ForwardDVMStderr`, which specifies whether the DVM standard error will be forwarded to the Genero Desktop Client. This attribute is ignored for GWC applications. Default value is `FALSE` or the parent definition when inherited.

Child elements

The `MAP` element may contain the following child elements:

1. Zero or one `PROXY` element.
2. Zero or one `TIMEOUT` element.
3. Zero or one PICTURE element.
4. Zero or one RENDERING element.
5. Zero or one THEME elements.

Parent elements
This element is a child of one of the following elements: OUTPUT (under APPLICATION) on page 388

MAXAVAILABLE
The MAXAVAILABLE element specifies the maximum number of available DVMs to be attached to a Web service.

Syntax
```
<MAXAVAILABLE> maxValue </MAXAVAILABLE>
```

Constraints
```
START <= MAXAVAILABLE
MIN_AVAILABLE <= MAXAVAILABLE
```

To control the number of DVMs - and in effect the number of licenses - used by each application, you can split your licenses using MAXAVAILABLE. By using a different fglprofile for each application, you can specify X licenses for application 1, Y licenses for application 2.

Parent elements
This element is a child of the POOL on page 395 element.

MAXREQUESTSPER_DVM
The MAXREQUESTSPER_DVM element specifies the maximum number of requests a DVM can handle before being stopped by the pool manager.

The value must be equal to or greater than 1.

Syntax
```
<MAXREQUESTSPER_DVM> maxRequests </MAXREQUESTSPER_DVM>
```

Parent elements
This element is a child of the POOL on page 395 element.

MONITOR
The MONITOR element in the Genero Application Server configuration file specifies from which machines the monitor URL is accessible. By default, the monitor page is not accessible and needs to be configured.

Important: Depending on the network configuration, the monitor is not always able to get the actual client IP address. If there is a proxy server between the client and the server, for example, the client IP address seen by the GAS may be the address from the proxy server.

Syntax
```
<MONITOR>
  <ALLOW_FROM>ip_address</ALLOW_FROM>
</MONITOR>
```
The *ip_address* is a valid IPv4 or IPv6 address. For IPv4 it can be a complete IP address or a network address (ending with a dot).

**Child elements**

- Zero or more [ALLOW_FROM](#) elements.

**Example configuring monitor access**

In the default deployment monitoring is specified by the resource `res.access.control`, which is defined with the value NOBODY by default:

> Note: MONITOR control rules will be ignored by the standalone dispatcher (httpdispatch).

> Important: The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.

To allow monitoring from hosts, in the GAS configuration file (default `FGLASDIR/etc/as.xcf`) you need to change the application element for MONITOR from:

```xml
  <MONITOR>
    <ALLOW_FROM>${res.access.control}</ALLOW_FROM>
    <!--
    <ALLOW_FROM>192.168.</ALLOW_FROM>
    <ALLOW_FROM>10.</ALLOW_FROM>
    <ALLOW_FROM>193.111.222.123</ALLOW_FROM>
    -->
  </MONITOR>
```

To (for example):

```xml
  <MONITOR>
    <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
    <ALLOW_FROM>192.168.</ALLOW_FROM>
    <ALLOW_FROM>10.</ALLOW_FROM>
    <ALLOW_FROM>193.111.222.123</ALLOW_FROM>
  </MONITOR>
```

In this example, the GAS monitor is reachable from localhost (127.0.0.1), 193.111.222.123 and all IP that begin with "192.168." or "10.". For more details on the monitor usage, see Monitoring Genero Application Server

**Parent elements**

This element is a child of one of the [APPLICATION_SERVER](#) element.

**MIME_TYPE**

The MIME_TYPE element sets the MIME type in HTTP Content-Type response header. If not specified, the default MIME_TYPE of "text/html" is used by the output driver.

**Usage example**

```xml
  <MIME_TYPE>text/html</MIME_TYPE>
  <MIME_TYPE>text/xml</MIME_TYPE>
```
Parent elements
This element is a child of one of the following elements: WEB_APPLICATION_RENDERING_COMPONENT on page 419 RENDRING on page 398

MIN_AVAILABLE
The MIN_AVAILABLE element specifies the minimum number of available DVMs to be attached to a Web Service.

It can be either less than or greater than the value specified by START. If \texttt{START} > \texttt{MINAVAILABLE}, the number of DVMs can decrease to reach \texttt{MINAVAILABLE}.

Syntax
\[
\text{[}<\text{MINAVAILABLE}> \text{minValue} \text{</MINAVAILABLE}>]\]

Constraint
0 <= MIN_AVAILABLE <= MAX_AVAILABLE

Parent elements
This element is a child of one of the following elements: POOL on page 395

MODULE
The MODULE element specifies the application module name (the name of the .42r module you want to run). If omitted, the GAS uses the name of the requested application.

While this element can be specified as part of an execution component, it is typically defined at the application level.

Usage example

\[
\text{<MODULE}>\text{Edit}</MODULE>\]

Parent elements
This element is a child of one of the following elements:

• WEB_APPLICATION_EXECUTION_COMPONENT on page 417
• SERVICE_APPLICATION_EXECUTION_COMPONENT on page 402
• EXECUTION (for an application) on page 369
• EXECUTION (for a service) on page 371

NAME
The NAME element is the name of the DTD.

Usage example

\[
\text{<NAME}>\text{HTML}</NAME>\]

Parent elements
This element is a child of one of the following elements: DOC_TYPE on page 366
The OUTPUT element specifies where the log messages are sent or written.

The OUTPUT element takes an attribute of Type and an optional path value. If the path value is not specified, the default of $FGLASDIR/log is used.

**Note:** The default Genero Application Server configuration file provides a value for the OUTPUT element, specifying a directory other than $FGLASDIR/log. This directory can vary depending on the operating system. Check the configuration file to identify the directory specified for your installation.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAILYFILE</td>
<td>Log files are written to disk. New log files are created each day. A directory is created each day with the naming convention naming YYYYMMDD. All log files created by the GAS during the day are stored in this directory. If a path is specified, the daily directory is created and stored under the log directory in the specified directory. If a path is not specified, the log files are created in the default log directory. DVM logs are redirected to files when DAILYFILE is set for the log output type.</td>
</tr>
<tr>
<td>CONSOLE</td>
<td>Log messages are sent to standard output.</td>
</tr>
</tbody>
</table>

**Examples**

```
<OUTPUT Type="DAILYFILE"/>
<OUTPUT Type="DAILYFILE">$(res.as.dir)/logdirs</OUTPUT>
<OUTPUT Type="CONSOLE"/>
<OUTPUT Type="CONSOLE,DAILYFILE"/>
```

In the first example, the daily log file is written to the default logging directory.

In the second example, the daily directory would be created in $(res.as.dir)/logdirs/log.

In the third example, log messages are sent to standard output.

In the fourth example, log messages are both written to a log file in the default logging directory and sent to standard output.

**Parent elements**

This element is a child of one of the following elements: LOG on page 383

**OUTPUT (under APPLICATION)**

Starting with Genero 3.00, the OUTPUT element is deprecated and used by GWC for HTML5 applications only.

**Note:**

As GWC for HTML5 is deprecated, the OUTPUT element is also deprecated. UI applications that are not using GWC for HTML5 use the UA_OUTPUT on page 414 element instead.
The **OUTPUT** element specifies the output parameters for a Web application, listing all maps required to make the defined Web application usable with different browsers/Front Ends.

It takes an optional attribute **Rule**, to assist with automatic discovery of the User Agent. For more information, see Automatic Discovery of User Agent.

**Child elements**
The **OUTPUT** element may contain the following child elements:

1. Zero or more **HTTP_HEADER** elements.
2. Zero or one **HTTP_COOKIES** elements.
3. Zero or more **MAP** elements.

**Parent elements**
This element is a child of one of the following elements: **APPLICATION (for an application)** on page 354

**OUTPUT_DRIVER**
The **OUTPUT_DRIVER** element specifies the output driver to be used. Valid values include:

- **GWC2** - The GWC2 output driver specifies that the snippet-based rendering engine be used to render the application for GWC front-ends.
- **JFE36** - This value is for use with GDC Front End clients.

**Usage example**

```xml
<OUTPUT_DRIVER>GWC2</OUTPUT_DRIVER>
<OUTPUT_DRIVER>JFE36</OUTPUT_DRIVER>
```

**Parent elements**
This element is a child of one of the following elements: **WEB_APPLICATION_RENDERING_COMPONENT** on page 419, **RENDERING** on page 398

**PARAMETERS**
The **PARAMETERS** element specifies the parameters to provide on the DVM command line. Each parameter is contained in its own **PARAMETER** element.

If allowed, parameters can also be set in the application URL. For example, `/ua/r/myapp?Arg=val1&Arg=Val2` provides two parameters. To enable URL parameters for Web applications, set the **AllowUrlParameters** attribute in the **EXECUTION (for an application)** on page 369 tag to **TRUE**. The default is **FALSE**. If the DVM already has parameters set by the command line, the parameters in the URL are added to the end of the command line.

**Important**: This attribute is not supported and should be removed for Web services applications.

**Syntax**

```xml
<PARAMETERS>
[<PARAMETER> parameterValue </PARAMETER> [...] ]
</PARAMETERS>
```

**Child elements**
- Zero or more **PARAMETER** elements.
Usage examples
The following example provides two parameters:

- Hello world!
- Again

```xml
<PARAMETERS>
  <PARAMETER>Hello world!</PARAMETER>
  <PARAMETER>Again</PARAMETER>
</PARAMETERS>
```

If URL parameters are allowed, these parameters are listed after the ones defined in the `PARAMETERS` element of the configuration file.

Parent elements
This element is a child of one of the following elements: `WEB_APPLICATION_EXECUTION_COMPONENT` on page 417, `SERVICE_APPLICATION_EXECUTION_COMPONENT` on page 402, `EXECUTION (for an application)` on page 369, `EXECUTION (for a service)` on page 371

PATH (under EXECUTION)
The `PATH` element specifies the current working directory for the application module.

