Genero Application Server User Guide
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genero Application Server User Guide</td>
<td>8</td>
</tr>
<tr>
<td>What's New in 3.20</td>
<td>9</td>
</tr>
<tr>
<td>Genero Application Server overview</td>
<td>11</td>
</tr>
<tr>
<td>What is the Genero Application Server?</td>
<td>11</td>
</tr>
<tr>
<td>Standalone Genero Application Server</td>
<td>15</td>
</tr>
<tr>
<td>Front-ends and Extensions</td>
<td>16</td>
</tr>
<tr>
<td>GAS Quick Start</td>
<td>16</td>
</tr>
<tr>
<td>Explore Genero application server resources</td>
<td>16</td>
</tr>
<tr>
<td>Quick start guide for applications with UI</td>
<td>17</td>
</tr>
<tr>
<td>Launch first application</td>
<td>17</td>
</tr>
<tr>
<td>Configure an application</td>
<td>21</td>
</tr>
<tr>
<td>Run an application</td>
<td>22</td>
</tr>
<tr>
<td>Deploy an application</td>
<td>23</td>
</tr>
<tr>
<td>Quick start guide for Web services applications</td>
<td>24</td>
</tr>
<tr>
<td>What is Web Service?</td>
<td>24</td>
</tr>
<tr>
<td>Explore Genero demo Web services (server side)</td>
<td>25</td>
</tr>
<tr>
<td>Configure a Web service</td>
<td>26</td>
</tr>
<tr>
<td>Run a client app on Web service using GBC</td>
<td>26</td>
</tr>
<tr>
<td>Deploy a Web service</td>
<td>27</td>
</tr>
<tr>
<td>Genero demo applications</td>
<td>29</td>
</tr>
<tr>
<td>The demo applications</td>
<td>29</td>
</tr>
<tr>
<td>Access demo applications with the Genero Browser Client</td>
<td>29</td>
</tr>
<tr>
<td>GAS Basics</td>
<td>30</td>
</tr>
<tr>
<td>Architecture of the Genero Application Server</td>
<td>30</td>
</tr>
<tr>
<td>Architecture overview</td>
<td>30</td>
</tr>
<tr>
<td>Reliability inherent in the architecture</td>
<td>31</td>
</tr>
<tr>
<td>Development architecture (standalone GAS)</td>
<td>32</td>
</tr>
<tr>
<td>Setting up production environment</td>
<td>32</td>
</tr>
<tr>
<td>Services Pool (GWS Only)</td>
<td>34</td>
</tr>
<tr>
<td>Components of the Genero Application Server</td>
<td>38</td>
</tr>
<tr>
<td>Dispatcher</td>
<td>38</td>
</tr>
<tr>
<td>Proxy</td>
<td>39</td>
</tr>
<tr>
<td>DVM</td>
<td>40</td>
</tr>
<tr>
<td>GAS installation and application data directories</td>
<td>40</td>
</tr>
<tr>
<td>Application environment</td>
<td>44</td>
</tr>
<tr>
<td>Application Web Address</td>
<td>48</td>
</tr>
<tr>
<td>URI Examples</td>
<td>48</td>
</tr>
<tr>
<td>File serving URIs</td>
<td>51</td>
</tr>
<tr>
<td>What is auto logout?</td>
<td>53</td>
</tr>
<tr>
<td>Web application timeouts</td>
<td>54</td>
</tr>
<tr>
<td>Re-login prompt</td>
<td>54</td>
</tr>
</tbody>
</table>
GAS configuration

Managing user access rights................................................................. 61
Validating the installation with the Genero Browser Client................................................. 61
Validate the installation for GBC without a Web server......................................................... 61
Validate the integration for GBC with a Web server................................................................. 62
Validating the Installation with the GDC......................................................................... 62
Validate the installation for GDC without a Web server......................................................... 62
Troubleshooting GAS configuration issues................................................................... 63
Proxy errors on Windows’ platform.............................................................................. 63
Cannot find 127.0.0.1 or localhost on Windows................................................................ 63
What if the application doesn’t start?............................................................................... 63
Error taking session ownership....................................................................................... 64
Licensing examples........................................................................................................... 65
Licensing - base example................................................................................................. 65
Licensing - using the RUN command................................................................................ 66
Licensing - multiple user agents....................................................................................... 66
Licensing - summary case.................................................................................................. 67
Genero front-ends and license counting........................................................................ 67
Licensing tips and tricks / troubleshooting.................................................................... 69

Web Server configuration....................................................................................... 70
ISAPI Extension Installation and Web Server Configuration.............................................. 71
FastCGI Installation and Web Server Configuration.......................................................... 98
Validating configuration files......................................................................................... 108
What is an XML Schema Definition file?................................................................. 109
Why specify the XML Schema Definition file?............................................................. 109
Validating with the gasadmin tool.................................................................................... 109
Selecting an XML editor................................................................................................. 110
Configuring applications on GAS.................................................................................. 111
Application configuration overview................................................................................. 111
Abstract applications...................................................................................................... 111
Create an application group............................................................................................ 112
Application configuration file......................................................................................... 113
Application configuration files for GBC, GWS, and GDC............................................... 114
Configure DVM environment variables......................................................................... 118
Use a script to set the environment................................................................................ 119
Override configuration resources................................................................................... 120
Next steps....................................................................................................................... 121
Troubleshooting application Issues.............................................................................. 121
How to implement delegation......................................................................................... 124
How delegation works................................................................................................. 125
Configure delegation for application or service.............................................................. 127
Optimize delegation with HTTP options........................................................................ 128
From the user agent to the REST service.................................................................... 130
From the REST service to the proxy.............................................................................. 132
Administering the GAS................................................................. 176
  Monitoring...................................................................................... 176
  Usage............................................................................................ 176
  Monitor URL............................................................................... 177
  Find invalid Web service configuration files................................. 180
  Monitor dispatcher logs................................................................. 180
  Monitor proxy logs..................................................................... 181
  Monitor proxies and DVM.............................................................. 182
  Monitor HTTP requests............................................................... 184
  Log files....................................................................................... 185
  Sensitive data in logs.................................................................. 187
  Using the debugger..................................................................... 188
  Run the debugger for the GAS on the Windows® platform....................... 189
  Run the debugger for the GAS on Linux®/UNIX®............................. 189
  Performance tuning..................................................................... 190
  Web server configuration: Keep Alive........................................... 190
  SPDY......................................................................................... 190
  Load balancing........................................................................... 191
  GAS requests............................................................................ 191
  Sessionless request processing.................................................... 192
  Session-bound request processing.............................................. 193
  Load balancing configuration examples....................................... 197

Genero Identity Provider (GIP)..................................................... 204
  Genero Identity Provider scenario................................................. 206
  Genero Identity Platform components......................................... 213
  Setting up the Genero Identity Provider....................................... 215
  Provide access to the openid-connect directory............................. 218
  Configure multiple GAS within one GIP...................................... 218
  Configure for an external GIP..................................................... 220
  Genero Identity Platform StarterApp reference.............................. 222
  Quick Start: Set authorization for group.................................... 224
  Create a user group.................................................................... 225
  Set group authorization scopes................................................. 226
  Deploy and secure an app.......................................................... 226
  Set group scopes for profile API................................................ 228
Deploy private images................................................................. 279
Deploy Web components for GBC.................................................. 279
MANIFEST file............................................................................. 280
DESCRIPTION (for manifest)......................................................... 281
TRIGGERS (for manifest)............................................................ 281
RESOURCES (for manifest)......................................................... 283
APPLICATION (for manifest)....................................................... 284
SERVICE (for manifest)............................................................. 284
Deploying and managing applications with GAR........................................ 285
Genero Archive lifecycle............................................................. 285
Deploy an archive....................................................................... 285
File system layout of deployed archive........................................ 289
List all deployed archives.......................................................... 289
Enable an archive...................................................................... 290
Deactivate (disable) a deployed archive....................................... 292
Undeploy an archive................................................................. 293
Clean up undeployed archives.................................................... 294
Upgrade an archive.................................................................... 295
Upgrade a Web service while the GAS is running.......................... 295
Compile, package, and redeploy after updating software.................. 296
Deploying, enabling, and running applications on GAS.................... 296
Deployment on remote server...................................................... 298

Run Genero Application Server in Docker container......................... 300
Install docker............................................................................. 300
Docker scripts.......................................................................... 301
Docker Quick Start Guide.......................................................... 301
Download installation packages.................................................. 302
Configure license details............................................................. 302
Create image and run docker container....................................... 302
Docker commands.................................................................... 303

Upgrading.................................................................................. 304
New Features of the GAS............................................................ 304
GAS 3.20 new features............................................................... 304
GAS 3.10 new features............................................................... 306
GAS 3.00 new features............................................................... 309
GAS 2.50 new features............................................................... 312
GAS 2.41 new features............................................................... 314
GAS 2.40 new features............................................................... 315
GAS 2.32 new features............................................................... 317
GAS 2.30 new features............................................................... 317
GAS 2.22 new features............................................................... 319
GAS 2.21 new features............................................................... 319
GAS 2.20 new features............................................................... 321
Upgrade Guides for the GAS....................................................... 322
Migrate your configuration files.................................................... 322
GAS 3.20 upgrade guide............................................................. 323
GAS 3.10 upgrade guide............................................................. 325
GAS 3.00 upgrade guide............................................................. 328
GAS 2.50 upgrade guide............................................................. 331
GAS 2.41 upgrade guide............................................................. 332
GAS 2.40 upgrade guide............................................................. 332
GAS 2.30 upgrade guide............................................................. 334
<table>
<thead>
<tr>
<th>Contents</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAS 2.22 upgrade guide</td>
<td>336</td>
</tr>
<tr>
<td>GAS 2.21 upgrade guide</td>
<td>337</td>
</tr>
<tr>
<td>GAS 2.20 upgrade guide</td>
<td>337</td>
</tr>
<tr>
<td>Migrating from GAS 2.10.x or GWC 2.10.x</td>
<td>339</td>
</tr>
<tr>
<td>GAS (GWC) 2.10 upgrade guide</td>
<td>341</td>
</tr>
<tr>
<td>GAS 2.00 upgrade guide</td>
<td>342</td>
</tr>
<tr>
<td>Migrating Templates and Snippets Customizations</td>
<td>343</td>
</tr>
</tbody>
</table>

**Reference** | 343
---
Tools and Commands | 343
Dispatcher: httpdispatch | 344
Dispatcher: fastcgidispatch | 345
Dispatcher: isapidispatch | 346
Proxy: uaproxy | 347
Proxy: gwsproxy | 347
gasadmin tool | 348
Predefined resources | 358
Predefined resources overview | 358
Common predefined resources | 359
GAS Configuration Reference | 359
GAS configuration file (as.xcf) | 360
Configuration for Web and service applications | 360
Configuration file hierarchies | 361
Syntax diagrams for configuration files | 366
Configuration file elements | 367

**Legal Notices** | 441

**Copyrights and Trademarks** | 441

**Index** | 443
# Genero Application Server User Guide

Manual organization at a glance.

<table>
<thead>
<tr>
<th>Genero Application Server overview on page 11</th>
<th>GAS Quick Start on page 16</th>
<th>GAS Basics on page 30</th>
<th>GAS configuration on page 61</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What’s New</td>
<td>• Explore Genero application server resources on page 16</td>
<td>• Architecture of the Genero Application Server on page 30</td>
<td>• Validating the installation with the Genero Browser Client on page 61</td>
</tr>
<tr>
<td>• What is the Genero Application Server? on page 11</td>
<td>• Quick start guide for applications with UI on page 17</td>
<td>• Components of the Genero Application Server on page 38</td>
<td>• Validating the Installation with the GDC on page 62</td>
</tr>
<tr>
<td>• Standalone Genero Application Server on page 15</td>
<td>• Quick start guide for Web services applications on page 24</td>
<td>• What is auto logout? on page 53</td>
<td>• Licensing examples on page 65</td>
</tr>
<tr>
<td>• Front-ends and Extensions on page 16</td>
<td>• Genero demo applications on page 29</td>
<td>• What is delegation? on page 55</td>
<td>• ISAPI Extension Installation and Web Server Configuration on page 71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• What is Single sign-on (SSO)? on page 55</td>
<td>• FastCGI Installation and Web Server Configuration on page 98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GAS installation and application data directories on page 40</td>
<td>• Validating configuration files on page 108</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application environment on page 44</td>
<td>• Configuring applications on GAS on page 111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application Web Address on page 48</td>
<td>• How to implement delegation on page 124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Administering the GAS on page 176</th>
<th>Working with Genero Web Services on page 267</th>
<th>Working with Genero Browser Client (GBC) on page 259</th>
<th>How to implement Single sign-on (SSO) on page 139</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Monitoring on page 176</td>
<td>• Web services URI information on page 267</td>
<td>• Deploy GBC with gasadmin on page 260</td>
<td>• OpenID Connect SSO on page 142</td>
</tr>
<tr>
<td>• Log files on page 185</td>
<td></td>
<td></td>
<td>• SAML SSO on page 152</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Validating configuration files on page 108</th>
<th>Configuring applications on GAS on page 111</th>
<th>How to implement delegation on page 124</th>
<th>Compression in Genero Application Server on page 169</th>
</tr>
</thead>
<tbody>
<tr>
<td>• How to implement delegation on page 124</td>
<td>• Compression in Genero Application Server on page 169</td>
<td>• Configure the GBC development environment on page 171</td>
<td>• Configure multiple dispatchers on page 171</td>
</tr>
</tbody>
</table>

| Monitoring on page 176 | Log files on page 185 | Web services URI information on page 267 | Deploy GBC with gasadmin on page 260 | OpenID Connect SSO on page 142 | SAML SSO on page 152 |
### GAS 3.20 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 3.20.

**Important:** This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: [GAS 3.20 upgrade guide](#) on page 323.

#### Table 1: Single Sign-On (SSO) and delegation

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genero Identity Provider (GIP) integration to the GAS, ready to use with minimal settings. This will bring authentication and authorization mechanism to your apps.</td>
<td>See <a href="#">Identity Provider (IdP)</a> on page 139.</td>
</tr>
</tbody>
</table>
Overview

The DELEGATE element provides a feature that allows a user to be logged out of the authentication server when a Web application is closed.

- For OpenID Connect two new parameters are added for this configuration, IDP_LOGOUT_URL and SIGN_OFF.
- For SAML authentication the log-out behavior is specified in the SIGN_OFF parameter.

The DELEGATE element provides support for OAuth2 SSO authentication as used by identity providers (IdP) such as Facebook and Instagram.

- There are enhancements to the ImportOAuth tool command. It supports parsing of its command-line arguments with getopt.

The OpenID Connect service has three possible ways of authentication redirect: via an HTML submit form using GET or Post, or using the default HTTP 302.

The GetToken, DeployGar, and DeployGbc are command line tools for working with the GIP. They follow the FGL default for command options using getopt.

In an enhancement for GAS v3.20.14, the command tools are run from a script that can be set in the PATH environment.

The ImportIdP tool for managing SAML identity providers supports parsing of its command-line arguments with getopt.

Table 2: Web Services and the GAS

<table>
<thead>
<tr>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>With DELEGATE_OPTIONS you can specify whether to send the body or just HTTP headers for a service using delegation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>See DELEGATE_OPTIONS (for a service) on page 384</td>
</tr>
</tbody>
</table>

Table 3: Engine and Architecture

<table>
<thead>
<tr>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>The dispatcher configuration is enhanced to listen for incoming requests on a dedicated IP address. The address is specified by the LISTEN element.</td>
</tr>
<tr>
<td>With TCP_ADMIN_PORT you can specify a port for GAS administration tasks.</td>
</tr>
<tr>
<td>With END_URL you can specify a URL that the user agent redirects to when your Web application ends.</td>
</tr>
<tr>
<td>The CACHE_CONTROL_MAX_AGE element allows you to specify the duration files sent by the GAS are held in front-end cache.</td>
</tr>
<tr>
<td>The ENVIRONMENT_VARIABLE element has a Concat attribute, which allows you to manage how inherited parent configuration settings are handled; appended, prepended, or discarded.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>See LISTEN on page 402</td>
</tr>
<tr>
<td>See TCP_ADMIN_PORT on page 429</td>
</tr>
<tr>
<td>See END_URL on page 388</td>
</tr>
<tr>
<td>See CACHE_CONTROL_MAX_AGE on page 378</td>
</tr>
<tr>
<td>See ENVIRONMENT_VARIABLE on page 388</td>
</tr>
</tbody>
</table>
Genero Application Server overview

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

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.

A brief summary of some of GAS features are described:

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>In an enhancement for GAS v3.20.14, dispatchers create a pid file. You can set the <code>httpdispatch</code> and <code>fastcgidispatch</code> dispatchers to write their process-ids to a file on disk with the <code>--pid-file</code> option of the command.</td>
<td>See • <strong>Dispatcher: httpdispatch</strong> on page 344 • <strong>Dispatcher: fastcgidispatch</strong> on page 345</td>
</tr>
<tr>
<td>The following <code>gasadmin</code> commands are enhanced with list options:</td>
<td>See <strong>gasadmin tool</strong> on page 348.</td>
</tr>
<tr>
<td>• <code>gasadmin config --list</code> lists all applications and services (not just the deployed ones) found in the GAS.</td>
<td></td>
</tr>
<tr>
<td>• <code>gasadmin gbc --list</code> lists also all static and deployed GBC found.</td>
<td></td>
</tr>
<tr>
<td>The <code>gasadmin session</code> command is enhanced with the <code>--close-session</code> and <code>--close-all-sessions</code> options. This provides for closing sessions gracefully, without displaying messages to the user agent.</td>
<td></td>
</tr>
<tr>
<td>In an enhancement for GAS v3.20.14, the <code>$(res.dir.separator)</code> resource is added. It allows you to specify directory paths for all platforms in the same configuration. It resolves to either a forward slash &quot;/&quot; on a UNIX™-like system including macOS™ or a backward slash &quot;&quot; on a Windows® system.</td>
<td>No additional reference.</td>
</tr>
</tbody>
</table>

Table 4: Deployment

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The GAS allows you to provide your GBC client in the application path's <code>gbc</code> directory.</td>
<td>See <strong>Provide GBC in application path</strong> on page 262.</td>
</tr>
<tr>
<td>The GIP DeploymentApp provides you with an interface to manage Genero Archives, and Genero Browser Clients deployed on the Genero Application Server (GAS). It replaces the legacy deployment portal web service.</td>
<td>See <strong>Deploying and securing applications and Web services</strong> on page 236 and <strong>Set default GBC client with the Genero Deployment App</strong> on page 265.</td>
</tr>
<tr>
<td>• There are enhancements to the GBC deployment page to display a list of static GBC found via the <code>as.xcf GBC_LOOKUP_PATH</code>.</td>
<td></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.
<table>
<thead>
<tr>
<th>Manages Communication between front-end and DVM</th>
<th>The GAS creates relationships between various front-ends and the Dynamic Virtual Machines (DVMs) which run the applications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embeds a Web Server</td>
<td>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.</td>
</tr>
<tr>
<td>Simplifies Application Deployment</td>
<td>The GAS simplifies the deployment phase by taking care of the connection to the applications. For web-type applications no software installation or configuration is needed on the client; only a browser is required to access the application.</td>
</tr>
<tr>
<td>Controls Interaction between DVM and front-ends</td>
<td>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 Browser Client (GBC) 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 GBC application delivery on page 12 and GAS role in GDC application delivery on page 13.</td>
</tr>
<tr>
<td>Provides Genero Web Services (GWS) for clients</td>
<td>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 14.</td>
</tr>
</tbody>
</table>

**GAS role in GBC application delivery**

The GBC 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.
Figure 1: GBC Application Delivery

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 macOS™ 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 24.

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).
An HTTP request to perform a function of the Web service is sent to the dispatcher from the client.

1. The dispatcher checks the application configuration in the configuration files (xcf, xrd) and routes the request to the GWS proxy.

2. The GWSProxy is in charge of the pool of DVMs that will serve the Web service application, and perform the requested functions.

3. The DVM returns a HTTP response with requested data, and response codes to indicate success or failure.

4. The client interprets the HTTP response instructions and processes the returned data, for example, for display.

**Related concepts**

Architecture of the Genero Application Server on page 30

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

---

**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**

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 344) 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.

**Related concepts**

Development architecture (standalone GAS) on page 32

Use the standalone GAS to simplify your application development and testing without the need of the Web server. Understanding how to use this feature will assist you in the task of preparing applications for production environments.

Web Server configuration on page 70
For the deployment of GAS on a production environment, you need to configure a Web server.

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 Browser Client (GBC)**
The GBC 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 Browser Client is provided as part of the FGLGWS installation. For more information see the *Genero Browser Client User Guide*.

For the latest list of the browsers supported by the Genero Browser Client, refer to the *Supported platforms and databases* document (available on the Products download page of the Four Js Web site) or contact your support center.

**Genero Web Services (GWS)**
Genero Web Services client 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.

**Any Web Service Client**
The Genero Application Server is able to serve Web Service or RESTful clients written in languages other than Genero.

Related concepts
- [What is Web Service?](#) on page 24
  Web service is an interface where data is exchanged between applications instead of users.

GAS Quick Start

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

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

See the *GAS configuration file (as.xcf)* on page 360.

**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 5: FGLASDIR directories and files on page 41.

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 (for example, WNT or UNX).

   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 344, 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>.

**Related concepts**

- GAS configuration file (as.xcf) on page 360
- GAS installation and application data directories on page 40

The Genero Application Server is configured through a configuration file. The default configuration file is `as.xcf`, located in the $FGLASDIR/etc directory.

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.

---

**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**

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 Browser Web Client (GBC) and Genero Desktop Client (GDC).

Before you begin, you must:

- Have the Genero product suite installed locally.
• It is recommended to 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 by typing the command

   ```
   httpdispatch
   ```

   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 element.

   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 the GBC.
   To view the demo application displayed using the Genero Browser Client, click on the demos link.
   Alternatively, you can enter the address http://localhost:6394/ua/r/gwc-demo.
   The application displays in the browser.
To run the demo application using the Genero Desktop Client for HTTP, complete the following steps:

Typically applications are run from 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 `http://localhost:6394/da/1/gwc-demo` in a new browser tab.

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.

**Figure 4: Demo application launched by the GBC**
The GDC monitor launches and the demos application is opened in a system window.

**Figure 6: Demo application launched by the GDC**

**Related concepts**

Genero demo applications on page 29
A variety of demo applications are provided to demonstrate Genero functionality.

URI Examples on page 48
Several URI examples with ways to help you launch applications.

Front-ends and Extensions on page 16
The Genero Application Server can serve applications using various front-ends and extensions.

Related tasks
Configure an application on page 21
Before you run an application you need to configure it so that it can be executed by the Genero Application Server.

Run an application on page 22
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.

Deploy an application on page 23
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.

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 Application configuration file on page 113 (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 (for example, 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 (asxcf) on page 360).

```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/3.20/cfextwa.xsd">
  <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), for example, HelloWorld.xcf
5. To test, save the configuration file in a GAS application group directory.
    
    You can save the xcf in the default application group, $(res.appdata.path)/app, directory, or in your own defined group. Alternatively, you can deploy the app.

    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 22.

**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 21

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).

Refer to the Install and License your Genero Products manual 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 Browser Client (GBC), GDC, and from Genero Studio.

1. **Run your application on the GBC front-end.**
   a) If your standalone dispatcher is not running, start it from the command line by typing the command:

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

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

   You have launched the application in the GAS.

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

   Typically applications are run from 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 by pressing CTRL+c.

   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>` Web configuration option to have Studio launch the application for GBC.

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 23.

**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 21.

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 (for example, 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 Deploying, enabling, and running applications on GAS on page 296.

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

5. In the same directory, create a MANIFEST file (see MANIFEST file on page 280) and save it with the name "MANIFEST" (without extension).

   The MANIFEST contents resembles the following:

   ```xml
   <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 (gar) file is created in your current directory that has the same name as the directory. See the `fglgar` topic in the Genero Business Development Language User Guide.

**Related concepts**

Deploying and managing applications with GAR on page 285
Deploying and managing applications and Web services using Genero Archives on the GAS.

**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 Genero Archive (gar) file locally on your machine.
   
   To deploy an archive named HelloWorld_deploy.gar:
   
   ```bash
   gasadmin gar --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 the Genero Archive (gar) file are placed in the directory.

2. Enable your deployed application locally on your GAS.
   
   To list all deployed archives:
   
   ```bash
   gasadmin gar --list-archives
   ```
   
   To enable the archive, reference it by its archive name:
   
   ```bash
   gasadmin gar --enable-archive HelloWorld_deploy
   ```
   
   This enables the application by copying its configuration file (for example 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?**

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 29. For more information on Web service, refer to the Genero Business Development Language User Guide.

**Related tasks**

Explore Genero demo Web services (server side) on page 25

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

Configure a Web service on page 26
Create a configuration file with the details the Genero Application Server needs to deliver the Web service to client applications.

**Run a client app on Web service using GBC** on page 26
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.

**Deploy a Web service** on page 27
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.

### Explore Genero demo Web services (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 344.

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 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"
  ...
  <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.

### Related tasks
- **Configure a Web service** on page 26
  Create a configuration file with the details the Genero Application Server needs to deliver the Web service to client applications.

- **Run a client app on Web service using GBC** on page 26
  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.

- **Deploy a Web service** on page 27
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.

**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 (for example, 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 (as.xcf) on page 360).

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="ws.default" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextws.xsd">
    <EXECUTION>
        <PATH><path_to_your_local_directory></PATH>
        <MODULE>CalculatorServer.42r</MODULE>
    </EXECUTION>
</APPLICATION>
```

4. To test, save the Web service configuration file in a GAS services group directory.

You can save the `.xcf`, for example, `mycalculator.xcf` in the default service group, `$res.appdata.path/services`, directory, or in your own defined group. Alternatively, you can deploy the service.

You have successfully configured an application.

**Related concepts**

*Configure applications for Web service* on page 116

Create an application configuration file (.xcf) for a Web services application.

**Run a client app on Web service using GBC**

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 *Install and License your Genero Products* manual 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 26 and use the calculator client application available from the GAS demos page, see *Access demo applications with the Genero Browser Client* on page 29, to interact with it.
1. Start the standalone dispatcher from the command line by typing the command:

   httpdispatch

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.

4. 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 option (Customize) Click here to add your own URL and provide the URL to your configured web service, for example, 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 26.

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

   <PATH>$\{res.deployment.path\}/server</PATH>

2. Create a new directory where you will archive the Web service source files (for example 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 MANIFEST file on page 280.

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

5. Create a Genero 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: fg1gar --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 a Genero Archive (gar) file named calculator_deploy.gar:

   gasadmin gar --deploy-archive calculator_deploy.gar
A subdirectory is created in your $(res.deployment.path) directory identified by the configuration file name and the date and time deployed, e.g. calculator_deploy-20150423-130838. All the files contained in the Genero Archive (gar) file 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 gar --list-archives
   ```

   To enable the archive, reference it by its archive name:
   
   ```
   gasadmin gar --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 344.

   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/dem0/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.

**Related concepts**

Deploying and managing applications with GAR on page 285
Deploying and managing applications and Web services using Genero Archives on the GAS.

**Genero demo applications**

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

**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).

**Access demo applications with the Genero Browser Client**

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

**The demos configuration in the GAS**

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

```xml
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo" Parent="defaultwa">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo.42m</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>${res.access.control}</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 element.

**Accessing the demos from GBC user interface**

To access the GAS `gwc-demo` application, you can use the Genero Browser Client user interface. See the Genero Browser Client User Guide. 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.
Accessing the demos directly with URL

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

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

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).

Figure 7: GAS with VMProxy Architecture
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 Genero web client 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).

Note: The GAS supports the redirection of the start of an application or service to a delegation service to perform some controls to authenticate a user before granting access and starting an application. For more information see How to implement delegation on page 124.

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 fastcgidispatch.exe is for use with FASTCGI-compliant Web servers such as Apache Tomcat®, while the isapidispatch.dll is for the Microsoft® 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 and track 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 on the VMProxy responsibilities, see VMProxy responsibilities.

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

Related concepts

What is the Genero Application Server? on page 11
The Genero Application Server (GAS) is an engine that delivers Genero applications for various Genero front-ends in both development and production environments.

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.
- One VMProxy manages a session. This can mean one or more applications are running in a session (depending on whether, for example, applications use RUN, or RUN WITHOUT WAITING to start applications).

If you need to stop a session by stopping the VMProxy, you can do so without affecting any other applications that are running concurrently in other sessions. For instance, 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.

Related concepts

Monitoring on page 176
Use the /monitor URL to view information on the current status of GAS dispatcher and on active applications.

**Performance tuning** on page 190
These topics cover various performance tuning considerations for configuring your Web server, your Genero Application Server, and your Genero applications.

**Load balancing** on page 191
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.

**Development architecture (standalone GAS)**
Use the standalone GAS to simplify your application development and testing without the need of the Web server. Understanding how to use this feature will assist you in the task of preparing applications for production environments.

The *httpdispatch* process allows you to connect directly to a GAS dispatcher without involving a Web server. This connection is the typical method used during development.

**Note:** From version 3.10 a Genero Application Server for Java (JGAS) is provided in the FGLGWS package. It can also be used for development environment testing as standalone. For more information, see the *Genero Application Server for Java User Guide*.

---

**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.

**Related concepts**
*Dispatcher: httpdispatch* on page 344

*httpdispatch* is the standalone dispatcher that starts the Genero Application Server (GAS) in command line. No web server is needed.

**Setting up production environment**

There are different options for configuring a GAS installation depending on your network. A Web server is required and security needs to be considered.

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.
**Installation example 1: all-in-one machine**

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

**Installation example 2: database on separate server**

You can have the GAS installed on the same machine as the Web server and the database on a separate server as shown.

**Installation example 3: behind frontal web server**

If you already have a Web server (server A) on the internet (in 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 GAS.

**Note:** If you are using the isapidispatch dispatcher, it is bound to the Web server and therefore needs to be on the same host. Installations using fastcgidispatch can be on the same host as the Web server but may also be installed on a separate host, see Installation example 4: GAS on a separate server to the Web server (fastcgidispatch only) on page 34.

This second Web server needs to be protected and therefore sits in the protected network. The frontal Web server forwards application requests to the internal Web server on server 1 as shown.
Installation example 4: GAS on a separate server to the Web server (fastcgidispatch only)

If your installation uses the fastcgidispatch dispatcher, you can install the GAS on a separate server to the Web server.

The Web server forwards application requests to the GAS on server 1 as shown.

It is recommended that the configuration for production environments be supervised by a Web specialist to avoid security issues.

**Related concepts**

Web Server configuration on page 70

For the deployment of GAS on a production environment, you need to configure a Web server.

**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 411 is disregarded. For instance, the
MAX_AVAILABLE 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 gwsproxy calculates this based on a combination of the following factors:

- **Time to Start a New DVM**
  The 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 MIN_AVAILABLE because the
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
  gwsproxy waits a full one second before shutting down the DVM.

- **Maximum (ten minutes)**
  If time to shutdown is more than ten minutes, the
  gwsproxy waits no longer than ten minutes before shutting down a DVM.

**Example 1: One GWSPProxy 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 START element defines how many DVMs to start for a particular
Web service. There is one GWSPProxy in charge of the pool of DVMs for the Web service. For our example, this
means that one GWSPProxy 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 35.

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 `MINAVAILABLE` 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, `MAXAVAILABLE` 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 \texttt{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 request 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.
Proxies deliver applications on the GAS. Being aware of the different types of proxies there are, assists you in developing and deploying applications accordingly.

The Genero Application Server is configured through a configuration file. The default configuration file is `as.xcf`, located in the `$FGLASDIR/etc` directory.

**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.

**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` is the standalone dispatcher for development only. It is provided to simplify your development setup and configuration through a direct connection without a Web server.
- `isapidispatch`. This is the dispatcher for Internet Information Services (IIS).
- `fastcgidispatch`. This is dispatcher for Fast CGI compliant Web servers like Apache.

Each dispatcher performs the 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

**Related concepts**

- **Dispatcher:** [httpdispatch](http://example.com) on page 344
  
  httpdispatch is the standalone dispatcher that starts the Genero Application Server (GAS) in command line. No web server is needed.

- **Dispatcher:** [isapidispatch](http://example.com) on page 346
  
  isapidispatch is the dispatcher dedicated to Internet Information Services (IIS) for Web servers on Windows® platforms.

- **Dispatcher:** [fastcgidispatch](http://example.com) on page 345
  
  fastcgidispatch is the dispatcher for Apache Web servers supporting FastCgi protocol.

- **Configure multiple dispatchers** on page 171
  
  If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

**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 Browser Client (GBC) interfaces, Genero Mobile for Android™ (GMA), Genero Mobile for iOS (GMI) see **Proxy: uaproxy** on page 347
- **gwsproxy:** proxy for Genero Web Service (GWS) type applications, see **Proxy: gwsproxy** on page 347

**VMProxy responsibilities**

Each proxy performs the 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
- 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 437)
- Manages sessions when the client is GDC, GBC, GMA or GMI

**gwsproxy**

The *gwsproxy* is responsible for the following:

- Handles the GWS DVM pool (see Services Pool (GWS Only) on page 34)
- Handles HTTP Web Services requests
- Forwards HTTP Web Services responses (SOAP, REST, XML over HTTP)

**Related concepts**

[Dispatcher](#) on page 38
Understand what the dispatcher does and identify which dispatcher to use with a specific Web server.

[DVM](#) on page 40
The Dynamic Virtual Machine (DVM) or runtime system is the process (*fglrun* process) 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.

**GAS installation and application data 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:

- The installation directory. See [GAS Installation directory](#) on page 40.
- The application data directory. See [GAS application data directory](#) on page 42.

**Important:** The application data directory needs to be on the same file system. For more information read the section on [Prerequisites for application data directory](#) on page 44.

**GAS Installation directory**

The installation directory contains the installation files. It is defined by the FGLASDIR environment variable and set by the resource `$(res.path.as)` in the GAS configuration file. The directories and files of the GAS installation directory are described in [Table 5: FGLASDIR directories and files](#) on page 41.
### Table 5: FGLASDIR directories and files

<table>
<thead>
<tr>
<th>Directory</th>
<th>GAS installation directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FGLASDIR</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>Contains script for setting environment variables: On Linux®/UNIX™: envas On Windows®: envas.bat</td>
</tr>
<tr>
<td>/bin</td>
<td>*.exe</td>
</tr>
<tr>
<td>/Docker</td>
<td>*.exe, Dockerfile</td>
</tr>
<tr>
<td>/etc</td>
<td>as.xcf</td>
</tr>
<tr>
<td>/tpl</td>
<td></td>
</tr>
<tr>
<td>/shortcut</td>
<td>*.gdc</td>
</tr>
<tr>
<td>/web</td>
<td>demos.html, etc.</td>
</tr>
</tbody>
</table>
GAS application data directory

The application data directory, or appdata, is where the application data files managed by the GAS are located. The appdata directory is set by the resource $(res.appdata.path) in the GAS configuration file. The directories and files of the appdata directory are described in Table 6: appdata directories and Files on page 43.
### Table 6: appdata directories and Files

<table>
<thead>
<tr>
<th>Directory</th>
<th>GAS application data directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appdata</td>
<td>On Windows®: C:\ProgramData\FourJs\gas\gas_version</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On Linux/UNIX: $FGLASDIR/appdata</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-directories</th>
<th>Files or Details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/app</td>
<td>*.xcf</td>
<td>Default application group; the default location for your application configuration files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sub-directories</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Files</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>/deployment</td>
<td></td>
<td>Contains a directory for each application.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applications deployed with Genero Archive.</td>
</tr>
<tr>
<td>/gbc_deployment</td>
<td></td>
<td>Contains a directory for each GBC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Genero Browser Clients deployed on the GAS.</td>
</tr>
<tr>
<td>/public</td>
<td></td>
<td>Note: 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 its sub-directory paths.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-directories</th>
<th>Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/common</td>
<td>Image files and resources</td>
<td>Stores common images used by applications. It is the default PUBLIC_IMAGEPATH directory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A directory is created for each application.</td>
</tr>
<tr>
<td>/deployment</td>
<td>Image files and resources</td>
<td>A directory is created for each application.</td>
</tr>
</tbody>
</table>

| /log              | vm, proxy, and dispatcher logs.   | The logs for a given day are stored in a directory named for that date.     |
|                   | Important: Users need write permissions for this directory. Users who installed the GAS have write permissions. If users that start the GAS (for example Apache®) are not in that group, then you need to grant them write permission. A directory is created for each dispatcher, for example, httpdispatch (for the standalone GAS). |
|                   | Note: DVM logs are redirected to files when DAILYFILE is set for the log output type. |

| /session          | Persistent session table information. | A directory is created for each dispatcher. |
|                   | Important: Users need write permissions for this directory. Users who installed the GAS have write permissions. If |
Prerequisites for application data directory

GAS requires that all appdata files are located on the same file system. This requirement is based on the following:

- 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 information, 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 customize the application 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.

Note: 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 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):

  ```xml
  <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.

Related concepts

GAS configuration file (as.xcf) on page 360
The Genero Application Server is configured through a configuration file. The default configuration file is as.xcf, located in the $FGLASDIR/etc directory.

Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

Resource deployment on page 276
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.

URI Examples on page 48
Several URI examples with ways to help you launch applications.

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 (GDC) and Genero Browser Client Genero (GBC), or Web service. See:
   - How to use environment variables in GDC and GBC on page 45
   - How to retrieve web server information in a web service on page 47

**How to get the URL of the application**

The FGL_VMPROXY_START_URL environment variable stores the real URL of an application. The real URL refers to the URL that the user clicked to start the application. Use the Genero fgl_getenv() function in your application to retrieve the value of the environment variable.

```plaintext
LET startUrl = fgl_getenv( "FGL_VMPROXY_START_URL" )
```

The variable name must be in uppercase.

**Note:** There are environment variables that are internal to the GAS, to include variables with the FGL_VMPROXY_ prefix. As the name suggests, these environment variables are set by the dispatcher starting the proxy. In general, you do not have access to these environment variables, however an exception was made for the FGL_VMPROXY_START_URL environment variable.

**How to use environment variables in GDC and GBC**

The GAS gets environment at the start of Web applications from these sources:
   - Standard HTTP request header fields
   - Web server Common Gateway Interface (CGI) environment variables

It handles these in different ways by converting them to environment variables you can use at runtime.

**HTTP request headers converted to FGL_WEBSERVER_HTTP_***

The uaproxy converts HTTP headers, such as the User-Agent, to environment variables by adding the FGL_WEBSERVER_HTTP_ prefix.

**Note:** The GAS replaces the dash (or minus "-") characters with underscore (_) characters. For example, it converts the header "User-Agent" to the FGL_WEBSERVER_HTTP_USER_AGENT environment variable.

If you need to use these variables in your application, use the Genero fgl_getenv() function to retrieve their value. For example to identify the client browser, you add a statement like this:

```plaintext
LET user_agent = fgl_getenv( "FGL_WEBSERVER_HTTP_USER_AGENT" )
```

**Web server CGI environment converted to FGL_WEBSERVER_***

The uaproxy converts some CGI variables (see Table 7: Web server environment variables on page 46) transmitted by the Web server (Apache, IIS, etc.) by adding them to the GAS environment with FGL_WEBSERVER_ as prefix. For more information on the standard specification, see rfc3875, or see your web server's documentation.
Table 7: Web server environment variables

<table>
<thead>
<tr>
<th>Web server variables</th>
<th>GAS converted variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE_ADDR</td>
<td>FGL_WEBSERVER_REMOTE_ADDR</td>
<td>Returns the network address of the client sending the request to the server.</td>
</tr>
<tr>
<td>REMOTE_USER</td>
<td>FGL_WEBSERVER_REMOTE_USER</td>
<td>Returns the user name supplied by the client as part of user authentication.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>FGL_WEBSERVER_SERVER_NAME</td>
<td>Returns the name of the server host to which the client request is directed.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>FGL_WEBSERVER_HTTPS</td>
<td>Returns &quot;ON&quot; if the request came in through a secure channel (for example, SSL); or it returns &quot;OFF&quot;, if the request is for an insecure channel.</td>
</tr>
</tbody>
</table>

Use the Genero fgl_getenv() function in your application to retrieve their values. For example, to find out if the Web server supports the HTTPS protocol, add this statement in your application:

```plaintext
LET httpsOn = fgl_getenv( "FGL_WEBSERVER_HTTPS" )
```

With Genero Browser Client applications, environment variables are also passed to the client bootstrap mechanism (bootstrap.html) that loads the user interface. This implementation, however, is reserved for internal use and is subject to change without notice. It must not be modified in any way. For more information on the bootstrap mechanism, see the Genero Browser Client User Guide.
How to retrieve web server information in a web service

With Genero Web services applications, environment variables cannot be used to pass HTTP request headers because the gwsproxy has already started. The only way to send the environment is through the HTTP header. Header names set by the GAS have the following form:

```
X-FourJs-Environment-Variable-MyHeaderName
```

Where `MyHeaderName` is the header name.