Usage example

```xml
(PATH)/home/appdir/sales/
```

Parent elements
This element is a child of one of the following elements:

- `WEB_APPLICATION_EXECUTION_COMPONENT` on page 417
- `SERVICE_APPLICATION_EXECUTION_COMPONENT` on page 402
- `EXECUTION (for an application)` on page 369
- `EXECUTION (for a service)` on page 371

PATH (under PICTURE)
Starting with Genero 3.00, the `PATH (under PICTURE)` element is deprecated and used by GWC for HTML5 applications only.

Note:
UI applications that are using Genero Web Client for JavaScript (GWC-JS), see the `PUBLIC_IMAGEPATH` on page 396 under the `UA_OUTPUT` on page 414 element instead.

The `PATH` element specifies where to look for an image or resource. The exact contents of the `PATH` element depends on whether the `Type` attribute is set to `WEBSERVER` or `APPSERVER`.

Parent elements
This element is a child of one of the following elements: `WEB_APPLICATION_PICTURE_COMPONENT` on page 419, `PICTURE` on page 392

PATH with Type WEBSERVER
Starting with Genero 3.00, the `PATH with Type WEBSERVER` attribute is deprecated and used by GWC for HTML5 applications only.

Note:
UI applications that are using Genero Web Client for JavaScript (GWC-JS), see the PUBLIC_IMAGEPATH on page 396 under the UA_OUTPUT on page 414 element instead.

When you specify the Type as WEBSERVER, the PATH value provides the URL for the directory where the resources reside on the Web server side. This URL typically consists of the Web server directory resource combined with a directory alias.

Syntax

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
  <PATH Id="pathID" Type="WEBSERVER">$(connector.uri)/path</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

1. resID is the unique identifier for this picture component.
2. pathID defines what is located in the specified path, such as "Image" or "Resource". The Id attribute for the PICTURE element is optional; if not specified, it defaults to "Image".
3. The TYPE attribute is optional; if not specified, it defaults to WEBSERVER.
4. $(connector.uri) is the resource for the Web server directory. If you are using a direct connection to the GAS, the resource $(connector.uri) is empty. If you connect through an Apache web server, $(connector.uri) is replaced by /gas/, assuming your URL is
   
   http://WebServer/gas/ua/r/AppID

   If $(connector.uri) is not specified in the picture path, the web server is searched for the images.
5. path is relative to the DOCUMENT_ROOT on page 367 path set in the configuration to map to the physical directory that stores the image files. You need to specify a valid path according to the DOCUMENT_ROOT path in your configuration.

Example

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.picture.appserver">
  <PATH Id="Image" Type="WEBSERVER" >$(connector.uri)/pic</PATH>
  <PATH Id="Resource" Type="WEBSERVER" >$(connector.uri)/fjs</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Note: If DOCUMENT_ROOT is set to /usr/local/genero/web, our example maps the Image path to /usr/local/genero/web/pic and the Resource path to /usr/local/genero/web/fjs.

Note: The front-ends use $(pictures.uri) in their templates to access the pictures. This corresponds to the picture component path: $(connector.uri)/pic.

Tip: When creating your HTML pages, use the absolute path to HTML objects like images or JavaScript files; for example, use /fjs/pic/accept.png rather than ../pic/accept.png, because URLs containing .. will be rejected by the GAS for security reasons.

PATH with Type APPSERVER
Starting with Genero 3.00, the PATH with Type APPSERVER attribute is deprecated and used by GWC for HTML5 applications only.

Note:

UI applications that are using Genero Web Client for JavaScript (GWC-JS), see the PUBLIC_IMAGEPATH on page 396 under the UA_OUTPUT on page 414 element instead.

When you specify the Type of APPSERVER, the PATH value is then an enumeration of disk locations where the resource is potentially located. The server will then build specific URL for each resources and then catch these URL to search through the given paths for the resource on disk and deliver it.
Syntax

```xml
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
  <PATH Id="pathID" Type="APPSERVER" [ ExtensionFilter="ext" ]
     [ DVMFallbackAllowed="TRUE|FALSE" ]>pathlist</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Syntax notes

1. `resID` is the unique identifier for this picture component definition.
2. `pathID` defines what is located in the specified path, such as "Image" or "Resource". The `Id` attribute for the PICTURE element is optional; if not specified, it defaults to "Image".
3. `ExtensionFilter` is an optional attribute that filters access to application resources like images. Its value is a list of extensions, separated by semi-colons. For example: .png;.gif;.jpeg;.jpg;.bmp;.ico;.js;.css. The extensions are case sensitive. You might want to allow .png and not .PNG. If ExtensionFilter is not defined, access to any resources of the application is granted. ExtensionFilter is also used to filter image extensions that are asked to the DVM. By default, any applications which inherit `defaultgwc` configuration have restricted access to resources (see defaultgwc image component references in as.xcf).
4. `DVMFallbackAllowed` is an optional attribute that allows images and resources to be asked to the DVM if they are not found at the locations pointed by `pathlist`. By default, asking images to the DVM is allowed.
5. `pathlist` is a list of directory paths separated by semi-colons.

Example

```xml
<!-- A GAS multi-location Picture Component -->
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.html5.picture">
  <PATH Id="Resource" Type="WEBERVER">$(connector.uri)/fjs</PATH>
  <PATH Id="Image" Type="APPSERVER"
     ExtensionFilter="$(res.image.extensions)">
    $(res.path.tpl.html5)/img;$(res.path.pic);$(application.path)
  </PATH>
  <PATH Id="SetHtml5" Type="APPSERVER"
     ExtensionFilter="$(res.web.extensions);.less;.svg"
     DVMFallbackAllowed="FALSE">
    $(res.path.tpl.html5);$(res.path.tpl.common)
  </PATH>
  <PATH Id="WebComponents" Type="APPSERVER"
     ExtensionFilter="$(res.web.components.extensions)"
     DVMFallbackAllowed="FALSE">
    $(res.path.docroot)
  </PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

PICTURE

Starting with Genero 3.00, the PICTURE element is deprecated and used by GWC for HTML5 applications only.

Note:

UI applications that are using Genero Web Client for JavaScript (GWC-JS), see the `PUBLIC_IMAGEPATH` on page 396 under the `UA_OUTPUT` on page 414 element instead.

The PICTURE element specifies the picture parameters required by this Web application. It takes an attribute `Using`, where you can reference a predefined `WEB_APPLICATION_PICTURE_COMPONENT` (to inherit the picture parameters of that component), or you can specify the path to the image directory by including a `PATH` element.
Syntax

```xml
<PICTURE Using="componentID">
  <PATH Id="pathID" Type="pathType">Path</PATH>
</PICTURE>
```

Syntax notes

1. `componentID` is the unique identifier for a Web application picture component.

Child elements

The `PICTURE` element may contain the following child elements:

1. Zero to many `PATH` elements.

Usage example

```xml
<PICTURE Using="cpn.gwc.html5.picture"/>

<PICTURE>
  <PATH>${connector.uri}/iiug/images</PATH>
</PICTURE>
```

Each application output map can define a `PICTURE` element. This element will be used to look for images and resources handled by URLs generated by the `resourceURI()` SBRE function or the legacy `imageURI()` SBRE function.

There are several ways to define a `PICTURE` element:

1. The `PICTURE` element can simply refer to a previously defined `picture component`:

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/cfextwa.xsd">
  <EXECUTION>
    <PATH>${res.path.demo.app}/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
      <PICTURE Using="cpn.gwc.picture.appserver"/>
    </MAP>
  </OUTPUT>
</APPLICATION>

<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/cfextwa.xsd">
  <EXECUTION>
    <PATH>${res.path.demo.app}/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
      <PICTURE Using="cpn.gwc.picture.webserver"/>
    </MAP>
  </OUTPUT>
</APPLICATION>
```
2. The `PICTURE` element can define its own picture component based on the Web Server behavior:

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlio:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/cfextwa.xsd">
<EXECUTION>
  <PATH>${res.path.demo.app}/card/src</PATH>
  <MODULE>card.42r</MODULE>
</EXECUTION>
<OUTPUT>
  <MAP Id="DUA_HMTL5">
    <PICTURE>
      <PATH Id="Image" Type="WEBSERVER">${connector.uri}/card/img</PATH>
    </PICTURE>
  </MAP>
</OUTPUT>
</APPLICATION>
```

3. The `PICTURE` element can define its own picture component based on the AppServer behavior:

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlio:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/cfextwa.xsd">
<EXECUTION>
  <PATH>${res.path.demo.app}/card/src</PATH>
  <MODULE>card.42r</MODULE>
</EXECUTION>
<OUTPUT>
  <MAP Id="DUA_HTML5">
    <PICTURE>
      <PATH Id="Image" Type="APPSERVER">${res.path.demo.app}/card/img;${defaultPicturePath}</PATH>
    </PICTURE>
  </MAP>
</OUTPUT>
</APPLICATION>
```

4. For each application output map, you can define several `PATH` element using different IDs inside a `PICTURE` element. This allows the use of a mixed mechanism for resource referencing:

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlio:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/cfextwa.xsd">
<EXECUTION>
  <PATH>${res.path.demo.app}/card/src</PATH>
  <MODULE>card.42r</MODULE>
</EXECUTION>
<OUTPUT>
  <MAP Id="DUA_HTML5">
    <PICTURE Using="cpn.gwc.picture.webserver">
      <PATH Id="AppServerImage" Type="APPSERVER">${res.path.demo.app}/card/img;${defaultPicturePath}</PATH>
    </PICTURE>
  </MAP>
</OUTPUT>
</APPLICATION>
```
Parent elements
This element is a child of one of the following elements: MAP on page 384

PLATFORM_INDEPENDENT
This element contains a collection of platform-independent RESOURCE elements.
The PLATFORM_INDEPENDENT element contains a list of platform-independent resources, available for both Linux®/UNIX™ and Windows® platforms.