Some variables transmitted by the Web server (Apache, IIS, etc.) are also available to the environment:

- `X-FourJs-Environment-Variable-REMOTE_ADDR`
- `X-FourJs-Environment-Variable-REMOTE_USER`
- `X-FourJs-Environment-Variable-SERVER_NAME`
- `X-FourJs-Environment-Variable-HTTPS`

There are options available to retrieve headers depending on whether you are working with GWS RESTful high-level APIs or using low-level APIs:

**RESTful high-level APIs**

- Set a `WSHeader` attribute as part of a function parameter. For example,
  
  ```plaintext
  ip_addr STRING ATTRIBUTE(WSHeader, WSOptional, WSName="X-FourJs-Environment-Variable-REMOTE_ADDR")
  ```

  Or:

  - Set a context dictionary variable (for example `context`) at the modular level with the `WSContext` attribute. This allows you to retrieve all `X-FourJs-Environment-Variable-xxx` set by the GAS by referencing a dictionary key value, for example, `DISPLAY context["Variable-REMOTE_ADDR"]`.

**Low-level APIs**

With low-level APIs you use the `com.HTTPServiceRequest.getRequestHeader()` method. For example:

```plaintext
LET param = req.getRequestHeader("X-FourJs-Environment-Variable-REMOTE_ADDR")
```

**Note:** For more information on RESTful high-level API and low-level API methods, see the Web services section in the Genero Business Development Language User Guide.

**Related concepts**

- **ENVIRONMENT_VARIABLE** on page 388
  The `ENVIRONMENT_VARIABLE` element provides the value to be set for an environment variable.

- **EXECUTION (for an application)** on page 390
  This `EXECUTION` element sets the runtime environment for an application by specifying parameters for executing it.
The Genero Application Server 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.

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

URI Examples

Several URI examples with ways to help you launch applications.

In this page examples are grouped under the headings: connections, launching applications, launching applications with arguments, and launching Web services.

- Connections
  - Direct connection to standalone GAS on page 48
  - Connection through Web server on page 49
- Launching applications
  - Running Web applications on page 49
  - Running desktop applications on page 49
- Launching applications with arguments
  - Applications with arguments set in the URL
  - Set GBC customization in URL parameter on page 50
- Web services
  - Running SOAP Web service applications on page 50
  - Running REST Web service applications on page 50

URI Syntax

A Web application URI entered in your browser's address bar takes the form of the following syntax:

http://app-server:port/scope/action/app-name

Where:

- *app-server* is the name or IP address of the Application Server
- *port* is the port number on which the Application Server listens
- *scope* is the client protocol, for example, *ua* (uaproxy) for applications, or *ws* (gwsproxy) for Web services
- *action* is the action requested of the Application Server. For example, *r* is used for a start-up request. This changes depending on the action requested of the server as the application runs.

Connections

Direct connection to standalone GAS

- This example launches the "myApp" Web application on the "localhost" Application Server, listening to port 6394:

  http://localhost:6394/ua/r/myApp
• If the startup URL lists additional path members, they are allowed but ignored:

```text
http://localhost:6394/ua/r/myApp/foo/bar
```

In this example, the "myApp" Web application is launched, and /foo/bar is ignored.

When the proxy is started, the FGL_VMPROXY_START_URL environment variable is set to the URL used to start the application. In our example, this includes the additional path members /foo/bar, making the full path available for use by the application.

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

```text
http://localhost:6394/ua/r/_default/myApp
```

The use of the _default group name is optional.

**Connection through Web server**

This example launches the "myApp" application through the "myWebServer" Web Server.

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

The `gas` is a directory or virtual directory on the Web Server defined by the `$connector.uri` resource in the GAS configuration file (`as.xcf`). Typically this directory is called "gas".

**Launching applications**

**Running Web applications**

This URL example launches the "myApp" application:

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

**Running desktop applications**

Using the URL with `da` protocol launches an application via the Genero Desktop Client (GDC) monitor:

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

**Note:** Prerequisites:

1. GDC is installed
2. Application extension associations for `gdc` are set

**Application with arguments set in URL**

If the `AllowUrlParameters` attribute in the EXECUTION (for an application) on page 390 is set to `TRUE`, parameters can be passed as arguments in the application URL using the format "Arg=value".

For example, the "myApp" application is launched with two arguments by the "myWebServer" Web server:

```text
```

• A question mark (?) follows the application name.
• In the key/value pair format, `Arg` is the key and `Val1` is the value of the argument.
• Each key/value pair is separated by an ampersand (&).

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.

For examples using arguments in the app URL, see Set GBC customization in URL parameter on page 50.
Set GBC customization in URL parameter

To specify the customization to use for the GBC user interface, add the gbc query string parameter to the application URL.

http://myApplicationServer:6394/ua/r/gwc-demo?gbc=<my-custom-gbc>

You can see:

- A question mark (?) follows the application name.
- gbc is the required query string parameter.

**Note:** In the Genero web client prior to version 3.00, the parameter gwc-js was used. From version 3.10, it is renamed gbc. Use of gwc-js is deprecated but remains for compatibility.

- The <my-custom-gbc> value is either the name of a text file referencing the customization directory name, or the actual name of the customization directory in paths defined by the GBC_LOOKUP_PATH on page 395 element.

**Important:** The gbc value provided in the query string takes precedence over the configuration for the GBC on page 394 element in the application configuration file (xcf).

Web services

Web services URI Syntax

A Web services URI entered in your browser's address bar takes the form of the following syntax:

http://app-server:port/scope/action/app-name

Where:

- **app-server** is the name or IP address of the Application Server
- **port** is the port number on which the Application Server listens
- **scope** is the client protocol ws (gwsproxy) for Web services
- **action** is the action requested of the Application Server. For example, r is used for a start-up request. This changes depending on the action requested of the server as the application runs.

Running REST Web service applications

- To get the OpenAPI specification file for a specified service:
  
  http://appserver:6394/ws/r/xcf/resource?openapi.json

- To access a Web service resource:
  

- If the Web service uses a group:
  
  http://appserver:6394/ws/r/group/xcf/resource/resource-endpoint

- Access through a Web server:
  
  http://webserver/gas/ws/r/xcf/resource/resource-endpoint

Running SOAP Web service applications
• 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
```

Related concepts

GAS installation and application data directories on page 40
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.

Application environment on page 44
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.

Related reference

File serving URIs on page 51
You can determine the file system location of resources specific to your deployed applications on the GAS from the application URL of a request. Examples are given for document root, public images, and web components.

File serving URIs
You can determine the file system location of resources specific to your deployed applications on the GAS from the application URL of a request. Examples are given for document root, public images, and web components.

If the application URL is given as `http://localhost:6394/demos.html`, you can locate various resource as described in Table 8: File serving URIs on page 52.
### 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 /CONFIGURATION/APPLICATION_SERVER/INTERFACE_TO_CONNECTOR/DOCUMENT_ROOT, see DOCUMENT_ROOT on page 386. For example, this excerpt is from the default GAS configuration file (as.xcf).</td>
</tr>
<tr>
<td>/ua/i/</td>
<td>In this URL the slash (/) following the ua/i/ begins the path to the public resource directory where applications' public images are found. See Paths to application resources on page 277. The public resource directory is set by $(res.public.resources) in the as.xcf file. It is typically set to common, which expands to the $(res.appdata.path)/public/common directory in your application data path. The PUBLIC_IMAGEPATH on page 414 that is used by all applications is set to $(res.public.resources) by default. For example, an image put in this path can be found at:</td>
</tr>
</tbody>
</table>

   http://localhost:6394/ua/i/common/my_photo.jpeg

   If you deploy an application with public images (see Resource deployment on page 276), they are put in its deployment path, $(res.appdata.path)/public/deployment/my_gar_file. The $(res.public.resources) is updated to include this path (as well as common) in a search for images. Images may then be found at a URI of the form: |

   http://localhost:6394/ua/i/public/deployment/my_gar_file/my_deployed_photo.jpeg

   The images in common are also accessible. |
### URI path Description

<table>
<thead>
<tr>
<th>URI path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ua/components</td>
<td>In this URL the slash (/) following the ua/components/ is the path to the web</td>
</tr>
<tr>
<td></td>
<td>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 440. For example, this excerpt is from the default GAS configuration file (as.xcf).</td>
</tr>
</tbody>
</table>

```
...<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
  ...
  <WEB_COMPONENT_DIRECTORY>${application.path}/webcomponents</WEB_COMPONENT_DIRECTORY>
  ...
</WEB_APPLICATION_EXECUTION_COMPONENT>
...```

In this example, when accessing a web component with the URL http://localhost:6394/ua/components/mywebcomponent, you expect to find the file in the directory specified by the WEB_COMPONENT_DIRECTORY entry, which is the /webcomponents directory in your application path.

---

**Related concepts**

PUBLIC_IMAGEPATH on page 414
The PUBLIC_IMAGEPATH element defines the public resources directory used by applications.

---

**What is auto logout?**

The GAS supports an automatic timeout feature which can be configured to display a log out page after a specified time of user inactivity on an application is detected.

When the AUTO_LOGOUT is set, the DVM detects when an application has no user activity. It then waits the number of seconds specified by the TIMEOUT element before triggering an auto logout event and sending the application a log out message.

- For Genero Browser Client (GBC) applications, you get a log out page.
- For Genero Desktop Client (GDC) applications, you 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.**

**Related concepts**

AUTO_LOGOUT on page 375
The AUTO_LOGOUT element defines the auto-logout mechanism to be used for an application.

PROMPT (for auto logout) on page 412
The PROMPT element provides a feature that allows a user to resume an application after an auto logout event.

Re-login prompt on page 54
The auto-logout prompt feature allows a Genero application to prompt a user to re-log in to the application after an auto-logout event. This feature is configured in the `AUTO-LOGOUT` element and does not require a change in your existing Genero BDL code.

### Web application timeouts

**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:

- POST request to send data to the DVM
- GET request to retrieve data

The Genero Application Server, on the other hand, cannot send a request to the front-end client because the front-end does not have a public address. As a result, requests are always initiated by the front-end and the server response is done with the same connection. Between requests, the GAS stores data sent from the DVM in its buffer and keeps it for the next GET request from the front-end.

**Related concepts**

- [WEB_APPLICATION_TIMEOUT_COMPONENT](#) on page 439
  
  The `WEB_APPLICATION_TIMEOUT_COMPONENT` element defines a set of timeout values to be used when configuring a Web application.

- [SERVICE_APPLICATION_TIMEOUT_COMPONENT](#) on page 422
  
  The `SERVICE_APPLICATION_TIMEOUT_COMPONENT` element creates a set of timeout values to be used when configuring a Web service.

- [TIMEOUT (for an application)](#) on page 431
  
  This element sets timeouts for process requests and the start of the DVM.

- [TIMEOUT (for a service)](#) on page 432
  
  This element sets timeouts for process requests and the start of the DVM.

### Re-login prompt

The auto-logout prompt feature allows a Genero application to prompt a user to re-log in to the application after an auto-logout event. This feature is configured in the `AUTO-LOGOUT` element and does not require a change in your existing Genero BDL code.

When an `AUTO_LOGOUT` event is triggered, the user-agent displays a screen or page to notify the user that a re-login is required if the user wants to continue.

If the user clicks on the re-login button, the user agent is redirected to a URL specified in the `PROMPT` configuration element where the user's credentials are checked.

Once the user is authenticated by the service, the user-agent is redirected back to the GAS to resume the application.

If nothing is done during the `Timeout` defined by the `PROMPT`, the application ends.

**Note:** The re-login prompt feature is supported on the GBC (starting from version GBC 1.00.38). It is not supported on Genero Mobile front-ends (GMA, GMI), and the Genero Desktop Client (GDC). If used, an HTTP 410 error page is displayed.

**Related concepts**

- [PROMPT (for auto logout)](#) on page 412
  
  The `PROMPT` element provides a feature that allows a user to resume an application after an auto logout event.

- [How autologout prompt is implemented on SSO](#) on page 165
The prompt feature can authenticate the user and resume the application after an auto-logout event.

**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 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 55.

**Related concepts**

**How delegation works** on page 125
The delegation process redirects the start of an application or service to a delegation service to authenticate a user. To help you understand delegation, different scenarios for starting and/or refusing a service and the communication paths involved are illustrated.

**Delegation use cases** on page 136
Three examples of possible uses for the delegation mechanism.

**Configure delegation for application or service** on page 127
To delegate the start of an application or service to the Genero REST service, specify a `DELEGATE` element in the `EXECUTION` component of your application or service.

**What is Single sign-on (SSO)?**

Single sign-on allows you 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.

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

**Genero Identity Provider**

Genero provides its own Identity Provider for securing applications and RESTful Web services.

See Genero Identity Provider (GIP) on page 204.

**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 142.

**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 152

**Related concepts**

How to implement Single sign-on (SSO) on page 139
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.

**Internationalization and localization**

How the Genero Application Server handles international applications.

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

**Best practices**

Implement these best practices when applications use one or more character sets to support different languages.

If you support users in multiple countries, you’ll need to consider internationalization and localization when running Genero applications.

**Important:** Read the recommendations for application localization set out in the page on *Understanding locale settings* in the *Genero Business Development Language User Guide*.

The following checklist is intended to help you ensure that your Genero application locale, which defines the language and character set, is properly set for when the GAS interacts with Genero applications using different encodings:

1. It is recommended to run the GAS in an UTF-8 environment to avoid file conversion overhead.

   **Note:** Since version 3.20.13, the GAS manages file transfer URLs based on a different charset provided by the DVM. This allows the DVM to start in any locale.

2. It is recommended to set the DVM locale in the `.xcf` file. See *DVM locale settings in .xcf files* on page 59 for details.

3. It is recommended to create the application configuration (`.xcf`) file in UTF-8 encoding.

**Related concepts**

- **GAS system encoding** on page 58
  - System character encoding matters when Genero Application Server (GAS) interacts with the operating system.
- **Encoding architecture** on page 56
  - International applications use one or more character sets to support different languages.

**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, 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
- Operating system locale

The DVM uses:

- Operating system locale

Related concepts

GAS configuration encoding files on page 58
Configuration for the Genero Application Server is provided in XML files (with .xcf file extension), which can include international characters and multiple languages.

GAS system encoding on page 58
System character encoding matters when Genero Application Server (GAS) interacts with the operating system.

Genero Browser Client charset

When a Genero Browser Client (GBC) type application starts, the bootstrap mechanism sets the charset needed to handle the application in the front-end.

The charset is defined in the metadata of the bootstrap.html. This file is found in the path to your custom GBC front-end, or in the default GBC client directory $FGLDIR/web_utilities/gbc if no customization is being used. This setting takes precedence over system locale settings.

Example

The code shown in the example comes from the bootstrap.html. The charset encoding the GBC uses by default is the UNICODE standard UTF-8 as it provides support for multiple languages.

Note: The bootstrap mechanism is an internal implementation. For more information, see the Application bootstrap page in Genero Browser Client User Guide.
This file can be modified by licensees according to the product manual.
-->

<html>
<head>
<meta charset="utf-8">
<title>Genero Browser Client</title>
<meta name="viewport" content="width=device-width, initial-scale=1">

...

</head>

GAS configuration encoding files

Configuration for the Genero Application Server is provided in XML files (with .xcf file extension), which can include international characters and multiple languages.

In XML files the UNICODE standard UTF-8 is typically used.

Note: The charset defined in XML files for the GAS 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"?>
<CONFIGURATION ...>
```

Example in as.xcf with ISO-8858-6 (Arabic) character set:

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

For more information on encoding in an XML file, see Extensible Markup Language - Character Encoding.

GAS system encoding

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

For example, GAS uses the character encoding set by the operating system when it performs the following:

- 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 environment:

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

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

  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>

• For the front-ends such as Genero Browser Client (GBC) that use UTF-8 encoding:
  
  • The charset in the metadata of the bootstrap.html file is used. For more information, see the Genero Browser Client User Guide.
  
  • For the DVM the conversion is done in the DVM locale, see DVM locale settings in .xcf files on page 59.

Related concepts

Genero Browser Client charset on page 57

When a Genero Browser Client (GBC) type application starts, the bootstrap mechanism sets the charset needed to handle the application in the front-end.

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 settings in .xcf files

The locale settings for the DVM must be defined in the application .xcf files.

Setting a DVM locale is described in the chapter on Application locale in the Genero Business Development Language User Guide.

On Linux®/UNIX™, the DVM locale is defined by the LC_ALL environment variable. On Windows® the DVM locale is defined by the LANG environment variable.

The DVM locale configuration must be defined in the application specific .xcf file, with the ENVIRONMENT_VARIABLE on page 388 element.

Note: If the application connects to an SQL database, make sure to properly define the database client locale. The next examples show how to set the Informix® client environment variables CLIENT_LOCALE and DB_LOCALE.

Example with UTF-8 locale settings in Linux:

Note: When using a UTF-8 locale, consider using char length semantics, with FGL_LENGTH_SEMANTICS=CHAR.

```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/3.20/cfextwa.xsd"
>
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="LC_ALL">en_US.utf8</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="FGL_LENGTH_SEMANTICS">CHAR</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="CLIENT_LOCALE">en_us.utf8</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="DB_LOCALE">en_us.utf8</ENVIRONMENT_VARIABLE>
  </EXECUTION>
</APPLICATION>
```
Example with UTF-8 locale settings in Windows:

Note: The Microsoft® C runtime setlocale() "_.utf8" code page name is supported starting from Windows® 10 build 17134 (April 2018 Update). For older Windows® versions, use the BDL UTF-8 fallback ".fglutf8".

```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/3.20/cfextwa.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="LANG">English_US.utf8</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="FGL_LENGTH_SEMANTICS">CHAR</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="CLIENT_LOCALE">en_us.utf8</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="DB_LOCALE">en_us.utf8</ENVIRONMENT_VARIABLE>
    <PATH>/app/stores/bin</PATH>
    <MODULE>order_input</MODULE>
  </EXECUTION>
</APPLICATION>
```

Example for a French locale with ISO-8859-1 charset in Linux:

```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/3.20/cfextwa.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="LC_ALL">fr_FR.ISO8859-1</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="CLIENT_LOCALE">fr_fr.819</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="DB_LOCALE">fr_fr.819</ENVIRONMENT_VARIABLE>
    <PATH>/app/stores/bin</PATH>
    <MODULE>order_input</MODULE>
  </EXECUTION>
</APPLICATION>
```

Example for a French locale with CP-1252 charset in Windows:

```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/3.20/cfextwa.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="LANG">french_FRANCE.1252</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="CLIENT_LOCALE">fr_fr.1252</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="DB_LOCALE">fr_fr.1252</ENVIRONMENT_VARIABLE>
    <PATH>/app/stores/bin</PATH>
    <MODULE>order_input</MODULE>
  </EXECUTION>
</APPLICATION>
```

Related concepts
Best practices on page 56
Implement these best practices when applications use one or more character sets to support different languages.

**Encoding architecture** on page 56
International applications use one or more character sets to support different languages.

---

**GAS configuration**

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

---

**Managing user access rights**

When the Genero Application Server starts an application, the user needs to have rights to execute the DVM and access any needed program files and resources.

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 a 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 Browser Client**

For the Genero Browser 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 GBC 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 Browser 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 367.

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 343.
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:
   ```
   http://localhost:6394/demos.html
   ```
4. Launch the GWC Demos program using a URI that provides a direct connection to the standalone GAS:
   ```
   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 on page 368 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 GBC 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 367.

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 GBC without a Web server on page 61.
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:
   http://myWebServer/gas/demos.html
4. Launch the GWC Demos program.
   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
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 367.

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** on page 368 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.

## Troubleshooting GAS configuration issues

Troubleshooting tips are provided for the most common issues encountered.

**Related concepts**

- **GIP Troubleshooting** on page 246
  Examining the Genero Identity Provider logs for error messages, will help resolve issues. You can also report issues to support.

- **Troubleshooting FastCGI** on page 105
  Troubleshooting tips addressing common issues with FastCGI.

### Proxy errors on Windows® platform

On Windows®, if you see 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:

```shell
cmd
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?

There are several reasons why an application does not start. Here are some basic troubleshooting procedures to follow as a standard approach to solving problems.

1. Check the application configuration (*xcf*) file - to ensure that all components are set properly.
2. 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.
3. 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 379 category filter to **CONFIGURATION**.
4. 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 programe` to enter the command-line debugger (`fgldb`). The application will then display in the Web browser. See Using the debugger on page 188.

**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 Browser Client (GBC) applications, see Configure the GBC development environment on page 171.

### 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.

**Related concepts**

**Troubleshooting HTTP errors** on page 121

Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

### Error taking session ownership

Here are some basic troubleshooting procedures to follow to help solve problems with GAS session ownership.

An error message is displayed in the dispatch logs when the GAS fails to take ownership of the session:

```
Failed to update the session lock file:
/Applications/fourjs/gas/3.20.04/appdata/session/httpdispatch/
session.owner.
[fail]
Failed to take session ownership
```

These are some things that should help you isolate the problem:

- Did you start the GAS using a different account to the one you used the last time? If so, check that the new account has permission to access the `session.owner` file. To resolve, set the appropriate permissions to the file.
- Have permissions been granted to create the `session.owner` file? To resolve, set the appropriate permissions to the session directory.
- Does your GAS have a multiple dispatcher configuration, and if so, is this the only GAS using this session directory? Check in the configuration file (`xcf`) of each GAS to ensure they are not starting with the same configuration for the session directories.

To resolve, make sure there are different configuration settings for the `appdata/session` directory resources. For more information see Why do my applications fail using multiple dispatchers? on page 107.

**Related concepts**

Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**Licensing examples**

Some scenarios and examples are illustrated where applications and GAS configurations may help you use licenses more efficiently.

For more information on licensing, see *Install and License your Genero Products* manual and your license agreement. For help with licensing in general and license purchase, contact your nearest Four Js sales office.

**Important:** This section 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 Genero Browser Client 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 where two runtime licenses are used.

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 call other applications using the Genero BDL RUN or the RUN WITHOUT WAITING command.

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

Related concepts
Licensing - base example on page 65
This scenario shows two user agents connected to two DVMs where two 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.
Related concepts
Licensing - base example on page 65
This scenario shows two user agents connected to two DVMs where two 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.

Related concepts
Licensing - base example on page 65
This scenario shows two user agents connected to two DVMs where two runtime licenses are used.

Genero front-ends and license counting
Information about how runtime licenses are consumed may help you use licenses as efficiently as possible.

A Genero license is not consumed at the MAIN statement of the application, but as soon as there is a user interface instruction in the code. This can be an instruction to open a window or form, to set an OPTIONS attribute, to display a menu, and so on.

If there is no UI instruction in the application, no license is consumed, even if it connects to a client. For example, no license is consumed if the application simply runs reports, accesses databases, completes file transfers, or makes front-end function calls.

TUI applications (applications run in text mode using a text-based user interface) and GUI applications (applications that use the Genero front-ends) have the same requirements for licenses.

There are licensing considerations specific to Web services and to the front-ends. These considerations are detailed in the following sections.
GDC applications

For Genero Desktop Client (GDC) applications, one license is consumed per GDC process. For example, if you have multiple GDC clients started on one machine, and each GDC client runs one or more applications, one license will be used per GDC.

To summarize: For each GDC that is running applications, no matter how many applications are running from this client, one license is consumed.

Web applications

For Genero Browser Client (GBC) applications, one license is consumed per application started. No additional licenses are consumed when an application is started by a RUN or a RUN WITHOUT WAITING request.

In the event that the user refreshes a browser page running a GBC application (regardless of whether it is the first application started or one started by RUN), the app is restarted.

In terms of licenses used, refreshing the browser consumes a new license. But as the first license is no longer active, it is only a matter of time before the DVM times out on the server and the license is released. See the section Freeing up licenses in this page.

DVM license use with RUN

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.

Web services

For Web service applications, one license is consumed per service started. If your Web service applications are secured by the Genero Identity Provider, additional licenses may be used. As GIP is based on micro services, each of its services your application calls consumes a license.

Genero Web service DVMs are managed in a pool by the GAS. Unlike application DVMs, the license is bound to the Web service process id. If a new child process is started, an additional license is consumed.

You can control the maximum number of licenses consumed by a Web service by configuring its MAX_AVAILABLE pool element. If the number of user requests exceeds what is available in the pool at any one time, the request will be queued (rather than blocked) until a DVM becomes available.

A Web service client does not consume a license. Users can make a request to a Web service server without needing a license.

Freeing up licenses

When a 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 the user does not exit the application, but instead closes the user agent (browser), 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.
Checking licenses in use

To determine the number of licenses used, run the license controller command on the application server where the Genero runtime is installed. For example run:

```
fglWrt -u
```

Followed by:

```
fglWrt -a info users
```

Recommendations

It is recommended to size your runtime license number to the number of users anticipated to run your applications. Another solution is to use CPU licenses. For more information see the CPU licenses topic in the Genero Licensing manual.

Related concepts

USER_AGENT on page 437
The USER_AGENT element sets a timeout to handle the user agent connection.

Setting the USER_AGENT timeout on page 69
With Genero Browser Client (GBC) applications, the USER_AGENT timeout element proves to be particularly useful in license management.

Licensing tips and tricks / troubleshooting on page 69
Identify GAS configuration elements that can impact licensing and identify additional licensing considerations.

Licensing tips and tricks / troubleshooting

Identify GAS configuration elements that can impact licensing and identify additional licensing considerations.

Configuring 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.

Related concepts

INTERFACE_TO_DVM on page 401
The INTERFACE_TO_DVM element specifies the address of the host where the GAS dispatcher runs.

Setting the USER_AGENT timeout

With Genero Browser Client (GBC) applications, the USER_AGENT timeout element proves to be particularly useful in license management.

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 GBC, 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.

Related concepts

USER_AGENT on page 437
The USER_AGENT element sets a timeout to handle the user agent connection.

**Setting MAX_AVAILABLE 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.

**Related concepts**
MAX_AVAILABLE on page 404
The MAX_AVAILABLE element specifies the maximum number of available DVMs to be attached to a Web service.

**Evaluate licensing when migrating from GDC**
The number of licenses consumed by an application can differ based on the client used. We recommend you evaluate your licensing needs when migrating applications from the Genero Desktop Client (GDC) to Genero Browser Client (GBC).

**Application time out**
GBC applications may not detect if the user has left the application, or 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 closes the browser without exiting the application, the application stays running and continues to consume a license. A timeout can be configured to release the DVM, which in turn releases the license. See Setting the USER_AGENT timeout on page 69.

**Licenses consumed per application**
For GDC applications, one license is consumed per monitor or console, no matter how many applications are run from this monitor.
For GBC applications, one license is consumed per application started, however no additional licenses are consumed if applications are started by a RUN or a RUN WITHOUT WAITING request. An application launcher or a start menu from your main application may be used to execute RUNs of sub-applications.

**License type options**
We recommend using a license of CPU type, instead of a User license, for your GBC applications as it provides for an infinite pool of licenses, thus simplifying the migration and overcoming the need for timeout or application reconfiguration. For more information see the CPU licenses topic in the Genero Licensing manual or for help with licensing options contact your nearest Four Js sales office.

**Related concepts**
Licensing examples on page 65
Some scenarios and examples are illustrated where applications and GAS configurations may help you use licenses more efficiently.

**Web Server configuration**
For the deployment of GAS on a production environment, you need to configure a Web server.
ISAPI Extension Installation and Web Server Configuration

Follow the procedures to configure a Microsoft® Internet Information Services (IIS) Web server for the GAS using ISAPI extensions and the isapidispatch dispatcher.

The Genero Application Server and IIS
The Internet Information Services (IIS) is configured with a GAS ISAPI Extension. 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 Install and License your Genero Products manual.

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

Related concepts
Setting up production environment on page 32
There are different options for configuring a GAS installation depending on your network. A Web server is required and security needs to be considered.

Related tasks
Manual configuration for IIS 8.x and IIS 10.x on page 71
These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 8.0 and IIS 10.0.

Manual configuration for IIS 7.x on page 84
These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 7.0.

Manual configuration for IIS 6.0 on page 94
These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 6.0.

Installing 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. 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.

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.

The installer needs some Internet Information Services (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:

1. Open Server Manager
2. Under the Manage menu, select Add Roles and Features to launch the Add Roles and Features Wizard
3. For Installation type, check the option Role-based or Feature-based Installation and click Next
4. For **Server Selection**, select the appropriate server, (the local server is selected by default) and click **Next**

5. For **Server Roles**, check Web Server (IIS) and click **Next**

6. For **Features**, accept the default settings and click **Next**

7. For **Web Server Role (IIS)**, accept defaults and click **Next**

8. 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**

9. For **Confirmation**, click the **Install** button. (When the IIS installation completes, the wizard shows the installation status in the **Results** screen)

10. Click **Close** to exit the wizard

*Activate IIS on Windows® 8 and 10*

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

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

**Figure 10: Administrative Tools window with IIS Manager highlighted**

2. Verify IIS is started. Open a browser and enter the URL: `http://localhost`. The IIS welcome screen displays.
Figure 11: IIS Welcome Screen

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.
Create application pool

You need to create an application pool that will allow you run Web and desktop applications. The GAS ISAPI Extension will be executed on behalf of the user that is registered in the pool's properties.

1. To create an application pool, perform the following steps:
   a) In Internet Information Services (IIS) Manager, right-click Application Pools and click Add Application Pool....
b) In the Add Application Pool... dialog, enter a name for the application pool, for example "GASAppPool".

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

d) In the Managed pipeline mode box, select Classic.

e) Click OK.

Figure 14: Internet Information Services (IIS) Manager Application Pools screen

Figure 15: Completed application pool configuration
2. To configure the application pool to run Web and desktop applications, perform the following steps:
   a) In Internet Information Services (IIS) Manager, right-click the application pool (GASAppPool in our example), and click Advanced Settings...
   b) 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 on page 437 timeout value of Genero 4GL applications that the application pool will run.
   c) In the Process Model area, set the Maximum Worker Processes field to 1.
   d) In the Recycling area, set the Disable Overlapped Recycle flag to True.
   e) In the Recycling area, set the Disable Recycling for Configuration Changes flag to True.
   f) In General area, if you want to start the pool at the same time as the Web server to decrease the initial response time, set the Start Mode flag to AlwaysRunning.
   g) Click OK.

3. To change the user identity that runs the GAS, perform the following steps:
   Important:
   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.
   a) In Internet Information Services (IIS) Manager, right-click the application pool (GASAppPool in our example), and click Advanced Settings...
   b) In the Advanced Settings dialog, in the Process Model area, click the Identity field.
   c) Click ... to open the Application Pool Identity dialog.
   d) According to your security policy, either select a built-in account or set a custom account.
   e) Click OK.
   f) In the Advanced Settings dialog, click OK.

4. Finally, check that you have configured your application pool creation correctly. Right click the application pool (GASAppPool in our example), and click Advanced Settings... again. Compare your settings with the following screenshot.
<table>
<thead>
<tr>
<th>Advanced Settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(General)</strong></td>
<td></td>
</tr>
<tr>
<td>.NET CLR Version</td>
<td>No Managed Code</td>
</tr>
<tr>
<td>Enable 32-Bit Applications</td>
<td>False</td>
</tr>
<tr>
<td>Managed Pipeline Mode</td>
<td>Classic</td>
</tr>
<tr>
<td>Name</td>
<td>GASAppPool</td>
</tr>
<tr>
<td>Queue Length</td>
<td>1000</td>
</tr>
<tr>
<td>Start Mode</td>
<td>AlwaysRunning</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td></td>
</tr>
<tr>
<td>Limit (percent)</td>
<td>0</td>
</tr>
<tr>
<td>Limit Action</td>
<td>NoAction</td>
</tr>
<tr>
<td>Limit Interval (minutes)</td>
<td>5</td>
</tr>
<tr>
<td>Processor Affinity Enabled</td>
<td>False</td>
</tr>
<tr>
<td>Processor Affinity Mask</td>
<td>4294967295</td>
</tr>
<tr>
<td>Processor Affinity Mask (64-bit option)</td>
<td>4294967295</td>
</tr>
<tr>
<td><strong>Process Model</strong></td>
<td></td>
</tr>
<tr>
<td>Generate Process Model Event Log Entry</td>
<td>True</td>
</tr>
<tr>
<td>Idle Time-out Reached</td>
<td>True</td>
</tr>
<tr>
<td>Identity</td>
<td>ApplicationPoolIdentity</td>
</tr>
<tr>
<td>Idle Time-out (minutes)</td>
<td>0</td>
</tr>
<tr>
<td>Idle Time-out Action</td>
<td>Terminate</td>
</tr>
<tr>
<td>Load User Profile</td>
<td>True</td>
</tr>
<tr>
<td>Maximum Worker Processes</td>
<td>1</td>
</tr>
<tr>
<td>Ping Enabled</td>
<td>True</td>
</tr>
<tr>
<td>Ping Maximum Response Time (seconds)</td>
<td>90</td>
</tr>
<tr>
<td>Ping Period (seconds)</td>
<td>30</td>
</tr>
<tr>
<td>Shutdown Time Limit (seconds)</td>
<td>90</td>
</tr>
<tr>
<td>Startup Time Limit (seconds)</td>
<td>90</td>
</tr>
<tr>
<td><strong>Process Orphaning</strong></td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>False</td>
</tr>
<tr>
<td>Executable</td>
<td>False</td>
</tr>
<tr>
<td>Executable Parameters</td>
<td>False</td>
</tr>
<tr>
<td><strong>Rapid-Fail Protection</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;Service Unavailable&quot; Response Type</td>
<td>HttpLevel</td>
</tr>
<tr>
<td>Enabled</td>
<td>True</td>
</tr>
<tr>
<td>Failure Interval (minutes)</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Failures</td>
<td>5</td>
</tr>
<tr>
<td>Shutdown Executable</td>
<td>True</td>
</tr>
<tr>
<td>Shutdown Executable Parameters</td>
<td>True</td>
</tr>
<tr>
<td><strong>Recycling</strong></td>
<td></td>
</tr>
<tr>
<td>Disable Overlapped Recycle</td>
<td>True</td>
</tr>
<tr>
<td>Disable Recycling for Configuration Changes</td>
<td>True</td>
</tr>
<tr>
<td>Generate Recycle Event Log Entry</td>
<td>True</td>
</tr>
<tr>
<td>Application Pool Configuration Changed</td>
<td>True</td>
</tr>
<tr>
<td>Isapi Reported Unhealthy</td>
<td>True</td>
</tr>
<tr>
<td>Manual Recycle</td>
<td>True</td>
</tr>
<tr>
<td>Private Memory Limit Exceeded</td>
<td>True</td>
</tr>
<tr>
<td>Regular Time Interval</td>
<td>True</td>
</tr>
<tr>
<td>Request Limit Exceeded</td>
<td>True</td>
</tr>
<tr>
<td>Specific Time</td>
<td>True</td>
</tr>
<tr>
<td>Virtual Memory Limit Exceeded</td>
<td>True</td>
</tr>
<tr>
<td>Private Memory Limit (KB)</td>
<td>0</td>
</tr>
</tbody>
</table>
Create an application

You need to create an application that is bound to the GAS ISAPI extension dispatcher. You need to configure authentication for the anonymous user, or a user based on your security policy.

1. To bind an application to the GAS ISAPI Extension dispatcher, perform the following steps:
   a) Create a directory on your disk that is the Application root directory on page 96. The recommended directory is for example $FGLASDIR/ISAPI.
   b) 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...

   ![Figure 17: Internet Information Services (IIS) Manager Default Web Site Home screen](image)

   c) In the Add Application dialog, enter the alias of your application, for example "gas".
   d) Enter the physical path to the directory created in step 1.
   e) Click Select...
   f) In the Select Application Pool dialog, select the application pool that you have defined previously.
   g) Click OK.
   h) Select Enable Preload.
i) In the **Add Application** dialog, click **OK**.

![Completed application configuration](image)

**Figure 18: Completed application configuration**

2. Configure authentication for the anonymous user.

   If all users have access to the application, the identity of the anonymous user should be configured as shown in the following steps:
   a) In **Internet Information Services (IIS) Manager**, select your application.
   b) In the **Features View** panel, double-click the **Authentication** icon.
   c) In the **Authentication** feature, select the **Anonymous Authentication** line.
   d) Ensure that the status is **Enabled**.
   e) In the **Actions** area, click **Edit...**
   f) In the **Edit Anonymous Authentication Credentials** dialog, select Application pool identity.
   g) Click **OK**.

![Configuration for anonymous authentication](image)

**Figure 19: Configuration for anonymous authentication**

3. To bind the application to the GAS ISAPI Extension dispatcher perform the following steps:
a) In **Internet Information Services (IIS) Manager**, select your application.
b) In the **Features View** panel, double-click the **Handler Mappings** icon.
c) In the **Handler Mappings** feature, in the **Actions** area, click **Add Wildcard Script Map**...
d) In the **Add Wildcard Script Map** dialog, enter the path to the GAS ISAPI Extension DLL: `$FLASDIR\bin\isapidispatch.dll`.
e) Enter a name for this mapping, for example "GAS ISAPI Extension".
f) Click **OK**.
g) To the question **Do you want to allow this ISAPI extension?**, click Yes.
h) In the **Handler Mappings** feature, in the **Actions** area, click **View Ordered List**...
i) Ensure that the GAS ISAPI Extension is at the top of the list.
j) Click **View Unordered List**...
k) Select the GAS ISAPI Extension and in the **Actions** area, click **Edit**
l) In the **Edit Script Map** dialog, click on **Request Restrictions**...
m) In the **Request Restrictions** dialog's **Mapping** panel, clear the **Invoke handler only if requests is mapped to**: option.
n) In the **Request Restrictions** dialog's **Access** panel, select the **Script** option.
o) Click **OK**.
Figure 20: Binding application to GAS ISAPI extension

4. To allow execution of the ISAPI extension perform the following steps:

Although the GAS ISAPI Extension has been allowed automatically when you answered Do you want to allow this ISAPI extension? with Yes in the last step, these steps allow you to do it manually and set a description.

a) In Internet Information Services (IIS) Manager, select the root node, the one that contains the host name.

b) In the Features View panel, double-click the ISAPI and CGI Restrictions icon.
c) In the ISAPI and CGI Restrictions feature, in the Actions area, click Add...

d) In the Add ISAPI or CGI Restriction dialog, enter the path to the GAS ISAPI Extension DLL: $FGLASDIR\bin\isapidispatch.dll.

e) Enter a description, for example "GAS ISAPI Extension".

f) Ensure that the Allow extension path to execute checkbox is selected.

g) Click OK.
Figure 22: ISAPI CGI restrictions configuration

After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see Finishing the installation on page 95.

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 Internet Information Services (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 23: 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.

   ![Figure 24: Control Panel with IIS Manager highlighted](image)

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 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 on page 437 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.

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.
7. Finally, check that you have configured your application pool creation correctly. Right click the application pool (GASAppPool in our example), and click Advanced Settings... again. Compare your settings with the following screenshot.

**Figure 28: Completed application pool configuration**

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 on page 437 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.

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.
7. Finally, check that you have configured your application pool creation correctly. Right click the application pool (GASAppPool in our example), and click Advanced Settings... again. Compare your settings with the following screenshot.
**Advanced Settings**

### (General)
- **.NET Framework Version**: No Managed Code
- **Managed Pipeline Mode**: Classic
- **Name**: GASAppPool
- **Queue Length**: 1000
- **Start Automatically**: True

### CPU
- **Limit**: 0
- **Limit Action**: NoAction
- **Limit Interval (minutes)**: 5
- **Processor Affinity Enabled**: False
- **Processor Affinity Mask**: 4294967295

### Process Model
- **Identity**: ApplicationPoolIdentity
- **Idle Time-out (minutes)**: 0
- **Load User Profile**: True
- **Maximum Worker Processes**: 1
- **Ping Enabled**: True
- **Ping Maximum Response Time (seconds)**: 90
- **Ping Period (seconds)**: 30
- **Shutdown Time Limit (seconds)**: 90
- **Startup Time Limit (seconds)**: 90

### Process Orphaning
- **Enabled**: False
- **Executable**: False
- **Executable Parameters**: False

### Rapid-Fail Protection
- **"Service Unavailable" Response Type**: HttpLevel
- **Enabled**: True
- **Failure Interval (minutes)**: 5
- **Maximum Failures**: 5
- **Shutdown Executable**: False
- **Shutdown Executable Parameters**: False

### Recycling
- **Disable Overlapped Recycle**: True
- **Disable Recycling for Configuration Change**: True
- **Generate Recycle Event Log Entry**: False
  - Application Pool Configuration Change: False
  - Isapi Reported Unhealthy: False
  - Manual Recycle: False
  - Private Memory Limit Exceeded: True
  - Regular Time Interval: True
  - Request Limit Exceeded: False
  - Specific Time: False
  - Virtual Memory Limit Exceeded: True
  - Private Memory Limit (KB): 0
  - Regular Time Interval (minutes): 1740
  - Request Limit: 0
  - Specific Times: TimeSpan[] Array
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 your disk that will be the Application root directory on page 96, the recommended 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".
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.

![Edit Application](image)

Figure 30: Completed application configuration

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 follows:

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, select 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 selected.
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 95.

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 Recycle 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 on page 437 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 directory on page 96. 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 95.

**Finishing the installation**

The GAS ISAPI Extension is ready to test once you have added the isapidispatch configuration file.

**Before you begin:**

It is assumed you have your ISAPI already configured for the version of Internet Information Services (IIS) you have. See Installing the ISAPI dispatcher on page 71.
1. Copy the `isapidispatch.ini` file from $FGLASDIR\etc to your Application root directory on page 96 (for example, $FGLASDIR\ISAPI).
   - 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 96 for details about the content of this file.

2. 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.

**Related concepts**

URI Examples on page 48
Several URI examples with ways to help you launch applications.

**Application root directory**

The application root directory is the physical directory on your disk the IIS uses as the starting-point for your applications on the web.

You create the application root directory (usually called "ISAPI") in the GAS installation directory path, for example, $FGLASDIR\ISAPI. You map the application root directory to a virtual directory or path, typically called "gas", in the IIS configuration. The virtual directory name is part of application URLs, whereas the physical directory is never part of your URL.

**Note:** While IIS allows you to map more than one wildcard application to a virtual directory, the GAS ISAPI Extension one should be the last on the virtual directory list, as otherwise it will not forward URLs to other ones.

All requests having an URL that begins with the virtual directory name, for example "/gas/", will be served by the GAS ISAPI Extension. As in the standalone GAS, the GAS ISAPI Extension serves applications configured for the GAS, such as demos, resource files, and static files.

The GAS ISAPI Extension configuration file, `isapidispatch.ini`, should be the only file present in the application root directory. Files in the application root directory are not served by IIS. There is no way to configure IIS to change this behavior.

**Related tasks**

Finishing the installation on page 95
The GAS ISAPI Extension is ready to test once you have added the isapidispatch configuration file.

**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 Application root directory on page 96 (for example, $FGLASDIR\ISAPI). It is read on start up 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 9: GAS ISAPI Extension 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>
</tbody>
</table>
Sample isapidispatch.ini

```plaintext
# 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=
```

**Related tasks**

[Finishing the installation](#) on page 95

The GAS ISAPI Extension is ready to test once you have added the isapidispatch 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
Branding ............................ [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 96.

In all other cases, check the installation process.

**Related tasks**

[Finishing the installation](#) on page 95

The GAS ISAPI Extension is ready to test once you have added the isapidispatch configuration file.

**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 on page 437 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
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 has the same lifetime: starting the Web server starts the GAS process, while stopping the Web server stops 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 345.

Related concepts
Dispatcher on page 38
Understand what the dispatcher does and identify which dispatcher to use with a specific Web server.

Tools and Commands on page 343
Information about the dispatchers, proxies, and command line utilities.

FastCGI GAS configuration on various Web Server
Understanding and implementing FastCGI on various servers.

The Apache module for FastCGI support differs for versions of Apache. You need to check your version of Apache. For versions prior to 2.4, you configure mod_fastcgi. If you are using Apache 2.4, configure mod_proxy_fcgi instead. See the Apache documentation.

A common URL launches a Genero application using the Genero Application Server, regardless of your FastCGI configuration.

A typical URL would be:

```
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 for versions of Apache prior to 2.4.