Child elements
A PLATFORM_INDEPENDENT element main contain the following child elements:
• Zero to many RESOURCE elements.

Parent elements
This element is a child of one of the following elements: RESOURCE_LIST on page 401

POOL
The POOL element sets the limitations regarding the number of Virtual Machines (DVMs) that are attached to a Web Service.
You specify four values within a POOL element:
• The number of DVMs to start when the GAS starts
• The minimum number of DVMs to have alive while the GAS is running
• The maximum number of DVMs to have alive while the GAS is running.
• The maximum number of requests a DVM can handle before being stopped by the pool.

  Note: The POOL element is only available for Web Services.

Syntax

```xml
<POOL>
  [START> startValue </START>]
  [MIN_AVAILABLE> minValue </MIN_AVAILABLE>]
  [MAX_AVAILABLE> maxValue </MAX_AVAILABLE>]
  [MAX_REQUESTS_PER_DVM> maxRequests </MAX_REQUESTS_PER_DVM>]
</POOL>
```

Child elements
The POOL element may contain the following child elements:

1. Zero or one START elements.
2. Zero or one MIN_AVAILABLE element.
3. Zero or one MAX_AVAILABLE element.
4. Zero or one MAX_REQUESTS_PER_DVM element.
Example

<POOL>
  <START>5</START>
  <MINAVAILABLE>3</MINAVAILABLE>
  <MAXAVAILABLE>10</MAXAVAILABLE>
  <MAXREQUESTS_PER_DVM>1</MAXREQUESTS_PER_DVM>
</POOL>

In this example, 5 DVMs are started to service the Web service when the GAS starts; the number can fall as low as 3 DVMs or rise as high as 10 DVMs. For more information on setting service pool elements, see the Service Pool section of the GAS Architecture topic.

Parent elements
This element is a child of one of the following elements:

- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 402
- EXECUTION (for a service) on page 371

PROXY (for an application)

Specifies the resource of the proxy you want to use for an application or a set of applications. For example, the resource references the proxy command `uaproxy` required for Genero Desktop Client and Genero Web Client for JavaScript applications.

Usage examples

<PROXY>$(res.uaproxy.cmd)</PROXY>

Parent elements
This element is a child of one of the following elements: UA_OUTPUT on page 414
MAP on page 384

PROXY (for a service)

Specifies the resource of the proxy you use for Web services. For example, the resource references the proxy command `gwsproxy` required for Web service applications.

Usage example

<PROXY>$(res.gwsproxy)</PROXY>

Parent elements
This element is a child of one of the following elements: APPLICATION (for a service) on page 356

PUBLIC_IMAGEPATH

The PUBLIC_IMAGEPATH element defines a path relative to the root path `appdata/public`, where `fg1run` looks for resources for common images used by applications. The "appdata" resource, as it is commonly known, is set by the resource `$(res.appdata.path)` in the GAS configuration file, see GAS directories on page 39.

Note: UI applications that use GWC for HTML5, see the PATH (under PICTURE) on page 390 element instead.
Syntax

```
<PUBLIC_IMAGEPATH>path</PUBLIC_IMAGEPATH>
```

Where `path` is a path relative to the root path `appdata/public`

Usage example 1

```
... 
<UA_OUTPUT>
  <PROXY></PROXY>
  <PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
  <GWC-JS>gwc-js</GWC-JS>
  <TIMEOUT Using="cpn.wa.timeout"/>
</UA_OUTPUT>
...
```

If the value of the resource `${res.public.resources}` is set by default as in the `as.xcf` file to "common", images are therefore sought in the `appdata/public/common` directory.

```
<RESOURCE Id="res.public.resources" Source="INTERNAL">common</RESOURCE>
```

Usage example 2

```
... 
<UA_OUTPUT>
  <PROXY></PROXY>
  <PUBLIC_IMAGEPATH>myapp/newpictures</PUBLIC_IMAGEPATH>
  <GWC-JS>gwc-js</GWC-JS>
  <TIMEOUT Using="cpn.wa.timeout"/>
</UA_OUTPUT>
...
```

In this example, the value of the `PUBLIC_IMAGEPATH` is set to "myapp/newpictures", so images are therefore sought in the `appdata/public/myapp/newpictures` directory.

Child elements

There are no child elements.

Parent elements

This element is a child of one of the following elements: `UA_OUTPUT` on page 414

`RAW_DATA`

The `RAW_DATA` element limits the size of a single log message.

Log messages can include the complete html response, and can therefore be very large (an entire html page). The optional `RAW_DATA` element specifies the maximum number of characters in any single log message. It takes an attribute `MaxLength`, the number of characters after which the log message is truncated. The value provided for `MaxLength` must be a non-negative integer.

If the `RAW_DATA` element is omitted, data is logged in its entirety.

Example

```
<RAW_DATA MaxLength="100" />
```
**Parent elements**
This element is a child of one of the following elements: LOG on page 383

**RENDERING**

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

The RENDERING element defines the rendering to be applied to this Web application. It takes an optional attribute Using, in which the unique identifier of a predefined WEB_APPLICATION_RENDERING_COMPONENT element can be specified.

The RENDERING element may contain an OUTPUT_DRIVER on page 389 child element, specifying the output driver to be used. If an output driver is defined here, it overrides any output driver settings inherited via the Usage attribute and its specified rendering component.

**Child elements**
The RENDERING element may contain the following child elements:

1. Zero or one OUTPUT_DRIVER element.
2. Zero or one XML_DECLARATION element.
3. Zero or one HTTP_RESPONSE_ENCODING element.
4. Zero or one HTTP_REQUEST_ENCODING element.
5. Zero or one MIME_TYPE element.
6. Zero or one DOC_TYPE element.

**Usage examples**

```
<RENDERING Using="cpn.rendering.wa" />
```

```
<RENDERING>
  <OUTPUT_DRIVER>JFE36</OUTPUT_DRIVER>
</RENDERING>
```

**Parent elements**
This element is a child of one of the following elements: MAP on page 384

**REPORT_VIEWER_DIRECTORY**
The REPORT_VIEWER_DIRECTORY element specifies the report viewer directory used by the Genero Web Report Viewer.

The REPORT_VIEWER_DIRECTORY allows you to configure the location of the Genero Web Report Viewer, where report viewer files may be accessed. A corresponding report viewer URL prefix, /ua/report/viewer, is provided to the GRE to load the report viewer implementation, see Application URIs on page 51.

**Syntax**

```
<REPORT_VIEWER_DIRECTORY>path</REPORT_VIEWER_DIRECTORY>
```
The `REPORT_VIEWER_DIRECTORY` element does not support any attributes or have any child elements.

**Usage example**

```xml
<REPORT_VIEWER_DIRECTORY>${res.gredir}/viewer</REPORT_VIEWER_DIRECTORY>
...
```

**Parent elements**
This element is a child of the following element: `INTERFACE_TO_CONNECTOR` on page 381

**REQUEST_RESULT (for an application)**
Time out (in seconds) for pending transactions.

The Request Result timeout is used to inform the user when a transaction is taking longer than expected. The `REQUEST_RESULT` element specifies the number of seconds to wait for the DVM to give an answer to the Application Server, after which the Application Server sends a "transaction pending" page to the front-end to inform the user that this transaction is taking longer than expected. This is also known as sending a keepalive response. The default transaction pending page automatically submits a new request to wait for the DVM to complete its processing.

Under normal operations, the front-end client sends a GET request to the Genero Application Server immediately after receiving a response. Meanwhile, the Genero Application Server stores data sent by the DVM for the application in its buffer, waiting for a GET request from the client. When the GET request is received by the Genero Application Server, if the server has data sent by the DVM in its buffer, the stored data is sent back to the front-end. If the DVM does not have data to send, the Genero Application Server waits and, if the DVM is still processing the request after the specified `REQUEST_RESULT` timeout expires, it sends the keepalive response to the front-end client and resets the `REQUEST_RESULT` timer.

**Tip:** The number of seconds specified for the `REQUEST_RESULT` timeout should be less that the cgi timeout. By default, the Apache Web server has the cgi timeout default to 300 seconds. Therefore, the `REQUEST_RESULT` timeout has an initial default setting of 60 seconds.

**Usage example**

```xml
<REQUEST_RESULT>60</REQUEST_RESULT>
```

In this usage example, the Request Result timeout is set to 60 seconds.

**Parent elements**
This element is a child of one of the following elements:

- `WEB_APPLICATION_TIMEOUT_COMPONENT` on page 421
- `TIMEOUT`

**REQUEST_RESULT (for a service)**
Time out (in seconds) for pending transactions.

The Request Result timeout is used to inform the user when a Web service transaction is taking longer than expected. The `REQUEST_RESULT` element specifies the number of seconds to wait for the DVM to give an answer to the Application Server, after which the Application Server sends a HTTP 400 error page to the front-end to inform the user that the request has taken too long to fulfill.

The front-end client cannot recover from a HTTP 400 error page, and any Web service client application must send a new request.
Tip: The number of seconds specified for the REQUEST_RESULT timeout should be less than the cgi timeout. By default, the Apache Web server has the cgi timeout default of 300 seconds. If setting the REQUEST_RESULT timeout, it should be set to less than that, for example, at 60 seconds.

Usage example

```xml
<REQUEST_RESULT>60</REQUEST_RESULT>
```

In this usage example, the Request Result timeout is set to 60 seconds. It tells the GWS proxy to release DVM in charge of a service that has not responded in the given time frame.

Note: If Request Result timeout is not set (the default), the GWS proxy never releases the DVM and will wait until DVM responds to the request.

Parent elements
This element is a child of one of the following elements:

- SERVICE_APPLICATION_TIMEOUT_COMPONENT on page 403
- TIMEOUT

RESOURCE
This element defines a variable you can define once and use elsewhere in your Genero Application Server and external application configuration files.

Resources allow you to create resources, or variables, for use within the configuration files and templates. Resources can be defined in the general RESOURCE_LIST element or within individual application or Web service configurations.

A RESOURCE element defines a resource, or variable, that can be used in configuration files and template files. It takes two attributes, an Id attribute and a Source attribute. The Id attribute is the identifier of the resource itself, while the Source attribute tells the Application Server where to find the value of the resource.

Syntax

```xml
<RESOURCE Id="resId" Source="INTERNAL | ENVIRON" > resData </RESOURCE>
```

Syntax notes

1. resId is the resource identifier
2. resData is the resource data. Its use depends on the value of the Source attribute.
   - If Source is INTERNAL, resData is the value of the resource.
   - If Source is ENVIRON, resData is the name of an environment variable.
3. Resources are used in the configuration files or in the template files using the syntax:

```bash
$(resId)
```

Usage examples

A resource defined inline.

```xml
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">$(res.fgldir)/bin/fglrun.exe</RESOURCE>
```
A resource defined as the value of an environment variable. In this example, the resource res.os contains the value of the environment variable OS. For example, on a Windows™ system, the environment variable OS could have the value Windows_NT.

<RESOURCE Id="res.os" Source="ENVIRON">OS</RESOURCE>

**Parent elements**
This element is a child of one of the following elements: PLATFORM_INDEPENDENT on page 395, UNX on page 416, WNT on page 423

**RESOURCE (for a service)**
The RESOURCE element defines a resource available for this application. For more information, see Resources.

**Parent elements**
This element is a child of one of the following elements: APPLICATION (for a service) on page 356

**RESOURCE (for an application)**
The RESOURCE element defines a resource available for this application. For more information on defining resources, see RESOURCE on page 400.

**Parent elements**
This element is a child of one of the following elements: APPLICATION (for an application) on page 354

**RESOURCE_LIST**
This element contains all RESOURCE elements, organized by operating system.

The RESOURCE_LIST element of the Genero Application Server configuration file allows you to define RESOURCE elements, which can then be referenced in your configuration files and template files. A resource is a type of variable. By defining and using resources, when the value of the resource needs updating, it becomes possible to modify the resource in one location, and the new value is carried through the various configuration and template files that reference the resource.

A resource is defined as platform-independent or platform-dependent, based on the section (parent element) in which the resource is defined.

**Syntax**

```
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT> [ resource ] [ ... ] </PLATFORM_INDEPENDENT>
  <WNT> resource [ ... ] </WNT>
  <UNIX> resource [ ... ] </UNIX>
</RESOURCE_LIST>
```

**Child elements**
A RESOURCE_LIST element contains the following child elements:

- One PLATFORM_INDEPENDENT element, containing a list of platform-independent resources.
- One WNT element, containing a list of WNT-specific resources.
- One UNIX element, containing a list of UNIX-specific resources.
Example

```xml
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT>
    <RESOURCE Id="res.fglgui" Source="INTERNAL">1</RESOURCE>
    ...
  </PLATFORM_INDEPENDENT>
  <WNT>
    <RESOURCE Id="res.dvm.wa" Source="INTERNAL">
      $(res.fgldir)\bin\fglrun.exe</RESOURCE>
    ...
  </WNT>
  <UNIX>
    <RESOURCE Id="res.dvm.wa" Source="INTERNAL">
      $(res.fgldir)/bin/fglrun.exe</RESOURCE>
    ...
  </UNIX>
</RESOURCE_LIST>
```

For more information on defining a resource, see RESOURCE on page 400.

Parent elements
This element is a child of one of the following elements: APPLICATION_SERVER on page 357

ROOT_URL_PREFIX
The ROOT_URL_PREFIX element specifies the URL to access the Web server when a reverse proxy server is used between the client and the GAS.

The ROOT_URL_PREFIX will override the URLs generated by the Web server and will construct them using this prefix value instead.

Note: The reverse proxy server works on behalf of the Application server. The Web client is not aware of the proxy and does not know or see what server it is being forwarded to behind the proxy. In this case, the Web server uses the ROOT_URL_PREFIX, which provides the correct interface to the client.

Syntax

```
[<ROOT_URL_PREFIX>URL_Web_Server_behind_proxy</ROOT_URL_PREFIX>] [...] 
```

Usage example

```xml
<ROOT_URL_PREFIX>http://serverA:8080/java-j2eedispatch</ROOT_URL_PREFIX>
```

• Where a reverse proxy server (e.g. server A) is forwarding requests to the GAS on serverB.
• Where "java-j2eedispatch" specifies the connector.uri part of the URI; typically this is the same as the connector.uri acknowledged by the GAS for the dispatcher specific to that Web server.

Note: If ROOT_URL_PREFIX is defined and is empty, it behaves as if not defined.

Parent elements
This element is a child of the following element:

• INTERFACE_TO_CONNECTOR on page 381
The SERVICE_APPLICATION_EXECUTION_COMPONENT creates a Web service execution component, which defines a set of execution parameters that are used when starting the Web service. It takes an attribute Id, which specifies the unique identifier for this set of execution definitions. It is this unique identifier that is referenced by a Web service, providing that Web service with a base set of execution parameters.

**Syntax**

```xml
<SERVICE_APPLICATION_EXECUTION_COMPONENT Id="compId">
  <ENVIRONMENT_VARIABLE Id="envId">env</ENVIRONMENT_VARIABLE> [...]
  | <PATH> path </PATH> |
  | <DVM> dvm </DVM> |
  | <MODULE> module </MODULE> |
  | <PARAMETERS> parameterSettings </PARAMETERS> |
  | <ACCESS_CONTROL> accessSettings </ACCESS_CONTROL> |
  | <DELEGATE> delegateSettings </DELEGATE> |
  | <WEB_COMPONENT_DIRECTORY> webcomponentsSettings </WEB_COMPONENT_DIRECTORY> |
  | <POOL> poolSettings </POOL> |
</SERVICE_APPLICATION_EXECUTION_COMPONENT>
```

**Child elements**
The SERVICE_APPLICATION_EXECUTION_COMPONENT element may contain the following child elements:

1. Zero or more ENVIRONMENT_VARIABLE elements.
2. Zero or one PATH element.
3. Zero or one DVM element.
4. Zero or one MODULE element.
5. Zero or one PARAMETERS element.
6. Zero or one ACCESS_CONTROL element.
7. Zero or one DELEGATE element.
8. Zero or one WEB_COMPONENT_DIRECTORY element.
9. Zero or one POOL element.

**Example**

```xml
<SERVICE_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="INFORMIXDIR">$(res.informixdir)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="INFORMIXSERVER">$(res.informixserver)</ENVIRONMENT_VARIABLE>
  ...  
  <DVM>$(res.dvm.wa)</DVM>
</SERVICE_APPLICATION_EXECUTION_COMPONENT>
```

**Parent elements**
This element is a child of one of the following elements: COMPONENT_LIST on page 363

**SERVICE_APPLICATION_TIMEOUT_COMPONENT**
The SERVICE_APPLICATION_TIMEOUT_COMPONENT element creates a Web service application timeout component, which defines a set of timeout values to be used when configuring a Web service. It takes an attribute Id, which specifies the unique identifier for this set of timeout definitions. It is this unique identifier that is referenced by a Web service, providing that Web service with a base set of timeout values.
The GAS handles the Web Services Server side. It takes care of the DVM requested by a Web Services client.

**Syntax**

```xml
<SERVICE_APPLICATION_TIMEOUT_COMPONENT Id="sTimeOutID">
    <DVM_AVAILABLE>dvmTimeOut</DVM_AVAILABLE>
    <KEEP_ALIVE>dvmKeepAliveTimeOut</KEEP_ALIVE>
    <REQUEST_RESULT>requestTimeOut</REQUEST_RESULT>
</SERVICE_APPLICATION_TIMEOUT_COMPONENT>
```

**Child elements**

The `SERVICE_APPLICATION_TIMEOUT_COMPONENT` element may contain the following child elements.

1. Zero or one `DVM_AVAILABLE` element.
2. Zero or one `KEEP_ALIVE` element.
3. Zero or one `REQUEST_RESULT` element.

**Usage example**

```xml
<SERVICE_APPLICATION_TIMEOUT_COMPONENT Id="cpn.ws.timeout.set1">
    <DVM_AVAILABLE>10</DVM_AVAILABLE>
    <KEEP_ALIVE>360</KEEP_ALIVE>
    <REQUEST_RESULT>60</REQUEST_RESULT>
</SERVICE_APPLICATION_TIMEOUT_COMPONENT>
```

In this example, the `Id` value - `cpn.ws.timeout.set1` - can be referenced when defining a Web service. When a Web service references a component by its `Id` value, it inherits the settings defined by that component.

**Parent elements**

This element is a child of one of the following elements: `COMPONENT_LIST`.

**SERVICE (for HTTP)**

This element contains a list of those `HEADER` elements defining the communication carried on the HTTP protocol between Web services and the client.

Starting with GAS 3.00 you can set custom HTTP headers for Web applications and Web services. This configuration takes place in the `HTTP` element of the `INTERFACE_TO CONNECTOR` element of the GAS configuration file.

**Syntax**

```xml
<SERVICE>
    [<HEADER Name="headerName">value; ...</HEADER>]
</SERVICE>
```

**Child elements**

The `SERVICE` element may contain zero to many `HEADER` elements.

**Example**

```xml
<INTERFACE_TO_CONNECTOR>
    ...
    <HTTP>
```

<SESSION_COOKIE Secure="true"> </SESSION_COOKIE>
<APPLICATION>
  <HEADER/>
</APPLICATION>
<SERVICE>
  <HEADER/>
</SERVICE>
</HTTP>
</INTERFACE_TO_CONNECTOR>

Parent elements
This element is a child of the HTTP on page 378 element.

SERVICE_LIST
The SERVICE_LIST element provides a list of groups and Web services applications (for those Web services applications defined within the Genero Application Server configuration file).

Syntax

```xml
<SERVICE_LIST>
  | <GROUP ...> | ... | | | <APPLICATION ...> | ... | | |
</SERVICE_LIST>
```

Child elements
The SERVICE_LIST element may contain the following child elements:
1. Zero or more GROUP elements.
2. Zero or more APPLICATION elements.