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 100.

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**

In your Apache configuration file (for example, `httpd.conf`):

- Enable the load module directive for `mod_fastcgi` by removing the hash symbol (#) from the start of the line. Be aware that your configuration may be different to that shown in the example.
- Add the lines to configure fastcgi server and set permissions for the GAS.

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

<IfModule fastcgi_module>
  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 ) resulting 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 169), 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.

**Configuring mod_fastcgi for external GAS**

Configure fastCGI for external GAS with the mod_fastcgi module configuration.

If you have GAS on an external server, you must configure `mod_fastcgi` in your Apache configuration for this. In your Apache configuration file (for example, `httpd.conf`):

- Check that the load module directive for `mod_fastcgi` is enabled. The hash symbol (#) must be removed from the start of the line. Be aware that your configuration may be different to that shown in the example.
• Add the lines to configure fastcgi on the external GAS.

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

<IfModule fastcgi_module>
  FastCgiExternalServer /gas -idle-timeout 300 -host <gas-server-ip>:<gas-server-port>
  Alias /gas /gas
</IfModule>
```

**Note:**

• "-idle-timeout" must be greater than `REQUEST_RESULT` timeout in GAS configuration. [By default: `REQUEST_RESULT`: 240 seconds, `mod_fastcgi` -idle-timeout: 300 seconds]

• `/gas` directory is just a virtual directory, no need to create one.

For more information see the *Apache: mod_fastcgi* topic in the *Genero Application Server User Guide version 2.30*

**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, check your Apache configuration file for the load module directives for these modules; bearing in mind that your configuration may be different. Make sure the hash symbol (#) is removed from the start of the line.

   ```
   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:

   • To ensure application URLs with spaces are decoded correctly, `proxy-fcgi-pathinfo` must be set to `unescape`. There is no PATH_INFO with `mod_proxy_fcgi` unless this is set.

   • You can add the `enablereuse=on` option in the `ProxyPass` configuration line, in order to recycle connections to the fastcgi dispatcher.

   • To avoid timeout issues, set the `ProxyPass timeout` greater than the `REQUEST_RESULT` GAS configuration. If, for instance, `REQUEST_RESULT` is set to 60 seconds (default setting), you need to ensure you set the `ProxyPass timeout` higher at 100 seconds.

   ```
   ...<IfModule proxy_fcgi_module>
      # Unescapes the path component of the request URL
      SetEnvIf Request_URI . proxy-fcgi-pathinfo=unescape

      # Maps the fastcgi server to the GAS URL space
      ProxyPass /gas/ fcgi://localhost:6394/ enablereuse=on timeout=100
   </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.

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

**Related concepts**

Troubleshooting FastCGI on page 105
Troubleshooting tips addressing common issues with FastCGI.

**Related tasks**

Apache 2.4: Configure mod_proxy_fcgi for remote server on page 101
Configure Apache 2.4 fastCGI with mod_proxy_fcgi module for a remote GAS server. If authorization is used, configure a rewrite rule.

**Apache 2.4: Configure mod_proxy_fcgi for remote server**
Configure Apache 2.4 fastCGI with mod_proxy_fcgi module for a remote GAS server. If authorization is used, configure a rewrite rule.

If you are configuring mod_proxy_fcgi to connect to a remote GAS server, your Apache configuration file needs to be configured correctly.

**Note:** Starting with Apache 2.4.11:

- To ensure application URLs with spaces are decoded correctly, `proxy-fcgi-pathinfo` must be set to `unescape`. There is no PATH_INFO with mod_proxy_fcgi unless this is set.
- You can add the `enablereuse=on` option in the `ProxyPass` configuration line, in order to recycle connections to the fastcgi dispatcher.
- To avoid timeout issues, set the `ProxyPass` timeout greater than the REQUEST_RESULT GAS configuration. If, for instance, REQUEST_RESULT is set to 60 seconds (default setting), you need to ensure you set the `ProxyPass` timeout higher at 100 seconds.

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, check your Apache configuration file for the load module directives for these modules; bearing in mind that your configuration may be different. Make sure the hash symbol (`#`) is removed from the start of the line.

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

3. Add the following configuration to the GAS virtual host:
   These directives typically need to be added to your GAS virtual host configuration (identified by the element `<VirtualHost>`) for both HTTP and HTTPS.

   ```
   ...<VirtualHost _default_>
   <IfModule proxy_fcgi_module>
     # Unescapes the path component of the request URL.
     SetEnvIf Request_URI . proxy-fcgi-pathinfo=unescape

     # Maps the fastcgi server to the GAS URL space
     enablereuse=on timeout=100
   </IfModule>
   ```
Where:

- `<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.

**Rewrite-rule for authorization headers**

If you are using authorization headers, the module `mod_rewrite` must be enabled in your Apache configuration file. For example, the load module directive for this module must appear without the hash (#) at the start of the line:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

Also you need to be aware of the following:

- Authorization headers need to be base64 encoded. Apache discards the Authorization header if it is not a base64-encoded user/password combination. A rewrite rule (see [Apache 2.4 documentation](https://httpd.apache.org/docs/2.4/rewrite/)) can be used to rewrite it from the server variable.
- A `ProxyPassReverse` directive is set to adjust or rewrite URLs in response headers 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, see [Apache 2.4 documentation](https://httpd.apache.org/docs/2.4/).

An example configuration is shown:

```
...
<VirtualHost _default_>
  <IfModule proxy_fcgi_module>
      ...
      # Unescapes the path component of the request URL
      SetEnvIf Request_URI . proxy-fcgi-pathinfo=unescape

      # Recreates the authorization header from the %{HTTP:Authorization} variable
      RewriteEngine on
      RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

      # Maps the fastcgi server to the GAS URL space
      enablereuse=on timeout=100

      # Rewrites URL in response headers
  </IfModule>
</VirtualHost>
...
```

**Related concepts**

[Troubleshooting FastCGI](#) on page 105
Troubleshooting tips addressing common issues with FastCGI.

**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 PATHINFO $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.

**Fastcgi for Lighttpd**

The Lighttpd Web server supports the FastCGI protocol natively.

**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_RESULT timeout in the GAS configuration.

**Sun® Java System Web Server 7.0**

Sun® Web Server 7 has 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_RESULT timeout in the GAS configuration.
Fastcgi for Windows
FastCGI protocol is not supported on Windows® systems.

Note: Fastgidispatch is supported on UNIX™-like platforms only. While fastgidispatch is known to work on Windows®, it is not recommended and is not supported. For Windows® systems, the Genero Dispatcher: isapidispatch on page 346 dedicated to the Internet Information Services (IIS) Web server is preferred.

Troubleshooting FastCGI
Troubleshooting tips addressing common issues with FastCGI.

Related concepts
GIP Troubleshooting on page 246
Examining the Genero Identity Provider logs for error messages, will help resolve issues. You can also report issues to support.

Troubleshooting GAS configuration issues on page 63
Troubleshooting tips are provided for the most common issues encountered.

Why does my application timeout on Apache?
There are several reasons why a timeout occurs and both GAS and Web server logs might need to be investigated.

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.

Related concepts
Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

AUTO_LOGOUT_COMPONENT on page 377
The AUTO_LOGOUT_COMPONENT element creates a component, which defines a mechanism for triggering and handling auto-logout events.

WEB_APPLICATION_TIMEOUT_COMPONENT on page 439
The WEB_APPLICATION_TIMEOUT_COMPONENT element defines a set of timeout values to be used when configuring a Web application.

TIMEOUT (for an application) on page 431
This element set timeouts for process requests and the start of the DVM.

Why does my application not work with fastcgi?
Incorrect fastcgi configurations and/or user permissions on directories and files are some of the most common reasons your application fails 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**

If the fastcgi does not start and the Genero Application Server displays "Invalid installation directory", examine the environment and the GAS log directory permissions.

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.

**Related concepts**

- **Log files** on page 185
  When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**Application URL not decoding**

If the application URL does not decode correctly, examine the Apache configuration for mod_proxy_fcgi.

There is a known Apache bug (bug 55329) that prevents mod_proxy_fcgi decoding application URLs with spaces or special characters correctly, which may cause this issue. This bug prevents, for example, applications with URLs like "http://server/gas/wa/r/test/220%2012", from working. This bug has been fixed for Apache 2.4.11.

Check the version of Apache you have:

- If you are using Apache 2.4.11 or later check that the Apache proxy-fcgi-pathinfo configuration is set:

  ```
  SetEnvIf Request_URI . proxy-fcgi-pathinfo=unescape
  ```

  For more details, see Apache 2.4: mod_proxy_fcgi on page 100 or Apache 2.4: Configure mod_proxy_fcgi for remote server on page 101.

- If you are using a version of Apache prior to 2.4.11, as the Genero BDL developer you need to be aware that you can get spaces encoded in URLs and need to handle these in your program code. It is recommended that you decode URLs before using them.

**Applications all down at the same time**

The most common application issues and how to resolve them are described.

You can determine the source of most fastcgi issues by examining messages in the log files produced by the GAS and/or the Web server. If the GAS is down, you can find some clues in:

- GAS logs (see your appdata/log/<dispatcher-name>/date directory)
- 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 error_log 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
```
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.

**Related concepts**

Log files on page 185

When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**Why do my applications fail using multiple dispatchers?**

Make sure that each dispatcher has its own configuration file and its own session directory.

Examine the fastcgidispatch.log of your application. If you see error messages like those shown, it indicates that the dispatcher was running but has been stopped as another dispatcher is starting with the same configuration for the session directories.

```
10:59:49.791098 890.969044 [session.c:794] 10089 2 - "Session" The session owner has changed
10:59:49.791468 890.969415 [session_timeout.c:491] 10089 2 - "Session timeout" Session owner has changed, exiting
```

To resolve, review your xcf file configuration; specially for session (SESSION_DIRECTORY) and log (LOG on page 403) directories. When configuring several dispatchers on the same host, make sure that each dispatcher has its own configuration file (as a copy of main GAS as.xcf) with different configuration settings for the following resources:

- appdata/session directory
- appdata/log directory
- appdata/tmp directory

Create separate session directories for each dispatcher on disk. Then change the default resources in the configuration file to use these directories. For example change:

```
<SESSION_DIRECTORY>${res.appdata.path}/session</SESSION_DIRECTORY>
```

To:

```
<SESSION_DIRECTORY>${res.appdata.path}/session_dispatcher_1</SESSION_DIRECTORY>
```

Apply similar changes for log and tmp. Restart the dispatcher and try again.

**Specifying separate appdata directories**

If you dedicate a GAS for Web application and another one for Web services applications on the same host, then it is not recommended to share the appdata/app and appdata/services directories with the main GAS as.xcf. You can override the res.appdata.path resource with, for example, the dispatcher -E option:

```
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 deploying applications with the `gasadmin gar` command, in order to deploy to the correct directory.

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 gar -E res.appdata.path=/work/tmp/gas/appdata --deploy-archive myapp.gar
  ```

**Related concepts**

- [Configure multiple dispatchers](#) on page 171
  
  If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

- [Error taking session ownership](#) on page 64
  
  Here are some basic troubleshooting procedures to follow to help solve problems with GAS session ownership.

- [GAS installation and application data directories](#) on page 40
  
  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.

- [Restarting fastcgidispatch without losing current sessions](#)
  
  Describes a method to stop the fastcgidispatch process that leaves the sessions alive and untouched.

  If fastcgidispatch fails to respond, you can use the `kill` command to stop its process. This allows for the potential of restoring the current sessions on restart; provided the service is restarted within a short time span of about 1 minute:

  ```
  kill -9 pid
  ```

  The "pid" is the process id of the running dispatcher. The command stops the process yet the sessions remain alive and untouched. When the dispatcher is restarted, the sessions continue to be active.

  **Note:** The fastcgidispacher can be managed via the Linux® service commands, such as `service fastcgidispatch restart`, etc.

  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](#)
  
  In contrast to using the `kill -9 pid` method, the effect of pressing 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 pressing CTRL+C as well if started in standalone mode - but not on SIGTERM.

**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?

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:

  ```xml
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfas.xsd"
  ```

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

  ```xml
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextwa.xsd"
  ```

- For Web services application configuration files, this entry should exist:

  ```xml
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextws.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.

Related concepts

Application configuration files for GBC, GWS, and GDC on page 114

Examples are given for configuring Web applications, Web services, and GDC applications.

Validating with the gasadmin tool

The gasadmin tool features options that can prove useful in validating the GAS configuration file and external application configuration files.

configuration-check

```bash
gasadmin config --configuration-check
```

When you run the `gasadmin config` command with the `--configuration-check` (`-e`), it validates the GAS configuration file (as.xcf) and exits. Errors are displayed to the standard output.

configuration-explode

```bash
gasadmin config --configuration-explode
```

When you run the `gasadmin config` command with the `--configuration-explode` (`-e`) option, it explodes the GAS configuration into a hierarchy of configuration elements and outputs them to file in XML format.
One file is created for each application. Each file lists the entire configuration for an application, expanding the inherited components.

```xml
<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**

```bash
gasadmin config -e --configuration-expand-resources
```

When you use the `--configuration-expand-resources (-r)` option with the `--configuration-explode (-e)` it replaces the GAS configuration file's resources and its parent resources with real values, and expand them into XML files.

It can be used in combination with `--configuration-explode (-e)` or `--configuration-explode-external (-t)` options.

```xml
<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>
```

**Related concepts**

`gasadmin tool` on page 348

The `gasadmin` tool is provided as an administrative command for the Genero Application Server.

**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 (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.

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

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.

To configure your application to inherit the settings of the abstract application, you 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.

  Important: When you add an 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).

  You can save the xcf in the default application group, $(res.appdata.path)/app, directory, or in your own created application group. The application is then immediately available without having to do a GAS restart. Alternatively, you can deploy the app.

Related concepts

Application configuration file on page 113
You create an application configuration file to provide the Genero Application Server (GAS) with the information needed to run an application.

GAS configuration file (as.xcf) on page 360
The Genero Application Server is configured through a configuration file. The default configuration file is as.xcf, located in the $FGLASDIR/etc directory.

APPLICATION_LIST on page 374
The APPLICATION_LIST element provides a list of default groups and abstract application configurations.

Abstract applications

An application configuration element can specify a parent attribute that references an abstract application that provides default configurations.

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 abstract Web application

The default application for the Genero Web client, defaultwa, 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"/>
</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 Browser Client (GBC).

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

Use groups to define aliases for directories where application configuration files can be stored.

A group consists of an alias (Id) and a directory (path). A GROUP (for an application) on page 397 element can be added to the APPLICATION_LIST component within the GAS configuration file.

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).

Organize your applications

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 accounting 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.

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

The resource $(res.path.app) resolves to appdata/app. The appdata directory is described in GAS installation and application data directories on page 40. 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.

**Related concepts**

GROUP (for a service) on page 397

This GROUP element allows you to specify a directory where external Web service application configuration files are located.

**Application configuration file**

You create an application configuration file to provide the Genero Application Server (GAS) with the information needed to run an application.

Typically, the name of the configuration file matches the name of the application and has an .xcf extension. For example, if the application name was "appl", create a configuration file named appl.xcf.

The configuration file defines an application environment, and starts with the Application element. Within this element, you can:

- define local resources
- specify the execution environment
- specify the timeout, resources, and output settings
- refer to previously defined components by using the attribute Using

The organization of the elements within the application configuration file depend on the type of application. See Configuration for Web and service applications on page 360.

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 element:

```
<GROUP Id="_default">directory</GROUP>
```

You can specify alternate directories; see GROUP (for an application) on page 397 or GROUP (for a service) on page 397.
The application configuration file becomes available once added to the GAS. It is re-read at each application launch. There is no need to restart the GAS after modifying an application configuration file.

This examples shows a well-formed external application configuration files.

**Example - 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.

Things to note in this example:

1. The `Id` attribute of `<APPLICATION>` element is omitted; even if included, its value is not read.
2. The `Parent` application is `defaultwa`. See Abstract applications on page 111.
3. The path to the application executables is defined by the `PATH` component. See PATH (under EXECUTION) on page 410.
4. The `MODULE` element can specify the name of the .42r module to run. In this example the module name is the same as the configuration file name, therefore the element is not necessary. See MODULE on page 407.

```xml
<APPLICATION Parent="defaultwa"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextwa.xsd">

<EXECUTION>
  <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
</EXECUTION>
</APPLICATION>
```

**Related concepts**

Application configuration files for GBC, GWS, and GDC on page 114
Examples are given for configuring Web applications, Web services, and GDC applications.

GAS configuration file (as.xcf) on page 360
The Genero Application Server is configured through a configuration file. The default configuration file is `as.xcf`, located in the `$FGLASDIR/etc` directory.

Application configuration files for GBC, GWS, and GDC
Examples are given for configuring Web applications, Web services, and GDC applications.

You can create an application configuration in the GAS `as.xcf` or in a separate application `xcf` file. The configuration for GAS and external file are basically the same, with some minor differences.

The examples show well-formed application configurations for the sample Genero Business Development Language demo application. See Genero demo applications on page 29.

**Configure Web applications**

What do you need to configure a Genero Browser Client application?

To add an application for Genero Browser Client (GBC), 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 (`defaultwa` in this example).
- The path to the compiled application files.
- The name of the application to launch.

These examples show well-formed application configurations for the sample Genero BDL demo application.
**Example 1: simple application for GBC**

This configuration is for a GBC application defined in the GAS configuration file (`as.xcf`).

```xml
<Application Id="my-gwc-demo" Parent="defaultwa">
  <Execution>
    <Path>$(res.path.fgldir.demo)</Path>
    <Module>demo</Module>
  </Execution>
</Application>
```

1. The application inherits the configuration settings of its parent ("defaultwa" in this example).
2. The path used is a `RESOURCE` defined in the `as.xcf`; you could also use the absolute path pointing to your application files.
3. The `MODULE` contains the name of the application to launch.

**Important:** When you add an application to the GAS configuration file, you must restart the GAS for the application to be recognized.

To launch your application in the browser, you can use this kind of URL:

http://appserver:6394/ua/r/my-gwc-demo

**Example 2: gwc-demo-external.xcf**

The main differences between this example and the example shown in Example 1: simple application for GBC on page 115 are the lack of the `Id` attribute and the addition of the reference to the XML schema.

```xml
  <Execution>
    <Path>$(res.path.fgldir.demo)</Path>
    <Module>demo</Module>
  </Execution>
</Application>
```

After configuring your application, you can test it to see if it is configured correctly by saving the `xcf` file in your `$(res.appdata.path)/app` directory and running it on the GAS.