Example

```xml
<CONFIGURATION>
  <APPLICATION_SERVER>
    ...
    <SERVICE_LIST>
      ...
    </SERVICE_LIST>
    ...
  </APPLICATION_SERVER>
</CONFIGURATION>
```

**Important:** You must include the SERVICE_LIST element, even if the Genero Application Server does not have any Web Services to define. In this situation, you simply specify an empty SERVICE_LIST element.

Parent elements
This element is a child of one of the following elements: APPLICATION_SERVER on page 357

SESSION_DIRECTORY
The SESSION_DIRECTORY element specifies where to store the session files of all applications and services started by the GAS.

This element is optional.
Usage example

```xml
<SESSION_DIRECTORY>/var/tmp</SESSION_DIRECTORY>
```

In this example, the session files are stored in sub directories of /var/tmp/session.

By default, the `SESSION_DIRECTORY` entry is not set and the session files are stored in FGLASDIR.

Parent elements
This element is a child of `INTERFACE_TO_CONNECTOR` on page 381.

**SESSION_COOKIE**
The `SESSION_COOKIE` element specifies that cookies the GAS uses to validate the session are created with the secure flag.

The main goal of cookies is to keep a state, using session variables, between two runs of an application by the same user. With the `SESSION_COOKIE` element you can specify that Genero session identifier cookies, as well as cookies created by Sticky Web services (see Configure sticky Web services on page 254), are created with the secure flag.

Syntax

```xml
<SESSION_COOKIE Secure="TRUE|FALSE"/>
```

Syntax notes
The `SESSION_COOKIE` element takes a mandatory `Secure` attribute. When set to `TRUE`, the cookie is restricted to secured connections (HTTPS) only. Valid values are `TRUE` or `FALSE`. The default is `FALSE`.

Example secure session cookie

```xml
<INTERFACE_TO_CONNECTOR>
  ...
  <HTTP>
    <SESSION_COOKIE Secure="TRUE"/>
  </SESSION_COOKIE>
  ...
</HTTP>
</INTERFACE_TO_CONNECTOR>
```

Secure session cookie in web-xml

**Note:** If using a Java Enterprise Edition (Java EE) server, the secure flag needs to be configured in the `web.xml` file as shown in the example:

```xml
<web-app>
  ...
  <session-config>
    <cookie-config>
      <secure>true</secure>
    </cookie-config>
  </session-config>
</web-app>
```
Child elements
There are no child elements.

Parent elements
This element is a child of the HTTP element.

SHORT
The SHORT element contains the short description to be associated with an application definition.

Usage example

```
<SHORT>A short description</SHORT>
```

Parent elements
This element is a child of the DESCRIPTION element.

SHORTCUT
Selects the template to be used to generate the GDC shortcut for this application.

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

Syntax

```
<SHORTCUT Id="shortcutId"> shortcutPath </SHORTCUT>
```

Syntax notes
1. `shortcutId` is the unique identifier for this element.
2. `shortcutPath` is the path to the shortcut template to be used in this theme.

Usage example

```
<WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.default.gdc">
  <TEMPLATE Id="_default">$(res.theme.default.gdc.template)</TEMPLATE>
  <SHORTCUT Id="_default">$(res.theme.default.gdc.shortcut.path)/gdc-http.gdc</SHORTCUT>
</WEB_APPLICATION_THEME_COMPONENT>
```

Parent elements
This element is a child of one of the following elements:

- THEME on page 411
- WEB_APPLICATION_THEME_COMPONENT on page 420
SNIPPET

The **SNIPPET** element associates a snippet object identifier with a template to be used by the Genero Web Client snippet-based rendering engine when rendering forms that include the specified object.

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the `wa` protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use Genero Browser Client, see *Genero Web Client for JavaScript (GWC-JS)* on page 184.

Syntax

```
<SNIPPET Id="snipId" Style="mystyle"> snippath </SNIPPET>
```

**Syntax notes**

1. `snipId` is the object identifier
2. `mystyle` is the value of the STYLE attribute
3. `snippath` is the path to the template snippet file

**Usage Example**

```
<SNIPPET Id="Edit" Style="FileUpload">
 $(res.path.tpl.html5)/FileUpload.xhtml</SNIPPET>
```

**Code Example**

```
EDIT f01 = formonly.f01, STYLE="FileUpload";
```

For more information, refer to the *Genero Web Client User Guide*.

**Parent elements**

This element is a child of one of the following elements: **THEME** on page 411 **WEB_APPLICATION_THEME_COMPONENT** on page 420

SOCKET_FAMILY

By default, the Genero Application Server uses Linux®/UNIX™ domain sockets to communicate between the dispatcher and the proxies on Linux®/UNIX™ systems. The Genero Application Server uses TCP sockets to communicate between the dispatcher and proxies on Windows® and with the J2EE dispatcher.

The **SOCKET_FAMILY** element specifies whether Linux®/UNIX™ domain sockets or TCP sockets are to be used on Linux®/UNIX™ systems. The configured default is Linux®/UNIX™. To use TCP sockets, change the value to TCP.

**Caution:** Forcing TCP sockets on UNIX™ should only be done for debugging purposes, at the request of support.

On Windows®, it is configured to TCP, but it is not used. It must exist, however, as it is required by the schema validating the file.
Usage example

```xml
<RESOURCE_LIST>
  <WNT>
    <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">TCP</RESOURCE>
    <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">C:\temp</RESOURCE>
  </WNT>
  <UNIX>
    <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">UNIX</RESOURCE>
    <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">/tmp</RESOURCE>
  </UNIX>
  ...
  <INTERFACE_TO_CONNECTOR>
    <TCP_BASE_PORT>$(res.ic.base.port)</TCP_BASE_PORT>
    <TCP_PORT_OFFSET>$(res.ic.port.offset)</TCP_PORT_OFFSET>
    <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
    <TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>
    <SOCKET_FAMILY>$(res.dispatcher.socket.family)</SOCKET_FAMILY>
    <SOCKET_PATH>$(res.dispatcher.socket.path)</SOCKET_PATH>
  </INTERFACE_TO_CONNECTOR>
</RESOURCE_LIST>
```

**Parent elements**

This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 381

---

**SOCKET_PATH**

The SOCKET_PATH element defines the directory where Linux®/UNIX™ domain sockets will be created and stored. By default, it is configured as /tmp for Linux®/UNIX™ systems, C:\temp for Windows®

**Usage example**

```xml
<RESOURCE_LIST>
  <WNT>
    <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">TCP</RESOURCE>
    <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">C:\temp</RESOURCE>
  </WNT>
  ...
  <INTERFACE_TO_CONNECTOR>
    <TCP_BASE_PORT>$(res.ic.base.port)</TCP_BASE_PORT>
    <TCP_PORT_OFFSET>$(res.ic.port.offset)</TCP_PORT_OFFSET>
    <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
    <TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>
    <SOCKET_FAMILY>$(res.dispatcher.socket.family)</SOCKET_FAMILY>
    <SOCKET_PATH>$(res.dispatcher.socket.path)</SOCKET_PATH>
  </INTERFACE_TO_CONNECTOR>
</RESOURCE_LIST>
```

**Parent elements**

This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 381
The **START** element specifies the number of DVMs to start for Web Services when the GAS starts.

**Constraint**

\[ \text{START} \leq \text{MAX\_AVAILABLE} \]

**Parent elements**

This element is a child of one of the following elements: [POOL](#) on page 395

**SYSTEM\_ID**

The **SYSTEM\_ID** element is the system identifier of the DTD.

**Usage example**

```xml
<SYSTEM\_ID>http://www.w3.org/TR/html4/strict.dtd</SYSTEM\_ID>
```

**Parent elements**

This element is a child of one of the following elements: [DOC\_TYPE](#) on page 366

**TCP\_BASE\_PORT**

The **TCP\_BASE\_PORT** element specifies the base value of the port the Genero Application Server is listening.

The true port that the Genero Application Server is listening to is the port specified by **TCP\_BASE\_PORT** + **TCP\_PORT\_OFFSET**.

**Usage example**

```xml
<TCP\_BASE\_PORT>6420</TCP\_BASE\_PORT>
```

**Parent elements**

This element is a child of [INTERFACE\_TO\_CONNECTOR](#) on page 381.

**TCP\_PORT\_OFFSET**

The **TCP\_PORT\_OFFSET** element specifies the offset value of the port the GAS is listening.

The true port that the Genero Application Server is listening to is the port specified by **TCP\_BASE\_PORT** + **TCP\_PORT\_OFFSET**.

**Usage example**

```xml
<TCP\_PORT\_OFFSET>75</TCP\_PORT\_OFFSET>
```

**Parent elements**

This element is a child of [INTERFACE\_TO\_CONNECTOR](#) on page 381.

**TEMPLATE**

The **TEMPLATE** element identifies the main template associated with the application. Its identifier is ".default".

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA\_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and
snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

Syntax

<TEMPLATE Id="_default"> templatePath </TEMPLATE>

Syntax notes

1. templatePath is the path to the template file

Usage example

<TEMPLATE Id="_default">$(res.path.tpl.html5)/main.xhtml</TEMPLATE>

Parent elements

This element is a child of one of the following elements: THEME on page 411 WEB_APPLICATION_THEME_COMPONENT on page 420

TEMPORARY_DIRECTORY

The TEMPORARY_DIRECTORY element specifies where to store files transferred between the front-end and the application server host.

Usage example

<TEMPORARY_DIRECTORY>/var/tmp</TEMPORARY_DIRECTORY>

In this example, the transferred files are stored in /var/tmp.

Parent elements

This element is a child of INTERFACE_TO_CONNECTOR on page 381.

THEME

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.

The THEME element defines the theme to be applied to the application. It takes an optional attribute Using, in which the unique identifier of a predefined WEB_APPLICATION_THEME_COMPONENT element can be specified.