You can use this kind of URL to launch your application in the browser:

http://appserver:6394/ua/r/gwc-demo-external

See **URI Examples** on page 48.

**Related concepts**

- [Configure applications for Web service](#) on page 116
- Create an application configuration file (`.xcf`) for a Web services application.
- [Configure GDC applications](#) on page 117
- What do you need to configure a Genero Desktop Client (GDC) application?
- [GAS installation and application data directories](#) on page 40
  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.
- [Configure delegation for application or service](#) on page 127
To delegate the start of an application or service to the Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

**Configure applications for Web service**
Create an application configuration file (.xcf) for a 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)
  
  **Important:** The parent referenced must contain the prefix "ws.". For example, these are valid references: Parent="ws.default", Parent="ws.myservice".

- 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.
- If you are using IIS as your Web server, set the COMSPEC environment variable in your Web service application configuration file.

  **Tip:** Set the value to what the DOS console returns for "echo %COMSPEC%".

  This variable must be set properly when the gwsproxy starts the fglrun process. You may have this environment variable set by IIS, however situations exist where the setting was not passed on; we ask that you explicitly set it as part of the configuration file for the Web service.

These examples show well-formed application configuration for Web service applications.

**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.

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 pointing to your application files.

```xml
<APPLICATION Id="calculator" Parent="ws.default">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/WebServices/calculator/server</PATH>
    <MODULE>calculatorServer</MODULE>
  </EXECUTION>
</APPLICATION>
```

**Important:** When you add an application to the GAS configuration file, you must restart the GAS for the application to be recognized.

**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 116. The main differences are the lack of the Id attribute and the addition of the reference to the XML schema.

**Note:** As 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.
1. The **ALLOW_FROM** on page 370 element specifies from what hosts access is allowed, the example here is defined in the **res.access.control** resource.

2. The **POOL** on page 411 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 113 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 267. For REST Web services URI examples, see **REST URI Examples**.

```xml
<APPLICATION Parent="ws.default"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/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>
```

**Related concepts**

**Working with Genero Web Services** on page 267
Familiarize yourself with the URLs used to access Web services applications deployed on the GAS and to get their WSDL (SOAP) or OpenAPI specification (REST). Learn how to validate they are working and troubleshoot failures, or configure a "sticky" Web services.

**Services Pool (GWS Only)** on page 34
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.

**Configure Web applications** on page 114
What do you need to configure a Genero Browser Client application?

**Configure GDC applications** on page 117
What do you need to configure a Genero Desktop Client (GDC) application?

**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
- For information on running applications via the GDC monitor see URI examples **Running desktop applications**.

These examples show well-formed application configuration for the sample Genero BDL demo application.
Example: simple configuration for GDC application

In the following example the configuration is for a GDC application defined in the GAS configuration file.

1. The application inherits the configuration settings of its parent ("defaultgdc" in this example).
2. The path used in this example is a RESOURCE defined in the as.xcf; you could also use the absolute path to your application files.
3. The MODULE contains the name of the application to launch.

```
<APPLICATION Id="my-app" Parent="defaultgdc">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo</MODULE>
  </EXECUTION>
</APPLICATION>
```

Important: When you add an application to the GAS configuration file, you must restart the GAS for the application to be recognized.

You can use this kind of URL to launch your application via the GDC monitor:


Example: gdc-demo-external.xcf

This external configuration file would accomplish the same task as the Example: simple configuration for GDC application on page 118 that is defined in the GAS configuration file. The only differences are the lack of the Id attribute and the addition of the reference to the XML schema.

```
<APPLICATION Parent="defaultgdc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextwa.xsd">
  <EXECUTION>
    <PATH>${res.path.fgldir.demo}</PATH>
    <MODULE>demo</MODULE>
  </EXECUTION>
</APPLICATION>
```

After configuring your application, you can test it to see if it is configured correctly by saving the xcf file in your $(res.appdata.path)/app directory and running it on the GAS.

You can use this kind of URL to launch your application via the GDC monitor:

http://appserver:6394/da/r/gdc-demo-external

Related concepts

Configure Web applications on page 114
What do you need to configure a Genero Browser Client application?

Configure applications for Web service on page 116
Create an application configuration file (.xcf) for a Web services application.

GAS installation and application data directories on page 40
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.

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 on page 388 elements in the application configuration file.
Example

Example (using Informix® database client):

```xml
<Application Id="myapp" Parent="defaultgwc">
  <Execution>
    <EnvironmentVariable Id="DBDATE">DBMY4</EnvironmentVariable>
    <EnvironmentVariable Id="FGLRESOURCEPATH">/home/myapp/resources</EnvironmentVariable>
    <EnvironmentVariable Id="INFORMIXDIR">/opt/informix</EnvironmentVariable>
    <EnvironmentVariable Id="INFORMIXSERVER">ORION</EnvironmentVariable>
    <EnvironmentVariable Id="LD_LIBRARY_PATH">/opt/informix/lib:...</EnvironmentVariable>
    <Path>/home/myapp/bin</Path>
    <Module>app.42r</Module>
  </Execution>
</Application>
```

Related concepts

Use a script to set the environment on page 119
Specify a command script for the application to launch the application.

Use a script to set the environment

Specify a command script for the application to launch the application.

As an alternative to defining `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™

Use the `DVM` element to define a shell that will execute a script specified with the `MODULE` element:

```xml
<Application Id="kiosk" Parent="defaultgwc">
  <Execution>
    <Path>/home/f4gl/gep/configfiles/officestoredemo</Path>
    <Dvm>/bin/sh</Dvm>
    <Module>gdc-kiosk.sh</Module>
  </Execution>
</Application>
```

1. `Path` element specifies where the script is stored.
2. `DVM` element defines the command to execute the shell script defined in `MODULE`.
3. `MODULE` element specifies the shell script file.

On Windows®

Use the `DVM` element to specify a `.bat` 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` file 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.gbc.user resource at runtime

Specify your customized Genero Browser Client (GBC) with the dispatcher override switch (-E) at runtime.

The res.path.gbc.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 GBC_LOOKUP_PATH path list. Therefore, the GAS is able to find the GBC it references.

Override the resource with the dispatcher override switch (-E).

For example, start the dispatcher as shown:

```
httpdispatch -E res.path.gbc.user=c:\<my_gbc_directory>
```

If the specified customization directory is not found, the default GBC directory at $(FGLDIR)/web_utilities/gbc path is used.

Related concepts
GBC_LOOKUP_PATH on page 395
The GBC_LOOKUP_PATH element specifies the location of installed Genero Browser Client (GBC) front-ends.

Override the appdata resource at runtime

If you have changed the default application directory (known as "appdata") location, use this override method to specify it 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:

```
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 deploying applications with the gasadmin gar command, in order to deploy to the correct directory.

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 gar -E res.appdata.path=/work/tmp/gas/appdata --deploy-archive myapp.gar
  ```

Related concepts
GAS installation and application data directories on page 40
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.

Configure multiple dispatchers on page 171
If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

**Next steps**

Other topics provide more details on application configuration and deployment.

See also:
- Developing Web applications and Web services on page 259
- Working with Genero Web Services on page 267
- Steps to packaging applications on page 273
- Application List Reference

**Troubleshooting application Issues**

Troubleshooting tips are provided for the most common issues encountered.

**Troubleshooting HTTP errors**

Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

**Viewing information about the GAS**

You need to have access to the following logs:

**Web services logs:**

- The dispatcher log for the dispatcher process. For example, `<dispatcher-name>.log`.
- The GWS proxy logs. For example, look for the `gwsproxy-<group>-app-name.log` started to serve each Web service.
- The VM logs. For example, look for the `vm-<group>-app-name-<number>.log` for each fglrun process used to serve the Web service.

**Web application logs:**

- The dispatcher log for the dispatcher process. For example, `<dispatcher-name>.log`.
- The UA proxy logs. For example, look for the `uaproxy-<session-id>.log` started for each Web application.
- The VM logs. For example, look for the `vm-<session-id>.log` for each fglrun process used to serve a Web application.

**Tip:** If the dispatcher is still running, you can troubleshoot the dispatcher and proxy logs using the monitor URL. Otherwise, locate the GAS logs in your `appdata/log/<dispatcher-name>/<date>` directory. See Log files on page 185.

**Important:** Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

1. Check the VM logs for standard errors or error messages.
   - It is recommended that you check your VM logs first because if there is an indication that the fglrun process crashes, then this will also lead to HTTP errors appearing in the proxy logs.
2. Check the proxy logs, Web services (gwsproxy) or Web applications (uaproxy), for errors.
   - If an error is reported in the proxy logs, it should have a corresponding entry in the dispatcher or VM logs for the process that was active at the time.
3. Check the dispatcher log file for HTTP request failures, etc.

   If, for example, you are troubleshooting HTTP request failures, search for HTTP codes other than 200. Codes in the 400 range indicate a request that failed given the information provided (for example, a required parameter was missing, etc.). Codes in the 500 range indicate an error with the server. For an example, see Troubleshoot error 503 on page 268.

4. If further assistance is required, contact your local Four Js support center. If requested to send the GAS logs, copy the logs for the date the issue arose to a zip file and email it to your support contact. Typically these logs are needed:
   
   • appdata/log/*.log.
   • Related dispatcher daily logs (dispatcher, proxy, and vm) for the date the issue arose.

**Related concepts**

*Get application session id* on page 123

When you want to find the DVM and proxy logs to troubleshoot an application, use this information to locate the session identifier representing the GAS session.

**What if the application doesn’t start?**

There are several reasons why an application does not start. Here are some basic troubleshooting procedures to follow as a standard approach to solving problems.

1. Check the application configuration (xcf) file - to ensure that all components are set properly.

2. 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.

3. 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 379 category filter to CONFIGURATION.

4. 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 188.

   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 Browser Client (GBC) applications, see Configure the GBC development environment on page 171.

**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.

**Related concepts**

*Troubleshooting HTTP errors* on page 121
Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

**Get application session id**
When you want to find the DVM and proxy logs to troubleshoot an application, use this information to locate the session identifier representing the GAS session.

**Session id**
The session id represents an identifier that uniquely identifies a GAS session for an application or Web service. For example, this long string "96c9ce0ded72135ddf43ad421a2d87b9" is the session id of an application running in the GAS. Knowing the session id for the application allows you to locate the relevant proxy and DVM logs for the session when troubleshooting or debugging. The different ways you can identify and locate the session id are discussed next.

**Identify a session ID in a log**
The example application URL shows the session id in a dispatcher log entry:

```
... "Incoming request" /ua/sua/7e26fadb0c9f6939c65702fc9a1ff2a4?
appId=0&pageId=1
```

The session id is the string, 7e26fadb0c9f6939c65702fc9a1ff2a4.

**Locate an application's session ID**
When in development mode, the default application ended page for a Genero Browser Client (GBC) Web application displays the session id. For more information, see the *Genero Browser Client User Guide*. You can also find the session id in the application URL of the monitor page.

**Session ID not available**
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.

**Related concepts**
- **Troubleshooting HTTP errors** on page 121
  Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

- **Log files** on page 185
  When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**Related information**
- List dispatcher sessions on page 354

**Rendering issues**
If an application does not render or display 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:

- Misalignment
- 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 the GBC development environment on page 171
2. Use browser developer tools to inspect elements of a GBC application interface to identify the widgets not rendering correctly. Browser developer tools are accessed by pressing the F12 key. Other things to check:
   - Check that the resource files (.css, .js, and so on) are accessible and not displaying a 404 error message.
   - Check that the HTML is properly generated. For example, check whether the color for the edit is set or is missing in the generated page.

**GBC customization**

Run your program without any GBC 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 GBC customization. For more information, see the Genero Browser Client User Guide.

**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 Genero Application Server 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.

**Related concepts**

SSO custom sample (simplesso) on page 165
The Four Js Genero GitHub repository contains a demo of how to delegate the start of a Genero application to another service, in order to handle the authentication via a REST service.

How delegation works
The delegation process redirects the start of an application or service to a delegation service to authenticate a user. To help you understand delegation, different scenarios for starting and/or refusing a service and the communication paths involved are illustrated.

Following steps are performed when a delegation occurs:

1. An application or web service start is requested. The type of request is defined by the /ua/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 specified in the application's configuration. The delegation service is written in Genero and managed by the GAS as a standard REST Web service. The delegation service should reside on the same Genero Application Server as the application.
3. The REST service instructs the GAS to:
   - Refuse the start of the application or service.
   - Allow the application to start. The delegate service is able to add some environment variables to give additional information to the allowed application.
   - Allow 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.

Figure 34: 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 /ua/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 of the application or service as if launched directly from the user agent.

Figure 35: REST service approves application or service start

In this figure:

- (1) Start a new application or service of the form /ua/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 the proxy (login and password are correct, for instance) to start.
- (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 (except 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.

Related concepts
Configure delegation for application or service on page 127
To delegate the start of an application or service to the Genero REST service, specify a `DELEGATE` element in the EXECUTION component of your application or service.

Configure delegation for application or service

To delegate the start of an application or service to the 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.

Delegate configuration example

```xml
<APPLICATION Parent="defaultwa"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextwa.xsd">
   <EXECUTION>
      <PATH>$(res.path)</PATH>
      <MODULE>myApp.42r</MODULE>
      <DELEGATE service="MyGroup/MyDelegateService">
         <anyparameter>MyFirstParameter</anyparameter>
         <other>MySecondParameter</other>
      </DELEGATE>
   </EXECUTION>
</APPLICATION>
```

Parameters defined in the `DELEGATE` configuration, such as `anyparameter` and `other` in this example, are transmitted to the REST service. See Pass parameters to the REST service on page 130 for details on the HTTP header syntax and receiving the parameters in the REST service.

**Important:** Some Web servers convert all HTTP header names to lowercase. For example, a parameter called `AnyParameter` in the configuration, in the header may become:

```
x-fourjs-environment-parameter-anyparameter
```

Therefore, it is not recommended to rely on case in the naming of parameters as there is no guarantee that it will be preserved. When working with HTTP headers in your Genero program code, make sure you allow for this by converting them to either upper or lower case.

Related concepts

From the user agent to the REST service on page 130
Each request of the form `/ua/r` and `/ws/r` coming from the user agent is delegated to the Genero REST service via its entry point.

`DELEGATE` on page 383
The `DELEGATE` element specifies the Genero REST service in charge of handling requests for access to applications.

Related tasks

Add OpenID Connect SSO to Web application on page 144
Add OpenID Connect SSO to a Genero Web application.

**Optimize delegation with HTTP options**

The delegation mechanism for a service can be optimized using HTTP OPTIONS.

The delegation mechanism can be configured to avoid sending the original request body destined for the final proxy to the delegate service, when it is not needed as part of an authorization request.

When configured, only the headers are sent, from which the delegate service determines whether the original request body can be sent to the final proxy. Use the `DELEGATE_OPTIONS` element for this configuration.

**Delegate optimization example**

When `DELEGATE_OPTIONS` with `verb="OPTIONS"` is set, the following original HTTP POST request is transformed as shown in the example requests:

```plaintext
POST /ws/r/my_service HTTP/1.1
Content-Type: text/plain
Authorization: Bearer Token_ID
User-Agent: GWS Agent
Date: Mon, 16 Jul 2018 14:39:43 GMT
Host: localhost:6363
Connection: close
Content-Length: 11
X-FourJs-Environment-Variable-REMOTE_ADDR: 127.0.0.1
X-FourJs-Environment-Variable-SERVER_NAME: localhost
Hello world
```

It is then for optimization purpose, transformed by the dispatcher as an HTTP OPTIONS request to the delegation service:

```plaintext
OPTIONS /ws/r/my_delegate/Delegate?url=http://localhost:6363/ws/r/my_service HTTP/1.1
Content-Type: text/plain
Authorization: Bearer Token_ID
User-Agent: GWS Agent
Date: Mon, 16 Jul 2018 14:39:43 GMT
Host: localhost:6363
Connection: close
Access-Control-Request-Method: POST
Content-Length: 0
X-FourJs-Environment-Variable-REMOTE_ADDR: 127.0.0.1
X-FourJs-Environment-Variable-SERVER_NAME: localhost
```

In this example, you can see:

1. It removes the body of the original request, ("Hello World").
2. It keeps the original headers (it sets the body length to zero, `Content-Length: 0`).
3. It adds a new HTTP header (`Access-Control-Request-Method: POST`) to provide the original HTTP request verb (for example `POST`) to the delegate service.

**Delegate service responses**

The delegate service may return one of the following responses to the optimized request, and the dispatcher takes appropriate action:

- If the delegate service agrees to transmit the request to the final proxy, it returns the HTTP code 307 response. The dispatcher then forwards the original request plus the message body to the final proxy.
In the case that the delegate service returns something other than HTTP code 307, the dispatcher will then discard the original request body and the original request will get the delegate response (plus body) as response (standard delegation mechanism).

**Example**

In this sample Web Service server code you see it is using OPTIONS to handle RESTful requests to optimize delegation.

```golang
FUNCTION DoDelegate(req)
    DEFINE req com.HttpServiceRequest
    DEFINE query WSHelper.WSQueryType
    DEFINE ori_url STRING
    DEFINE access_token STRING
    DEFINE ind INTEGER
    DEFINE method STRING

    # Ensure we work in optimized delegation via OPTIONS
    IF req.getMethod()!="OPTIONS" THEN
        CALL req.sendResponse(400,"bad delete request, HTTP OPTIONS verb required");
        RETURN
    END IF

    # Ensure we got the Access-Control-Request-Method header
    LET method = req.getRequestHeader(C_ACCESS_CONTROL_REQUEST_METHOD)
    IF method IS NULL THEN
        CALL req.sendResponse(400,"bad delete request, Access-Control-Request-Method is missing");
        RETURN
    END IF

    # Ensure we have a valid delegate request
    CALL req.getUrlQuery(query)
    IF query[1].name!="url" THEN
        CALL req.sendResponse(400,"bad delete request, no url found");
        RETURN
    END IF

    # Extract original URL
    LET ori_url = query[1].value
    CALL query.deleteElement(1)

    # Retrieve access token from query string
    LET ind = query.search("name","access_token")
    IF ind>0 THEN
        LET access_token = query[1].VALUE
    ELSE
        CALL req.sendResponse( 401, "No access token found")
        RETURN
    END IF

    # Check Access
    IF CheckAccess(req, baseURL, method, ori_url, access_token) THEN
        CALL req.sendResponse( 307, "GENERO_INTERNAL_DELEGATE")
    ELSE
        CALL req.sendResponse( 403, "Forbidden")
    END IF

END FUNCTION
```
From the user agent to the REST service

Each request of the form /ua/r and /ws/r coming from the user agent is delegated to the Genero REST service via its entry point.

REST service delegation entry point

When a /ua/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 is configured for delegation, the REST service is called with a /Delegate appended in the URL.

```
IMPORT com
DEFINE req com.HTTPServiceRequest
...
LET req = com.WebServiceEngine.GetHttpServiceRequest(-1)
LET path = req.getUrlPath() 
IF path.indexOf("/ws/r/RestGroup/RestService/Delegate",1)>1 THEN
  CALL HandleDelegation(req)
ELSE
  CALL HandleStandardService(req)
END IF
...
```

Pass parameters to the REST service

If parameters are defined in the DELEGATE configuration, they are transmitted to the Genero REST service at each /ua/r or /ws/r request in 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

```
IMPORT com
DEFINE req com.HTTPServiceRequest
...
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 the example application configuration in Configure delegation for application or service on page 127.

Pass the user agent URL to the REST service

When a request is delegated to a REST service, the original URL is transmitted to the service in the URL query string of the request with url as the key and the original URL as its value. It is properly encoded so that the com.HTTPServiceRequest.getURLQuery() method can decode the URL and all query parameters.

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:

```
```

**Process the delegation request in the delegation service**

To process the request in the delegation service, if the service handles only delegation, you can directly extract the original URL with the `getURLQuery()` or `readFormEncodedRequest()` method:

```
IMPORT com
IMPORT FGL WSHelper
...
DEFINE req com.HTTPServiceRequest
DEFINE original STRING
DEFINE url STRING
DEFINE query WSHelper.WSQueryType
DEFINE ind INTEGER
...
CALL req.getURLQuery(query)
IF query.getLength()==0 THEN
  CALL req.sendTextResponse(400,NULL,"Not a valid delegate request")
ELSE
  IF query[1].name == "url" THEN
    # retrieve original URL
    LET url = query[1].value
    # remove original URL from query array to keep only query of original URL
    CALL query.deleteElement(1)
  ELSE
    CALL req.sendTextResponse(400,NULL,"No url parameter found")
  END IF
END IF
DISPLAY url      # http://host:port/ua/r/MyGrp/MyApp
# Handle additional query parameters from the original URL
FOR ind = 1 TO query.getLength()
  DISPLAY "query"||ind
  DISPLAY "  name is ",query[ind].name
  DISPLAY "  value is ",query[ind].value
END FOR
...
```

**Process non-delegated and delegated requests**

If the service must also handle non-delegated requests, use the `getURLPath()` method to retrieve the REST operation to perform them.

If the REST path contains the "/Delegate" string, it is a delegation request and you need to extract the original URL as a query value of the query key named `url`, as described in Process the delegation request in the delegation service on page 131.

```
IMPORT com
IMPORT FGL WSHelper
...
DEFINE req com.HTTPServiceRequest
DEFINE path STRING
...
LET path = req.getURLPath()
CASE
  WHEN path.getIndexOf("/Delegate",1) #Perform delegate operation as above example
    ...
  WHEN path.getIndexOf("/GetCustomerInfo",1) # Handle regular REST request
    ...
```

CALL req.sendTextResponse(200, NULL, "Done...")
OTHERWISE
CALL req.sendTextResponse(400, NULL, "Invalid REST request")
END CASE

Related concepts
From the REST service to the proxy on page 132
The delegation REST service must notify the dispatcher when it approves the start of an application or service.

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
- 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.

```java
IMPORT com
DEFINE req com.HTTPServiceRequest
...
CALL req.sendResponse(307,"_GENERO_INTERNAL_DELEGATE_")
...
```

Note: You can return an HTTP message body from the REST service that is then transmitted to the proxy if the original incoming request was POST or PUT, otherwise body is skipped.

Passing parameters to Web applications or Web services
When you need to pass additional parameters or environment variables to a starting proxy, you do so through the HTTP header. The HTTP header name must be of the form:

```
X-FourJs-Environment-envvar:value
```

Where:
- `envvar` is the name of the variable.
- `value` is the value of the variable.

Each parameter must be set in the HTTP header response when you specify the 307 HTTP return code. For example:

```java
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_")
...
```

These headers are returned by the service to the proxy to be parsed and added in the environment of started DVMs.

Note: Case sensitivity of environment variables are handled differently in different operating systems. For example, in Windows® they are case insensitive, and in UNIX® they are case sensitive. In addition some Web servers convert
all header names to lowercase. In order to provide for these constraints, the proxy converts all environment variable
names to uppercase. For example, the proxy may receive the header in the form:

- X-FourJs-Environment-Hello:World
- x-fourjs-environment-hello:World

It converts the environment variable to uppercase and sets it in the DVM environment as HELLO=World where it
can be retrieved.

Retrieving parameters for Web application

The parameters set in the Passing parameters to Web applications or Web services on page 132 are converted to
environment variables and thus you can retrieve them in your Genero application with a call to the Genero function
fgl_getenv(VARIABLENAME) (the variable name must be in uppercase)

Genero program sample:

```genero
MAIN
  ...
  DISPLAY fgl_getenv("HELLO") -- Displays "World"
  DISPLAY fgl_getenv("NAME") -- Displays "Georges"
  ...
```

Retrieving parameters for Web services

In the context of the Web service, the parameters set in the Passing parameters to Web applications or Web services
on page 132 can be retrieved in your delegation service or in another Web service with a call to the Genero
getRequestHeader method of the com.HTTPServiceRequest class.

Genero program sample:

```genero
IMPORT com
DEFINE resp com.HTTPServiceRequest

LET h1 = resp.getRequestHeader("X-FourJs-Environment-Hello")
DISPLAY h1 -- Displays "World"

LET h2 = resp.getRequestHeader("X-FourJs-Environment-Name")
DISPLAY h2 -- Displays "Georges"

...
```

Related concepts

Configure delegation for application or service on page 127
To delegate the start of an application or service to the Genero REST service, specify a DELEGATE element in the
EXECUTION component of your application or service.

Application environment on page 44
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.

REST service example on page 134
In this example, the REST service returns an HTTP error code 404 to the browser until the query string contains the
string ByPass.

Delegation use cases on page 136
Three examples of possible uses for the delegation mechanism.

**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 a Genero Web client application is started, the parameter **ACCESS** is also set and can be retrieved in the Genero program with `fgl.getenv("ACCESS")`.

```gas
IMPORT COM
IMPORT XML
IMPORT FGL WSHelper

PRIVATE CONSTANT C_BASEURL = "/ws/r/qa-test/MyService/"
PRIVATE CONSTANT C_X_FOURJS_ENVIRONMENT_ = "X-FourJs-Environment="
PRIVATE CONSTANT C_GENERO_INTERNAL_DELEGATE = "_GENERO_INTERNAL_DELEGATE_"

MAIN
DEFINE  req    com.HttpServiceRequest
DEFINE  methd STRING
DEFINE  url    STRING
DEFINE  path   STRING
DEFINE  ind    INTEGER
DEFINE  operation 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()
    LET methd = req.getMethod()
    LET path = req.getUrlPath()
    DISPLAY "Incoming request: ",methd," path=",path
    LET ind = path.getIndexOf(C_BASEURL,1)
    IF ind<1 THEN
      CALL req.sendResponse(400,"Invalid request")
    ELSE
      # Dispatch request based on operation name (ex : /ws/r/qa-test/MyService/operation1)
      LET operation = path.subString(ind+C_BASEURL.getLength(),path.getLength())
      CALL DispatchService(req,operation)
    END IF
    DISPLAY "Response sent"
  END IF
CATCH
  EXIT WHILE
END TRY
END WHILE
END MAIN

FUNCTION DispatchService(req,operation)
DEFINE  req    com.HttpServiceRequest
DEFINE  operation STRING
DEFINE  ind    INTEGER
LET ind = operation.getIndexof("/",1)
IF ind>0 THEN
  CALL req.sendResponse(400,"invalid operation")
END IF
```

| GAS configuration | 134 |
ELSE
  CASE operation
    WHEN "Delegate" # Handle a dispatcher delegate service
      CALL DelegateWA(req)
    OTHERWISE
      CALL req.sendResponse(400,"unknown service '"||operation||"'.");
  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)
  DEFINE req com.HttpServiceRequest
  DEFINE q WSHelper.WSQueryType
  DEFINE url STRING
  DEFINE ind INTEGER
  DEFINE ByPass BOOLEAN

  # Decode Delegate query
  CALL req.getURLQuery(q)
  IF q.getLength()==0 THEN
    CALL req.sendResponse(400,"no query string")
    RETURN
  ELSE
    IF q[1].name=="url" THEN
      LET url = q[1].value
      CALL q.deleteElement(1)
      # Valid delegate request with original URL as key parameter
    ELSE
      CALL req.sendResponse(400,"no valid delegate request")
      RETURN
    END IF
  END IF
  # Process Delegate request
  LET ByPass = FALSE
  # Check if user-agent query string has a ByPass string ?
  FOR ind=1 TO q.getLength()
    IF q[ind].name=="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 the web client application via environment variable:
    ACCESS=OK
    CALL req.setResponseHeader(C_X_FOURJS_ENVIRONMENT_||"ACCESS","OK")
    # Start application with HTTP code 307
    CALL req.sendResponse(307,C_GENERO_INTERNAL_DELEGATE)
  END IF
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.

A sample local authentication / authorization implementation of Single Sign On (SSO) for Genero, with support for re-login after a period of inactivity, can be found under the Four Js Genero GitHub repository. See https://github.com/FourjsGenero/ex_simpleso.

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 user name and password will be sent in clear text.

A request with user name and password as parameters is processed by the delegation service. The service checks 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.

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.

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 is processed by the delegation service. The cookie is 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.

Related concepts

How to implement Single sign-on (SSO) on page 139

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.

SSO custom sample (simplesso) on page 165
The Four Js Genero GitHub repository contains a demo of how to delegate the start of a Genero application to another service, in order to handle the authentication via a REST service.

**How to develop delegation for Genero apps**

Outlines the requirements for developing your own delegation service for applications running on Genero clients.

Developing a delegation service involves you creating the program that provides a secure process for allowing users access to your applications. The code contained in the service needs to take into account the different Genero clients that may be used to provide the applications. It will also be influenced by the different methods these clients use to launch applications.

Typically, the same code can be used by most of the clients. For the purpose of simplifying what you need to code, we can group the clients into Genero Browser Client (GBC) and "Non-GBC". The non-GBC clients consists of:

- Genero Desktop Client (GDC)
- Genero Mobile for Android™
- Genero Mobile for iOS

**Delegation for GBC and non-GBC clients**

When the user agent launches an application, your delegation service receives a request for access to the application. Typically, you code to perform authentication or a log-in process. An HTTP code is returned to the dispatcher; allowing access or not.

For the non-GBC applications this is performed in the same request. For the GBC type application, two requests are needed to open the application:

1. The first request (shown in the "GBC first call" workflow in the diagram ) needs to perform the authentication and fetch the GBC bootstrap page. This page gathers information about the browser and loads the GBC.
2. The second request (the default for non-GBC clients) starts the actual Genero BDL application.
The diagram represents the process of starting an application with delegation for both the GBC and non-GBC. The workflow is illustrated at a high level showing the components involved.

**GBC application bootstrap**

The GBC first call request is shown on the right hand side of the diagram, following the sequence to load the bootstrap.

At step 1 prior to the GBC bootstrap, a call must be made first by the /ua/r/app to request to load the bootstrap.

The dispatcher process (step A in the diagram) calls your delegate service with an additional HTTP header called X-FourJs-Environment-Parameter-Extra-BOOTSTRAP:REQUIRED. This means that you have not yet returned the bootstrap page.

If your delegation service allows access to the application (step 2), you return the HTTP code 307 (step 3) and the bootstrap page is then loaded by the user-agent, which automatically re-sends the same /ua/r/app request to launch the application.
Non-GBC and GBC applications

Having the bootstrap for the GBC, the request by the `/ua/r/app` (shown on the workflow on the left) performs the actual launch of the application.

Once the application has started, then all subsequent requests to the application are in `/ua/sua` form and they no longer go through the delegation service.

Coding for delegation

In your delegate service you implement all that is required for the service. You perform your authentication, or the log in tasks you want users to perform (step 2 in the diagram). For instance, you may decide to return an HTML form for the user to enter credentials.

For the GBC type application because two requests are performed, it is recommended that you code to set a cookie with the session id on the first call, and store this information (usually in a database).

When the second request comes (for the GBC type application only, step B) you can check if the session id stored on the first call corresponds with the session id in the second request. This ensures that the first request has been successful and the application launch can go ahead.

On the second request also the presence of an additional HTTP header called `X-FourJs-Environment-Parameter-Extra-BOOTSTRAP` means that the bootstrap page has been loaded.

At step 3 you must also code to return all the `X-FourJs-Environment-*` variables that the uaproxy needs to start the application.

Note: There is no need to set them at the bootstrap request stage because the proxy is not started at this stage.

For an example of coding a simple delegation service, see the "ex_simplesso" service on the Four Js Genero GitHub page.

Related concepts

How delegation works on page 125

The delegation process redirects the start of an application or service to a delegation service to authenticate a user. To help you understand delegation, different scenarios for starting and/or refusing a service and the communication paths involved are illustrated.

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.

Identity Provider (IdP)

An IdP provides a secure identity information service for authenticating users accessing your applications and Web services. Using an IdP is recommended when implementing a single sign-on (SSO) solution.

Identity providers rely on specifications such as OpenID-Connect, OAuth, and Security Assertion Markup Language (SAML) to grant access to Web applications on behalf of an authenticated user, using access tokens of various kinds, but without providing the user's credentials to the application.

Genero comes with a ready-to-use Genero IdP, or you can use a third party IdP.

Providing authentication using the Genero IdP

Providing authentication using the Genero IdP is covered in these topics:

- Genero Identity Provider (GIP) on page 204
- Setting up the Genero Identity Provider on page 215
- Configure for an external GIP on page 220
Providing authentication using a third-party IdP

To provide authentication for SSO using a third-party IdP:

1. Set up an account with a trusted IdP provider (such as Google, see Configure OpenID Connect identity on Google on page 143.)
   As a third party registered on the authorization IdP server, you are issued with tokens (public and shared secret ids).
2. Use the tokens provided and add delegation to your application configuration files requiring authentication.
   This allows you to provide access to the protected resources hosted by your Web application. See Add OpenID Connect SSO to Web application on page 144 Add SAML SSO to a Genero Web application on page 155.

Genero supports the delegation of authentication services to the following types of identity providers:

- OpenID Connect (OIDC) identity provider. See Add OpenID Connect SSO to Web application on page 144.
- SAML identity provider. See Add SAML SSO to a Genero Web application on page 155.

A Genero delegate service is delivered in $FGLDIR/web_utilities/services for SAML and OpenID Connect. These manage all delegated requests for applications or services run on the GAS.

How it works

Since the authentication process is relayed to an IdP provider, user access to your application or Web service is redirected to the IdP. Depending on the type of IdP provider, an HTTP accept/reject response is returned in either a signed XML assertion document (SAML) or a signed JSON document (OpenID Connect) via the Genero delegate service.

If access is allowed, the delegate service then extracts from the returned document some user data before starting the application on the GAS. Once an application has been started after user authentication by the IdP, environment variables are provided defining the user role and access. The environment can be retrieved in your Genero application via fgl_getenv() instructions. See Retrieve the OpenID Connect user identifier on page 145 and Retrieve identity attributes with SAML on page 158.

Related concepts

Single sign-on workflow on page 141
Before an application can start, the SSO delegate process passes through various stages of communication with the identity provider (IdP) to authenticate the user.

Related tasks

Quick start: Set up OpenID Connect in the GAS on page 142
Follow these steps to quickly set up OpenID Connect for your Genero Application Server and Genero Web applications.

Quick start: Set up SAML in the GAS on page 152
Follow these steps to quickly set up SAML for your Genero Application Server and Genero Web applications.

**Single sign-on workflow**

Before an application can start, the SSO delegate process passes through various stages of communication with the identity provider (IdP) to authenticate the user.

**Figure 37: SSO workflow to start the application**

The diagram represents the process of starting an application performing SSO with the GAS. The workflow is illustrated at a high level showing the components involved. The communication paths that are numbered (1 to 7) are explained as follows:

1. User agent requests the start of an application of the form `/ua/r`.
2. The SSO delegate service redirects the user agent to the IdP defined in the configuration file (`xcf`).
3. The IdP queries the user agent directly for user login and password.
   
   **Note:** Step 3 may not happen if the IdP recognizes an user already registered in a previous SSO login session.
4. If the login is ok, depending on the protocol, the IdP may create SSO tokens in its local database and then redirect the user agent to the SSO delegate service `callback` URL (previously registered).
5. On the incoming `callback` URL, the delegate service requests the token directly from the IdP and checks its signature validity via the IdP's public key.
6. If the token is valid, the delegate service sets a temporary cookie for the initial `/ua/r` application URL and redirects the user agent to that URL.
7. If the temporary cookie is valid, the `ua/r` application is started on behalf of the authenticated user.

**Note:** Once the application has started, it may receive an access token to query some REST services that the authenticated users has access to.

**Related concepts**

Identity Provider (IdP) on page 139
An IdP provides a secure identity information service for authenticating users accessing your applications and Web services. Using an IdP is recommended when implementing a single sign-on (SSO) solution.

**OpenID Connect SSO**

OpenID Connect is a Single sign-on (SSO) protocol supported by the Genero Application Server.

A delegation Web service for SSO OpenID Connect based on the REST service is provided in the FGLGWS package. It is delivered in the package under `FGLDIR/web_utilities/services/openid-connect/`. The solution is supported on the GAS delegation mechanism.

OpenID Connect implementation creates a circle of trust between the Genero Application Server and an OpenID Connect provider. OpenID Connect providers include Google and Microsoft. To learn more about OpenID Connect, see the [OpenID Connect web site](#).

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 142)
- Providing the redirect URL of the GAS to the IdP, (see [Add OpenID Connect SSO to Web application](#) on page 144)

**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 applications.

Before you begin, you must [Configure GAS for OpenID Connect SSO](#) on page 143.

In this quick start, you add OpenID Connect Single sign-on (SSO) to a Genero Browser 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 applications requiring SSO

   The first three parameters are mandatory:
   - IDP: the provider of the IdP account (for example, 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 (for example, 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>
   <CLIENT_PUBLIC_ID>XXXXXXXX.apps.googleusercontent.com</CLIENT_PUBLIC_ID>
   <CLIENT_SECRET_ID>XXXXXX-XXXXXX</CLIENT_SECRET_ID>
   </DELEGATE>
   </EXECUTION>
   </APPLICATION>
   ```

2. **Execute a Genero Browser 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 **signin** 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 Browser 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
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, for example, see Configure OpenID Connect identity on Google on page 143.
2. If the GAS 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 GAS is ready to use OpenID Connect SSO to authenticate end users.

### Related concepts
- Genero OpenID Connect FGLPROFILE on page 148
- Genero OpenID Connect implementation uses its own FGLPROFILE file.

### 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/
2. Create a new project (or use an existing one)
3. From the project page select Credentials.
4. In the Credentials page, select OAuth client ID from the Create credentials drop-down menu.

   This opens the Create OAuth client ID page where you select a product type and create a client identity. For more information and help creating the client id, refer to the OpenID Connect page in the Google documentation.

   a) Choose Web Application as product
   b) In the Authorized JavaScript origins field, specify your JavaScript hostname (for example, https://host:port/gas)
   c) In the Authorized redirect URIs field, specify the URI redirection where the GAS is listening for the response (i.e. https://host:port/gas/ws/r/services/OpenIDConnectServiceProvider/oauth2callback)
   d) Click Create.

   The OAuth2 Client and Client Secret IDs are displayed.

   **Note:** You will need to save these in your Web service application configuration file. See Add OpenID Connect SSO to Web application on page 144.

You have now set up Google as your IdP for your Web services to use OpenID Connect SSO.
Add OpenID Connect SSO to Web application
Add OpenID Connect SSO to a Genero Web application.

This task must be performed in the .xcf application configuration file.

Add `<DELEGATE service="services/OpenIDConnectServiceProvider">` to the application configuration (.xcf) file.