The THEME element may contain, TEMPLATE on page 410, SHORTCUT on page 407, and SNIPPET on page 408 child elements, specifying the various templates and snippets to be used. You can specify multiple theme elements within an application, as different themes can be called by different windows and/or forms. If a template defined in this THEME element has the same unique identifier as a template inherited via a WEB_APPLICATION_THEME_COMPONENT setting, the local THEME element is used. In other words, templates defined explicitly for the application override any templates defined in the WEB_APPLICATION_THEME_COMPONENT that have the same template identifier.
Child elements

The **THEME** may contain the following elements:

1. Zero or one **TEMPLATE** element.
2. Zero or one **SHORTCUT** element.
3. Zero or more **SNIPPET** elements.

Usage examples

```
<THEME Using="cpn.theme.default.gwc" />
```

```
<THEME Using="cpn.theme.default.gwc">
  <TEMPLATE Id="_default">/templatedir/deftemp.html</TEMPLATE>
</THEME>
```

For more information, see *Defining a Theme Component*.

Parent elements

This element is a child of one of the following elements: **MAP** on page 384

**TIMEOUT** (for a file transfer)

The **TIMEOUT** element specifies the number of seconds that transferred files are available after application end and before they are deleted. The duration is always specified in seconds.

Usage example

```
<TIMEOUT>600</TIMEOUT>
```

By default, the timeout duration is set to 600 seconds.

Parent elements

This element is a child of one of the following elements: **FILE_TRANSFER** on page 372

**TIMEOUT** (for an application)

The **TIMEOUT** element sets the timeouts for the application. You can reference a predefined **WEB_APPLICATION_TIMEOUT_COMPONENT** to inherit the timeout settings of that component by including the **Using** attribute, specifying the unique identifier for that timeout component, and/or you can set individual timeout elements specific to the application.

Settings defined locally within the **TIMEOUT** element override settings defined in included timeout components.

Child elements

The **TIMEOUT** element may contain the following child elements:

1. Zero or one **USER_AGENT** element.
2. Zero or one **REQUEST_RESULT** (for an application) on page 399 element.
3. Zero or one **DVM_AVAILABLE** element.
4. Zero or one **DVM_PINGTIMEOUT** element.

For more information, see **WEB_APPLICATION_TIMEOUT_COMPONENT**.
Usage examples

```xml
<TIMEOUT Using="cpn.wa.timeout.set1" />
```

```xml
<TIMEOUT>
   <USER_AGENT>300</USER_AGENT>
   <REQUEST_RESULT>240</REQUEST_RESULT>
   <DVM_AVAILABLE>10</DVM_AVAILABLE>
</TIMEOUT>
```

Parent elements

This element is a child of one of the following elements: UA_OUTPUT on page 414, MAP on page 384

**TIMEOUT (for a service)**

The **TIMEOUT** element sets the timeouts for the Web services application. You can reference a predefined SERVICE_APPLICATION_TIMEOUT_COMPONENT to inherit the timeout settings of that component by including the **Using** attribute, specifying the unique identifier for that timeout component, and/or you can set individual timeout elements specific to the application.

Settings defined locally within the **TIMEOUT** element override those settings defined in a referenced SERVICE_APPLICATION_TIMEOUT_COMPONENT.

Child elements

Possible timeout elements include:

1. Zero or one **DVM_AVAILABLE** element.
2. Zero or one **KEEP_ALIVE** element.
3. Zero or one **REQUEST_RESULT (for a service)** on page 399 element.

For more information on setting timeout parameters, refer to **TIMEOUT (for an application)** on page 412.

Usage examples

```xml
<TIMEOUT Using="cpn.ws.timeout.set1" />
```

```xml
<TIMEOUT>
   <DVM_AVAILABLE>10</DVM_AVAILABLE>
   <KEEP_ALIVE>240</KEEP_ALIVE>
   <REQUEST_RESULT>60</REQUEST_RESULT>
</TIMEOUT>
```

Parent elements

This element is a child of one of the following elements: APPLICATION (for a service) on page 356

**TIMEOUT (for auto logout)**

The **TIMEOUT** element specifies how long (in seconds) the DVM waits from when it detects an application has no user activity before it triggers an auto logout event.

After this timeout period elapses, the front-end program gets a log out message, see **AUTO_LOGOUT** on page 358.
Syntax

```xml
<TIMEOUT>timeoutSeconds</TIMEOUT>
```

Usage example

```xml
<TIMEOUT>0</TIMEOUT>
```

A timeout duration set to 0 seconds means the auto logout is ignored and the application keeps running. A correct configuration requires that the `TIMEOUT` be set as required to enable auto logout when there is no user activity.

Parent elements

This element is a child of the following element: `AUTO_LOGOUT_COMPONENT` on page 359

**UA_OUTPUT**

The `UA_OUTPUT` element specifies the configuration parameters for an application delivered by the UA proxy.

Starting with Genero 3.00, all UI applications delivered by the Genero Application Server are rendered by the UA proxy, to include the Genero Desktop Client (GDC), Genero Web Client for JavaScript (GWC-JS), Genero Mobile for Android and Genero Mobile for iOS. These applications are configured in part by the `UA_OUTPUT` element.

Syntax

```xml
<UA_OUTPUT [Using=component-id]>
  <PROXY>executable-name</PROXY>
  <PUBLIC_IMAGEPATH>image-path</PUBLIC_IMAGEPATH>
  [<GWC-JS>gwc-js-directory</GWC-JS>]
  [<TIMEOUT>...</TIMEOUT>]
</UA_OUTPUT>
```

Child elements

The `UA_OUTPUT` element may contain the following child elements:

1. One `PROXY (for an application)` on page 396 element.
2. One `PUBLIC_IMAGEPATH` on page 396 element.
3. Zero or one `GWC-JS` on page 374 element.
4. Zero or one `TIMEOUT (for an application)` on page 412 element.

Usage example

```xml
<UA_OUTPUT>
  <PROXY>$(res.uaproxy.cmd)</PROXY>
  <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
  <GWC-JS>gwc-js</GWC-JS>
  <TIMEOUT Using="cpn.wa.timeout"/>
</UA_OUTPUT>
```

Parent elements

This element is a child of one of the following elements: `APPLICATION (for an application)` on page 354
**UA_OUTPUT_COMPONENT**

The **UA_OUTPUT_COMPONENT** element sets configuration for **UA_OUTPUT** that define settings for an application delivered by the UA proxy.

**Syntax**

```xml
<UA_OUTPUT_COMPONENT Id=component-id >
  <PROXY>...</PROXY>
  <PUBLIC_IMAGEPATH>...</PUBLIC_IMAGEPATH>
  [<TIMEOUT>...</TIMEOUT>]
  [<GWC-JS>...</GWC-JS>]
</UA_OUTPUT_COMPONENT>
```

Attributes for this element include:

1. **component-id** is the unique identifier for this set of UA definitions.

**Child elements**

The **UA_OUTPUT_COMPONENT** element may contain the following child elements:

1. One **PROXY** (for an application) on page 396 element.
2. One **PUBLIC_IMAGEPATH** on page 396 element.
3. Zero or one **TIMEOUT** (for an application) on page 412 element.
4. Zero or one **GWC-JS** on page 374 element.

**Usage**

You use this element to provide your application with a base set of UA parameters. UA components provide instructions on the resources used by the application; such as the proxy, the GWC-JS, timeouts, or path to public images.

This element is defined within the **COMPONENT_LIST** element of the GAS configuration file.

**Example GAS UA component definition**

```xml
<UA_OUTPUT_COMPONENT Id="cpn.wa.output">
    <PROXY>$(res.uaproxy.cmd)</PROXY>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <TIMEOUT Using="cpn.wa.timeout"/>
    <GWC-JS>$(res.gbc)</GWC-JS>
</UA_OUTPUT_COMPONENT>
```

The **UA_OUTPUT** element of the default web application (**defaultwa**) is defined using this component.

```xml
<APPLICATION Id="defaultwa" Abstract="TRUE">
    <EXECUTION Using="cpn.wa.execution.local"/>
    <AUTO_LOGOUT Using="cpn.wa.autologout"/>
    <UA_OUTPUT Using="cpn.wa.output"/>
</APPLICATION>
```

**Sample application xcf**

This example shows a typical external application configuration. The default Web application configuration **Parent="defaultwa"** is referenced in the **APPLICATION** element. Through this the application inherits an implicit set of default UA resources for the application.
The only elements explicitly defined for the UA_OUTPUT element is `<GWC-JS>myGwcJs</GWC-JS>` . This specifies a specific GWC-JS that overrides the inherited one.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Parent="defaultwa"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
    <RESOURCE Id="res.public.webcomponents" Source="INTERNAL">deployment/$(/res.deployment.basename)</RESOURCE>
    <RESOURCE Id="res.public.resources" Source="INTERNAL">deployment/$(/res.deployment.basename)
        $(/res.path.separator)common</RESOURCE>
    <RESOURCE Id="res.deployment.basename" Source="INTERNAL">ggc-quick-start</RESOURCE>
    <RESOURCE Id="res.deployment.path" Source="INTERNAL">$(/res.deployment.root)/$(/res.deployment.name)</RESOURCE>
    <RESOURCE Id="res.deployment.name" Source="INTERNAL">ggc-quick-start-20181019-101007</RESOURCE>
    <EXECUTION AllowUnsafeSession="TRUE">
        <PATH>$(/res.deployment.path)/</PATH>
        <MODULE>price</MODULE>
    </EXECUTION>
    <UA_OUTPUT>
        <GWC-JS>mygbc</GWC-JS>
    </UA_OUTPUT>
</APPLICATION>
```

Parent elements

This element is a child of the COMPONENT_LIST on page 363 element.

UNIX

This element contains a collection of UNIX™-specific RESOURCE elements.

The UNIX element contains a list of those resources that are only available on UNIX™ operating systems. There is no difference between UNIX™ systems like Linux®, AIX®, HP-UX, Solaris, and so on.

Child elements

A UNIX element may contain the following child elements:

- Zero to many RESOURCE elements.

Parent elements

This element is a child of one of the following elements: RESOURCE_LIST on page 401

USER_AGENT

The USER_AGENT configuration parameter specifies the number of seconds the Genero Application Server is to wait for a client request before assuming that the front-end client has died or that there has been a network failure.

Under normal operation, the front-end sends a GET request to the Genero Application Server immediately after receiving a response. The client-side front-end (CSF) will also send keep-alive requests (/wa/ka) to keep the application alive in the case of user inactivity. If the client has not sent a request to the Genero Application Server before the USER_AGENT timeout expires, the Genero Application Server assumes that the front-end client has died and sends an instruction to the application's DVM to shut down.
With the Genero Desktop Client Active X, the USER_AGENT timeout usually does not expire. When the user closes the application, the DVM handling that application is properly shut down.

The USER_AGENT timeout proves to be particularly useful with the Genero Web Client. As with the other front-ends, when a user properly exits an application, the DVM handling that application is properly shut down. When the user does not properly exit the application, the DVM remains alive even though the front-end has died. This can occur with the Genero Web Client when a user closes the browser instead of properly exiting the application; the front-end client has no mechanism to tell the Genero Application Server that the user has closed his browser. With the USER_AGENT timeout, the Genero Application Server closes the socket to the DVM, which causes the DVM to shut down.

Usage example

```xml
<USER_AGENT>300</USER_AGENT>
```

In this example, the User Agent timeout is set to 300 seconds.

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_TIMEOUT_COMPONENT on page 421, TIMEOUT (for an application) on page 412

VARIABLE

Zero or one VARIABLE. Define a session variable and optionally the initial value.

Syntax

```xml
<VARIABLE Id="varId"> val </VARIABLE>
```

Syntax notes

1. varId is the variable name.
2. val is the variable value.

Example

```xml
<VARIABLE Id="var7" />
<VARIABLE Id="var8">Initial value</VARIABLE>
```

Parent elements

This element is a child of one of the following elements: HTTP_COOKIE on page 378

WEB_APPLICATION_EXECUTION_COMPONENT

The WEB_APPLICATION_EXECUTION_COMPONENT defines a set of execution parameters that are used when starting the Web application. It takes an attribute Id, which specifies the unique identifier for this set of execution definitions. It is this unique identifier that is referenced by an application, providing that application with its set of execution parameters. The attribute AllowUrlParameters defines whether the parameters provided in the request query string should be ignored ("FALSE", default value) or provided to the DVM ("TRUE")

Syntax

```xml
<WEB_APPLICATION_EXECUTION_COMPONENT Id="compId"
   AllowUrlParameters="allowParam" >
   <ENVIRONMENT_VARIABLE Id="envId" > env </ENVIRONMENT_VARIABLE> [...] 
</WEB_APPLICATION_EXECUTION_COMPONENT>
```
Child elements

The \texttt{WEB\_APPLICATION\_EXECUTION\_COMPONENT} element may contain the following child elements:

1. Zero or more \texttt{ENVIRONMENT\_VARIABLE} elements.
2. Zero or one \texttt{PATH} element.
3. Zero or one \texttt{DVM} element.
4. Zero or one \texttt{MODULE} element.
5. Zero or one \texttt{PARAMETERS} element.
6. Zero or one \texttt{ACCESS\_CONTROL} element.
7. Zero or one \texttt{DELEGATE} element.
8. Zero or one \texttt{WEB\_COMPONENT\_DIRECTORY} element.

Example

\begin{verbatim}
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="INFORMIXDIR">$(res.informixdir)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="INFORMIXSERVER">$(res.informixserver)</ENVIRONMENT_VARIABLE>
  ...
  <DVM>$(res.dvm.wa)</DVM>
  ...
  <WEB_COMPONENT_DIRECTORY>$(connector.uri)/ua/i/$(application.path)/webcomponents</WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
\end{verbatim}

Parent elements

This element is a child of one of the following elements: \texttt{COMPONENT\_LIST} on page 363

\texttt{WEB\_APPLICATION\_HTTP\_COOKIES\_COMPONENT}

This element defines persistent session variables or constants.

The \texttt{WEB\_APPLICATION\_HTTP\_COOKIES\_COMPONENT} element defines persistent session variables or constants. It takes an \texttt{Id} attribute, which specifies the unique identifier for this component. It is this unique identifier that is referenced by an application configuration.

Syntax

\begin{verbatim}
<WEB_APPLICATION_HTTP_COOKIES_COMPONENT Id="compId">
  <HTTP_COOKIE Id="cid" [Expires="endTime" | Domain="mydomain" | Secure="TRUE|FALSE" | HttpOnly=""]>
    <VARIABLE Id="varId">val</VARIABLE> ... 
    <CONSTANT Id="cstId">cst</CONSTANT> ...
    <HTTP_COOKIE> ...
  </HTTP_COOKIE> ...
</WEB_APPLICATION_PICTURE_COMPONENT>
\end{verbatim}
Syntax notes

1. *compId* is the unique identifier for this component (required).

Child elements

The **WEB_APPLICATION_HTTP_COOKIES_COMPONENT** element may contain the following child elements:

1. Zero or one **HTTP_COOKIE** elements.

Parent elements

This element is a child of one of the **COMPONENT_LIST** on page 363 element.

**WEB_APPLICATION_PICTURE_COMPONENT**

This element defines how images are served by the GAS.

The **WEB_APPLICATION_PICTURE_COMPONENT** element specifies the directory from which images are served. It takes an *Id* attribute, which specifies the unique identifier for this Picture component. It is this unique identifier that is referenced by an application configuration, providing that application with the location of its image directory or directories.

Syntax

```xml
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
    <PATH Id="pathID" Type="pathType">Path</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Syntax notes

1. **resID** is the unique identifier for this picture component.

Child elements

The **WEB_APPLICATION_PICTURE_COMPONENT** may contain the following child elements:

1. Zero to many **PATH** elements.

Parent elements

This element is a child of one of the following elements: **COMPONENT_LIST** on page 363

**WEB_APPLICATION_RENDERING_COMPONENT**

A rendering component defines how an application is rendered for delivery via the Web to the front-end.

**Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use Genero Browser Client, see **Genero Web Client for JavaScript (GWC-JS)** on page 184.

The **WEB_APPLICATION_RENDERING_COMPONENT** element specifies the output driver and other elements that determine how an application is to be rendered for a Web application (GWC-HTML5). It takes an attribute **Id**, which specifies the unique identifier for this Rendering component. It is this unique identifier that is referenced by an application, specifying the output driver that the application should use.
Syntax

```xml
<WEB_APPLICATION_RENDERING_COMPONENT Id="compId">
  [<OUTPUT_DRIVER> outputDriver </OUTPUT_DRIVER>]
  [<XML_DECLARATION> xmlDecl </XML_DECLARATION>]
  [<HTTP_RESPONSE_ENCODING Source="REQUEST|INLINE">
    httpResponseEnc</HTTP_RESPONSE_ENCODING>
  ]
  [<HTTP_REQUEST_ENCODING Source="REQUEST|INLINE">
    httpRequestEnc</HTTP_REQUEST_ENCODING>
  ]
  [<MIME_TYPE> mimetype </MIME_TYPE>]
  [<DOC_TYPE> doctype </DOC_TYPE>]
</WEB_APPLICATION_RENDERING_COMPONENT>
```

Child elements

The `WEB_APPLICATION_RENDERING_COMPONENT` may contain the following child elements:

1. Zero or one `OUTPUT_DRIVER` element.
2. Zero or one `XML_DECLARATION` element.
3. Zero or one `HTTP_RESPONSE_ENCODING` element.
4. Zero or one `HTTP_REQUEST_ENCODING` element.
5. Zero or one `MIME_TYPE` element.
6. Zero or one `DOC_TYPE` element.

Example

```xml
<WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.gwc">
  <OUTPUT_DRIVER>GWC</OUTPUT.Driver>
  <HTTP_RESPONSE_ENCODING Source="INLINE">ISO-8859-1</HTTP_RESPONSE_ENCODING>
</WEB_APPLICATION_RENDERING_COMPONENT>

<WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.gwc2">
  <OUTPUT_DRIVER>GWC2</OUTPUT.Driver>
  <HTTP_REQUEST_ENCODING Source="REQUEST"/>
  <HTTP_REQUEST_ENCODING Source="REQUEST"/>
  <MIME_TYPE>application/xml</MIME_TYPE>
</WEB_APPLICATION_RENDERING_COMPONENT>
```

Parent elements

This element is a child of one of the following elements: `COMPONENT_LIST` on page 363

`WEB_APPLICATION_THEME_COMPONENT`

This element specifies the theme (set of templates) that the application or set of applications can use.

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the wa protocol did previously. For information on how templates and snippets were used by the front-ends, please refer to the Genero Application Server 2.50 User Guide.

For new development, use Genero Browser Client, see Genero Web Client for JavaScript (GWC-JS) on page 184.
A theme component defines the theme that an application or set of applications can use. A theme is made up of one main template and multiple template snippets that will drive the rendering of snippet-based rendering engine (SBRE) components. The same snippet identifier can appear more than one time, but each one must be associated with an unique style name, so that each template snippet is uniquely identified by the identifier/style couple. If no template snippet exists for the specified style, the default one is used.

The `<WEB_APPLICATION_THEME_COMPONENT>` element specifies the theme (set of templates) that the application or set of applications can use. It takes an attribute `Id`, which specifies the unique identifier for this Template component. It is this unique identifier that is referenced by a Web application to make accessible the list of templates defined therein.

**Syntax**

```xml
<WEB_APPLICATION_THEME_COMPONENT Id="compId" >
  <TEMPLATE Id="_default" > templatePath </TEMPLATE >
  <SHORTCUT Id="_default" > shortcutPath </SHORTCUT >
  <SNIPPET Id="snipId" [ Style="stylename" ] > snippath </SNIPPET>
[...]
</WEB_APPLICATION_THEME_COMPONENT>
```

**Child elements**

The `<WEB_APPLICATION_THEME_COMPONENT>` may contain the following elements:

1. Zero or one `<TEMPLATE>` element.
2. Zero or one `<SHORTCUT>` element.
3. Zero or more `<SNIPPET>` elements.

**Example**

```xml
<WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.html5.gwc">
  <TEMPLATE Id="_default" > $(res.path.tpl.html5)/main.xhtml</TEMPLATE>
  <SNIPPET Id="UIFrame"> $(res.path.tpl.html5)/UIFrame.xhtml</SNIPPET>
  ...
</WEB_APPLICATION_THEME_COMPONENT>
```

**Parent elements**

This element is a child of one of the following elements:

- `<COMPONENT_LIST>` on page 363

**WEB_APPLICATION_TIMEOUT_COMPONENT**

The `<WEB_APPLICATION_TIMEOUT_COMPONENT>` element creates a Web application timeout component, which define a set of timeout values to be used when configuring a Web application. It takes an attribute `Id`, which specifies the unique identifier for this set of timeout definitions. It is this unique identifier that is referenced with an application's configuration, providing that application with a set of timeout values.

Why are Web application timeouts necessary? When a front-end connects to a DVM via the Genero Application Server (GAS), the connection between the front-end client and the GAS is not persistent (although the connection between the GAS and the DVM is persistent). The Genero Application Server needs the timeout settings to determine whether these components have remained alive and that communication can continue between the two.

The front-end can send two types of requests to the DVM: a POST request when sending data to the DVM and a GET request when asking whether there is data to retrieve. The Genero Application Server,
however, cannot send a request to the front-end client because the front-end does not have a public address.

As a result, a request is always initiated by the front-end and the server response is done with the same connection. Between requests, the Genero Application Server stores data sent from the DVM in its buffer and keeps it for the next GET request from the front-end.

Syntax

```
<WEB_APPLICATION_TIMEOUT_COMPONENT Id="appTimeOutID">
  <USER_AGENT> uaTimeOut </USER_AGENT>
  <REQUEST_RESULT> requestTimeOut </REQUEST_RESULT>
  <DVM_AVAILABLE> dvmTimeOut </DVM_AVAILABLE>
  <DVM_PINGTIMEOUT> dvmPingTimeOut </DVM_PINGTIMEOUT>
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

Child elements

The `WEB_APPLICATION_TIMEOUT_COMPONENT` element may contain the following child elements:

1. Zero or one `USER_AGENT` element.
2. Zero or one `REQUEST_RESULT` element.
3. Zero or one `DVM_AVAILABLE` element.
4. Zero or one `DVM_PINGTIMEOUT` element.

Example

```
<WEB_APPLICATION_TIMEOUT_COMPONENT Id="cpn.wa.timeout.set1">
  <USER_AGENT>300</USER_AGENT>
  <REQUEST_RESULT>60</REQUEST_RESULT>
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
  <DVM_PINGTIMEOUT>300</DVM_PINGTIMEOUT>
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

In this example, the Id value - `cpn.wa.timeout.set1` - can be referenced when defining an application. When an application references a component by its Id value, it inherits the settings defined by that component.

Parent elements

This element is a child of one of the following elements: `COMPONENT_LIST` on page 363

WEB_COMPONENT_DIRECTORY

The `WEB_COMPONENT_DIRECTORY` element specifies the path where Web components for an application are located.

Starting with Genero 3.00, this element added to the URL, builds the path used to find a Web component. It defines paths from where Web components are served via the GAS. The `WEB_COMPONENT_DIRECTORY` configuration only applies to Genero Desktop Client (GDC) applications delivered via GAS, and Genero Web Client for JavaScript (GWC-JS).

**Note:** For your legacy GWC-HTML5 applications, the `WEB_COMPONENT_DIRECTORY` configuration entry is ignored. Web components in this case are still required to be located in `FGLASDIR/web/components` as before. For more details on web component usage, see the Genero Business Development Language User Guide.

Syntax

```
<WEB_COMPONENT_DIRECTORY>path;path</WEB_COMPONENT_DIRECTORY>
```
It allows for multiple paths to be specified, the separator used between resource paths is a semi-colon, ';'.

**Usage example**

```xml
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
  ...
  <DELEGATE service="MyGroup/MyDelegateService"> ... </DELEGATE>
  <WEB_COMPONENT_DIRECTORY>$(application.path)/webcomponents;
  $(my.web.components)/static-files/webcomponents</WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

**Important:** Element order. If the `<WEB_COMPONENT_DIRECTORY>` element is present, it must be set in the correct order within the parent element, see [EXECUTION (for an application)](page 369), [EXECUTION (for a service)](page 371), or [WEB_APPLICATION_EXECUTION_COMPONENT](page 417).

**Child elements**

There are no child elements.

**Parent elements**

This element is a child of one of the following elements: [EXECUTION (for an application)](page 369), [EXECUTION (for a service)](page 371), [WEB_APPLICATION_EXECUTION_COMPONENT](page 417)

**WNT**

This element contains a collection of Windows™-specific RESOURCE elements.

The WNT element contains a list of Windows NT™ resources: those resources are only available on the Windows™ operating systems.

**Child elements**

A WNT element main contain the following child elements:

- Zero to many RESOURCE elements.

**Parent elements**

This element is a child of one of the following elements: [RESOURCE_LIST](page 401)

**XML_DECLARATION**

The XML_DECLARATION element specifies if the xml declaration is present or not in the document response. This element only works with `OUTPUT_DRIVER=GWC2`.

**Usage example**

```xml
<XML_DECLARATION>TRUE</XML_DECLARATION>
```

**Parent elements**

This element is a child of one of the following elements: [WEB_APPLICATION_RENDERING_COMPONENT](page 419) [RENDERING](page 398)
Glossary and Acronyms

In this section, many terms and acronyms used throughout this document are briefly defined.
For more details on the various web technology terms and acronyms found on this page, visit www.w3.org.

Table 70: Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS</td>
<td>Cascading style sheets. CSS is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents.</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.</td>
</tr>
<tr>
<td>DUA</td>
<td>Driver User Agent.</td>
</tr>
<tr>
<td>DVM</td>
<td>The Dynamic Virtual Machine or Runtime System that is installed on the Application Server and executes the application program.</td>
</tr>
<tr>
<td>GAD</td>
<td>Genero ADministration Application. This application is accessed by clicking on the Administration tab on the Genero Application Server Welcome Page (demos.html).</td>
</tr>
<tr>
<td>GAS</td>
<td>Genero Application Server. Defined by the computer system that houses the Dynamic Virtual Machine (DVM).</td>
</tr>
<tr>
<td>GDC</td>
<td>Genero Desktop Client.</td>
</tr>
<tr>
<td>GDCAX</td>
<td>Genero Desktop Client / Active X.</td>
</tr>
<tr>
<td>GWC</td>
<td>Genero Web Client. A client technology that renders the application in an HTML Graphical User Interface (browser).</td>
</tr>
<tr>
<td>GWS</td>
<td>Genero Web Services. A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows® applications can talk with Linux®/UNIX™ applications. See also SOA.</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language. An HTML file is a text file containing markup tags. The markup tags tell the Web browser how to display the page.</td>
</tr>
<tr>
<td>JavaScript</td>
<td>JavaScript is a scripting language designed to add interactivity to HTML pages. A JavaScript consists of lines of executable computer code that can be embedded directly into HTML pages. It is an interpreted language, meaning that the scripts execute without preliminary compilation. Most browsers support JavaScript, and anyone can use JavaScript without purchasing a license.</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture. In SOA, autonomous, loosely-coupled and coarse-grained services with well-defined interfaces provide business functionality and can be discovered and accessed through a supportive infrastructure. This allows internal and external system integration as well as the flexible reuse of application logic through the composition of services to support an end-to-end business process.</td>
</tr>
<tr>
<td>SUA</td>
<td>Sending User Agent.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>User Agent</td>
<td>A User Agent is a client agent. It can be a browser or the Genero Desktop Client.</td>
</tr>
<tr>
<td>Web Server</td>
<td>A computer that delivers (serves up) Web pages. Every web server has an IP address and possibly a domain name. For example, if you enter the URL <a href="http://www.mycompany.com">http://www.mycompany.com</a> in your browser, this sends a request to the server whose domain name is mycompany.com. The server then fetches the home page and sends it to your browser. Any computer can be turned into a web server by installing server software and connecting the machine to the Internet.</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language. WSDL is an XML-based language for describing Web services and how to access them.</td>
</tr>
<tr>
<td>XML</td>
<td>Short for Extensible Markup Language, a specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. For more information, please refer to the W3C web site at <a href="http://www.w3.org">www.w3.org</a>.</td>
</tr>
<tr>
<td>XML Schema</td>
<td>XML Schema is an XML-based alternative to a DTD. An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).</td>
</tr>
<tr>
<td>XHTML</td>
<td>EXtensible HyperText Markup Language. XHTML is aimed to replace HTML. XHTML is almost identical to HTML 4.01. XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application.</td>
</tr>
<tr>
<td>XPath</td>
<td>XPath is a language for navigating in XML documents.</td>
</tr>
<tr>
<td>XSD</td>
<td>See XML Schema.</td>
</tr>
<tr>
<td>XSL</td>
<td>XML Style Sheets. XML does not use predefined tags (you can use any tag names you wish), and the meaning of these tags are not well understood. For example, a &lt;table&gt; element could mean an HTML table, a piece of furniture, or something else - and a browser does not know how to display it. XSL describes how the XML document should be displayed.</td>
</tr>
<tr>
<td>XSLT</td>
<td>XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.</td>
</tr>
</tbody>
</table>
Legal Notices

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by CollabNet (http://www.Collab.Net/).

This product includes software developed by the University of California, Berkeley and its contributors.

This product includes software developed or owned by Caldera International, Inc.