Add the `DELEGATE` tag to all Genero Browser Client applications requiring Single sign-on (SSO), plus the 3 mandatory parameters:

- **IDP**: the IdP account (for example, `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 (for example, email, phone, address)

```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/3.20/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>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

With the above configuration and default GAS configuration, the delegation points to the delegation REST Web service in the `$FGLDIR`.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 124.

The Genero Application Server will handle the OpenID Connect protocol and start the Web application only when the user has been authenticated, otherwise an HTML error page is returned.

Related concepts
**DELEGATE** on page 383
The `DELEGATE` element specifies the Genero REST service in charge of handling requests for access to applications.

**Delegation use cases** on page 136
Three examples of possible uses for the delegation mechanism.

Related tasks
**Retrieve the OpenID Connect user identifier** on page 145
Follow these steps to retrieve the OpenID Connect Single sign-on (SSO) user identifier in your Genero application.

**Authorization and OpenID Connect SSO** on page 145
Authorize whether a user already authenticated by OpenID Connect SSO can access a Genero application.

**Configure OAuth redirect with automatic form submit**

Configure the OpenID Connect service to perform an automatic form submit redirect at authentication request.

By default, the OpenID Connect service redirects the user-agent with an HTTP 302 response. In some cases the user agent or IdP requires the authentication redirect method to be performed via an HTML submit form.

You do this configuration in the configuration file located in `$FGLDIR/web_utilities/services/openid-connect/res`.

Locate the entry `oidc.authenticate.redirect` in the configuration file.

Specify the required value, allowed values are:

a. `oidc.authenticate.redirect = "302"`: (default) perform an HTTP redirection (recommended way)

b. `oidc.authenticate.redirect = "GET"`: an automatic HTML form submit with the GET method is performed by the user agent webview to send the authentication request.

c. `oidc.authenticate.redirect = "POST"`: an automatic HTML form submit with the POST method is performed by the user agent webview to send the authentication request.

**Related concepts**

Identity Provider (IdP) on page 139

An IdP provides a secure identity information service for authenticating users accessing your applications and Web services. Using an IdP is recommended when implementing a single sign-on (SSO) solution.

**Related tasks**

Add OpenID Connect SSO to Web application on page 144

Add OpenID Connect SSO to a Genero Web application.

**Retrieve the OpenID Connect user identifier**

Follow these steps to retrieve the OpenID Connect Single sign-on (SSO) user identifier in 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_` and in uppercase. (OIDC stands for OpenID Connect).

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 142), you will have an attribute called `OIDC_EMAIL` set that is then retrievable with the instruction in your application.

To retrieve the user identifier, add this code to your Genero application:

```plaintext
LET userEmail = fgl_getenv("OIDC_EMAIL")
```

**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.

**Related concepts**

Connect to the application database with SSO on page 166

There are several solutions for automatically connecting to the database server, after starting an application program with a Single sign-on (SSO) delegation.

**Authorization and OpenID Connect SSO**

Authorize whether a user already authenticated by OpenID Connect SSO can access a Genero application.

The Genero Application Server must be configured for OpenID Connect Single sign-on (SSO). See Configure GAS for OpenID Connect SSO on page 143.

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:

```plaintext
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:

```bash
fglrun AccessProgram  
"10151604318349889392"  
"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/3.20/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>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 latter case, the end user will be warned with an error page in the web browser, generated by the OpenID Connect service.
**Related concepts**

**DELEGATE** on page 383

The **DELEGATE** element specifies the Genero REST service in charge of handling requests for access to applications.

---

**Authorize re-log in with OpenID Connect SSO**

Use this procedure to configure re-log in to a Genero application authenticated by OpenID Connect SSO after an auto logout event.

**Note:** The FGLGWS package provides a delegation Web service for SSO OpenID Connect that supports the **PROMPT (for auto logout)** on page 412 feature. In the example the **PROMPT** is set to use this delegation service:

```xml
<PROMPT Timeout="60" Type="DELEGATE">services/OpenIDConnectServiceProvider</PROMPT>
```

1. **Add a DELEGATE element in your application configuration (xcf) file.**

   This example shows the application configuration for delegation and the auto logout prompt feature.

   **Note:** Within the **DELEGATE** element, the **GOOGLE_OPENID_PUBLIC_ID** and **GOOGLE_OPENID_SECRET_ID** are values got when registering your GAS on Google developer console. For more information see Quick start: Set up OpenID Connect in the GAS on page 142.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <APPLICATION Parent="defaultgwc" >
   <EXECUTION>
   <PATH>$ (res.deployment.path) </PATH>
   <MODULE>MyApp.42r</MODULE>
   <DELEGATE service="services/OpenIDConnectServiceProvider">
   <IDP>https://accounts.google.com</IDP>
   <SCOPE>email</SCOPE>
   <CLIENT_PUBLIC_ID>GOOGLE_OPENID_PUBLIC_ID</CLIENT_PUBLIC_ID>
   <CLIENT_SECRET_ID>GOOGLE_OPENID_SECRET_ID</CLIENT_SECRET_ID>
   </DELEGATE>
   </EXECUTION>
   </APPLICATION>
   <AUTO_LOGOUT>
   <TIMEOUT>10</TIMEOUT>
   <PROMPT Timeout="60" Type="DELEGATE">services/OpenIDConnectServiceProvider</PROMPT>
   </AUTO_LOGOUT>
   </APPLICATION>
   ```

2. **Add a PROMPT element in the AUTO_LOGOUT element**

   The delegation service represents the GAS’s SSO OpenID Connect Service, which the user-agent will be redirected to when the user wants to re-log in. The **Timeout** represents the number of seconds the user-agent displays a screen or page to notify the user that a re-log in is required if he wants to continue.

   Once the user is authenticated by the service, the user-agent is redirected back to the GAS to resume the application.

---

**Related concepts**

**How autologout prompt is implemented on SSO** on page 165

The prompt feature can authenticate the user and resume the application after an auto-logout event.

**Configure OpenID Connect SSO log out**

Configure log out from the OpenID Connect Single Sign on (SSO) authentication server after an application ends.

---

**In General**

The Genero delegate service, OpenIDConnectServiceProvider, automatically redirects the user agent to the OpenID Connect end session URL (if specified). Normally, you do not need to configure this logout if you have an OpenID delegate service set for your application.
As Identity providers generally have not yet implemented the logout session management protocol, then a configuration is required. You use the SSO tags, `IDP_LOGOUT_URL` and `SIGN_OFF`, in the `DELEGATE` element to specify the logout.

**Note:** If you do not specify SSO log out options in the application configuration, on closing the application the user will **not** be logged out from the IdP. This is the equivalent of specifying `<SIGN_OFF>FALSE</SIGN_OFF>` in the `DELEGATE` element.

**Troubleshooting**

Google does not follow the OpenID-Connect protocol fully. If your IdP provider is Google, for example, the following configuration shows how to implement the logout.

The delegation SSO log-out feature is provided in FGLGWS and GAS version 3.20.

1. Add a `DELEGATE` element in your application configuration (`xcf`) file.

   **Note:** Within the `DELEGATE` element, the `GOOGLE_OPENID_PUBLIC_ID` and `GOOGLE_OPENID_SECRET_ID` are values got when registering your GAS on Google developer console. For more information see Quick start: Set up OpenID Connect in the GAS on page 142.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <APPLICATION Parent="defaultgwc" >
      <EXECUTION>
         <PATH>$(res.deployment.path)="/PATH>
         <MODULE>MyApp.42r</MODULE>
         <DELEGATE service="services/OpenIDConnectServiceProvider">
            <IDP>https://accounts.google.com</IDP>
            <SCOPE>email</SCOPE>
            <CLIENT_PUBLIC_ID>Google_OpenID_Public_ID</CLIENT_PUBLIC_ID>
            <CLIENT_SECRET_ID>Google_OpenID_Secret_ID</CLIENT_SECRET_ID>
            <SIGN_OFF>QUERY</SIGN_OFF>
         </DELEGATE>
      </EXECUTION>
      <AUTO_LOGOUT>
         <END_URL>http://www.4js.com</END_URL>
      </AUTO_LOGOUT>
   </APPLICATION>
   ```

2. Add an `IDP_LOGOUT_URL` parameter in the `DELEGATE` element

   The URL represents the log out page of the Google IdP authentication server. This URL requirement varies depending on the IdP used.

3. Add a `SIGN_OFF` parameter.

   This parameter allows for three possible log out methods from the IdP authentication server:
   - **TRUE.** Closing the application will perform SSO log out. Restarting the application, the user is prompted for SSO login.
   - **FALSE.** Closing the application will not perform SSO log out. Restarting the application, the user is not prompted for SSO login.
   - **QUERY.** (default value) Closing the application, the user is prompted with the options to log out of the SSO or not.

   **Note:** If the user elects to log out, the IdP needs to redirect back to the user agent too, so it is recommended to also specify an application `END_URL`.

**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 GAS 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:

```xml
<Application Parent="ws.default"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextwa.xsd">
    <Execution>
        <!-- ENVIRONMENT_VARIABLE entries removed for this example -->
        <Path>$(res.path.fgldir.services)/openid-connect/bin</Path>
        <Module>OpenIDConnectServer -logfile "$(res.appdata.path)" -debug DEBUG</Module>
    </Execution>
</Application>
```

**Note:** Logging is based on the Genero ERRORLOG() function. As several instances of the same OpenID Connect server can write to the log file, the PID of the server process is written to the log file as well.

OpenID Connect support for OAuth2

OpenID Connect service provides support for simple OAuth2 authentication.

OAuth2 is used by identity provider (IdP) providers such as Facebook and Instagram. As OAuth2 does not have a metadata feature that enables the GAS to automatically find the required URL endpoints based on the URL, the OpenID Connect service provides a tool called ImportOAuth to register these.

**The ImportOAuth program**

Use the ImportOAuth program to register the mandatory URL endpoints based on the OAuth2 identity provider (IdP) URL.

With the Genero ImportOAuth program, you can:

- Register a new OAuth2 identity provider (IdP)
- List all registered OAuth2 IdPs
- Remove OAuth2 IdP identified by its URI
The ImportOAuth.4gl source code is provided in $FGLDIR/web_utilities/services/openid-connect/src/, and the compiled version is in the bin directory.

Syntax

```
ImportOAuth [ options ] [ IdP ]
```

1. **options** are described in Table 10: ImportOAuth options on page 150.
2. **IdP** is the URL of an OAuth2 identity provider.

**Note:** The ImportOAuth tool command line follows the convention of other Genero command line tools for both short and long versions of options.

### Table 10: ImportOAuth options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Usage example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Display help with the command.</td>
<td>fgldr ImportOAuth -h</td>
</tr>
<tr>
<td>-l, --list</td>
<td>List all imported IdPs stored in the database.</td>
<td>fgldr ImportOAuth -l</td>
</tr>
<tr>
<td>-r, --remove</td>
<td>Remove the OAuth2 entry for the specified IdP from the database.</td>
<td>fgldr ImportOAuth -r <a href="https://www.instagram.com">https://www.instagram.com</a></td>
</tr>
<tr>
<td>-i, --import</td>
<td>Import the IdP endpoints and register them in the database for OAuth2.</td>
<td>See Import option usage example on page 150</td>
</tr>
<tr>
<td>-a, --authz arg</td>
<td>OAuth2 authorization end point URL (mandatory).</td>
<td>See Import option usage example on page 150</td>
</tr>
<tr>
<td>-t, --token arg</td>
<td>OAuth2 token end point URL (mandatory).</td>
<td>See Import option usage example on page 150</td>
</tr>
<tr>
<td>-p, --profile arg</td>
<td>OAuth2 user profile end point URL (optional).</td>
<td>See Import option usage example on page 150</td>
</tr>
<tr>
<td>-o, --logout arg</td>
<td>OAuth2 logout end point URL (optional).</td>
<td>See Import option usage example on page 150</td>
</tr>
</tbody>
</table>

**Import option usage example**

To import an IdP as OAuth2, you must execute the ImportOAuth program with the --import option. In the following examples line breaks are added for readability. For Instagram the command is:

```
fgldr ImportOAuth --import
   --authz https://api.instagram.com/oauth/authorize
   --token https://api.instagram.com/oauth/access_token
   --logout https://instagram.com/accounts/logout
   --profile https://api.instagram.com/v1/users/self?
   https://www.instagram.com
```

The first two parameters --authz and --token are mandatory, and the endpoint URLs of the IdP are required:

1. **--authz URL**: mandatory authorization endpoint URL
2. **--token URL**: mandatory token endpoint URL
3. **--logout URL**: optional logout endpoint URL
4. **--profile URL**: optional user profile endpoint URL
5. **URL of the IdP**: mandatory

**Important:** The profile URL requires an access_token to get user profile information. In some cases, the access_token must be provided via the query string (for example, Instagram requires it). Therefore you must add
an ending question mark (?) during importation so that OpenID-Connect service providers know how to generate that request.

Related tasks

How to integrate Facebook SSO in the GAS on page 151
Configuration for an application you want to get authenticated via facebook.

How to integrate Instagram SSO in the GAS on page 151
Configuration for an application you want to get authenticated via Instagram.

How to integrate Facebook SSO in the GAS
Configuration for an application you want to get authenticated via facebook.

About this task:

Facebook SSO requires OAuth authentication. In the DELEGATE element of your application configuration specify the OAUTH parameter.

1. Run the ImportOAuth program to import the Facebook endpoints.

   Note: This is a once-off operation to import the endpoints. Once the endpoints are registered in the database, you can continue to configure your applications for Facebook. For more information, see The ImportOAuth program on page 149.

   In the following example line breaks are added for readability.

   ```
   fglrun ImportOAuth --import
   --authz https://www.facebook.com/v3.0/dialog/oauth
   --token https://graph.facebook.com/v3.0/oauth/access_token
   --logout https://www.facebook.com/logout.php
   --profile https://graph.facebook.com/me
   https://www.facebook.com
   ```

   In the application configuration file add an OAUTH parameter in the DELEGATE element.

   ```
   <APPLICATION>
   <EXECUTION>
   <PATH>c:\user</PATH>
   <MODULE>facebook.42r</MODULE>
   <DELEGATE service="services/OpenIDConnectServiceProvider">
   <OAUTH>https://www.facebook.com</OAUTH>
   <CLIENT_PUBLIC_ID>PUBLIC_ID registered on facebook developer console</CLIENT_PUBLIC_ID>
   <CLIENT_SECRET_ID>SECRET_ID registered on facebook developer console</CLIENT_SECRET_ID>
   </DELEGATE>
   </EXECUTION>
   </APPLICATION>
   ```

   - The OAUTH parameter specifies the Facebook URL.
   - The CLIENT_PUBLIC_ID and CLIENT_SECRET_ID are tokens obtained when registering your GAS for Facebook Single Sign-On.

How to integrate Instagram SSO in the GAS
Configuration for an application you want to get authenticated via Instagram.

About this task:

Instagram SSO requires OAuth authentication. In the DELEGATE element of your application configuration specify the OAUTH parameter.

1. Run the ImportOAuth program to import the Instagram endpoints.
**Note:** This is a once-off operation to import the endpoints. Once the endpoints are registered in the database, you can continue to configure your applications for Instagram. For more information, see The ImportOAuth program on page 149.

In the following example line breaks are added for readability.

```
fglrun ImportOAuth --import
   --authz https://api.instagram.com/oauth/authorize
   --token https://api.instagram.com/oauth/access_token
   --logout https://instagram.com/accounts/logout
   --profile https://api.instagram.com/v1/users/self?
       https://www.instagram.com
```

2. In the application configuration file add an `OAUTH` parameter in the `DELEGATE` element.

```
<APPLICATION>
  <EXECUTION>
    <PATH>c:\user</PATH>
    <MODULE>instagram.42r</MODULE>
    <DELEGATE service="services/OpenIDConnectServiceProvider">
      <OAUTH>https://www.instagram.com</OAUTH>
      <SCOPE>depends on the IdP</SCOPE>
      <CLIENT_PUBLIC_ID>PUBLIC_ID registered on Instagram developer console</CLIENT_PUBLIC_ID>
      <CLIENT_SECRET_ID>SECRET_ID registered on Instagram developer console</CLIENT_SECRET_ID>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

- The `OAUTH` parameter specifies the Instagram URL.
- The `CLIENT_PUBLIC_ID` and `CLIENT_SECRET_ID` are tokens obtained when registering your GAS for Instagram Single Sign-On.

**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 saml.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**

Follow these steps to quickly set up SAML for your Genero Application Server and Genero Web applications.

In this quick start, you add SAML Single sign-on (SSO) to a Genero Web 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 file, 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 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 application:

a) Add the `DELEGATE` tag to all Genero Web 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:

3. Execute a Genero Web 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 Web 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 GAS, a circle of trust must be established between the service providers (the Genero Application Server) 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 163.

1. If the GAS 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 GAS and a SAML provider. Import the IdP metadata file into the GAS 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 GAS.
   a) Start the dispatcher (if needed).
   b) Log in to your SAML provider and create a circle of trust based on the GAS 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 GAS is ready to support SAML SSO.

**Related concepts**

Genero SAML FGLPROFILE on page 164
Genero SAML Single sign-on (SSO) implementation uses its own FGLPROFILE file.

**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) in the GAS 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 --import option and the IdP's 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**

```sh
gflrun ImportIdP [options] [url | file]
```

1. `options` are described in Table 11: ImportIdP options on page 155.
2. `url` is the URL of a SAML identity provider.
3. `file` is the metadata file of a SAML identity provider.

**Note:** The ImportIdP tool command line follows the convention of other Genero command line tools for both short and long versions of options.

**Table 11: ImportIdP options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Usage example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Display help with the command</td>
<td><code>fglrun ImportIdP -h</code></td>
</tr>
<tr>
<td><code>-i, --import</code></td>
<td>Import the IdP specified by the URL or metadata file and register it in the database for SAML.</td>
<td>See Import option usage example on page 155.</td>
</tr>
<tr>
<td><code>-l, --list</code></td>
<td>List all registered IdPs in the database.</td>
<td><code>fglrun ImportIdP -l</code></td>
</tr>
<tr>
<td><code>-r, --remove</code></td>
<td>Remove the registered entry for the specified IdP from the database.</td>
<td><code>fglrun ImportIdP -r http://host:port/saml/jsp/myIdPMetadata.jsp</code></td>
</tr>
<tr>
<td><code>-N, --NameIDFormat</code></td>
<td>List IdPs in the database that support a specified name identifier (NameID Format).</td>
<td><code>fglrun ImportIdP -N</code></td>
</tr>
</tbody>
</table>

**Import option usage example**

To register a SAML IdP, you must execute the ImportIdP program with the --import option. The command is:

```sh
fglrun ImportIdP --import http://host:port/saml/jsp/myIdPMetadata.jsp
```

Where the IdP metadata file or URL is a mandatory parameter.

**Related tasks**

- Configure GAS to support SAML SSO on page 154
- Follow these steps to setup Genero SAML service.

**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
<APPLICATION version="1.0">
  <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 points to the delegation REST Web service in the `$FGLDIR`.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 124.

The GAS 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.

**Related concepts**

**DELEGATE** on page 383

The `DELEGATE` element specifies the Genero REST service in charge of handling requests for access to applications.

**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
<APPLICATION version="1.0">
  <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 uses the specified IdP as its Single sign-on (SSO) identity provider.

**Related concepts**

**The ImportIdP program** on page 154

Use the `ImportIdP program` to register a SAML identity provider.

**DELEGATE** on page 383
The **DELEGATE** element specifies the Genero REST service in charge of handling requests for access to applications.

**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 the 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 Genero Application Server 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 GAS, it defines what piece of data is sent from the IdP to the GAS to represent the user.

To define the ID format you want to receive from your IdP, perform the following steps.

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 GAS 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/3.20/cfextwa.xsd">
<EXECUTION>
  <PATH>$(res.path.qa)/applications/myapp</PATH>
  <MODULE>App.42r</MODULE>
  <DELEGATE service="services/SAMLServiceProvider">
    <IDFORMAT>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</IDFORMAT>
  </DELEGATE>
</EXECUTION>
</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.

**Related concepts**

The **ImportIdP program** on page 154

Use the ImportIdP program to register a SAML identity provider.

**DELEGATE** on page 383

The **DELEGATE** element specifies the Genero REST service in charge of handling requests for access to applications.

**Configure delegation for application or service** on page 127

To delegate the start of an application or service to the Genero REST service, specify a **DELEGATE** element in the **EXECUTION** component of your application or service.

**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:

```javascript
LET id = fgl_getenv("SAML_ID")
```
Set the authentication context

At the GAS 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 GAS 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"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/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 GAS will return an access denied page.

**Related concepts**

**DELEGATE** on page 383
The DELEGATE element specifies the Genero REST service in charge of handling requests for access to applications.

**Configure delegation for application or service** on page 127
To delegate the start of an application or service to the Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

**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_.

```bash
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 154.

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):

```
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:

```
fglrun AccessProgram
"AZEd3R4" \
"qa-test/application" \
"fullname" "genero test" \
"email" "genero@4js.com" \
"country" "France"
```

The application AccessProgram.4gl in $FGLDIR/web_utilities/services/saml 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 version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/ 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.

**Related concepts**

DELEGATE on page 383

The **DELEGATE** element specifies the Genero REST service in charge of handling requests for access to applications.

Configure delegation for application or service on page 127

To delegate the start of an application or service to the Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

**Authorize re-log in with SAML SSO**

Use this procedure to configure re-log in to a Genero application authenticated by SAML SSO after an auto logout event.

**Note:** The FGLGWS package provides you with a delegation Web service for SAML Single sign-on (SSO) that supports the PROMPT (for auto logout) on page 412 feature. In the example the PROMPT is set to use this delegation service:

```xml
<PROMPT Timeout="60" Type="DELEGATE">services/SAMLServiceProvider</PROMPT>
```

1. Add a DELEGATE element in your application configuration (xcf) file.

   This example shows the application configuration for delegation and the auto logout prompt feature.

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <APPLICATION Parent="defaultgwc">
     <EXECUTION AllowUnsafeSession="TRUE">
       <PATH>$(res.deployment.path)</PATH>
       <MODULE>MyApp</MODULE>
       <DELEGATE service="services/SAMLServiceProvider" />
     </EXECUTION>
     <AUTO_LOGOUT>
       <TIMEOUT>10</TIMEOUT>
       <PROMPT Timeout="60" Type="DELEGATE">services/SAMLServiceProvider</PROMPT>
     </AUTO_LOGOUT>
   </APPLICATION>
   ```

2. Add a PROMPT element in the AUTO_LOGOUT element

   The delegation service represents the GAS’s SAML SSO Service, which the user-agent will be redirected to when the user wants to re-log in. The Timeout represents the number of seconds the user-agent displays a screen or page to notify the user that a re-log in is required if he wants to continue.

   Once the user is authenticated by the service, the user-agent is redirected back to the GAS to resume the application.

**Related concepts**

How autologout prompt is implemented on SSO on page 165

The prompt feature can authenticate the user and resume the application after an auto-logout event.

**Execute an application with SAML SSO**

Follow these steps to execute a Web application and authenticate the 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 **signin** button.

   If the credentials are valid, your browser redirects to your Genero Web 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 Web application will start and authenticate with the same SAML user.

**Configure SAML SSO log out**
Configure log out from the SAML SSO authentication server after an application ends.

The Genero delegate service, SAMLServiceProvider, automatically queries the user when closing the Web application if he wants to log out from the IDP or not. You can change this behavior by setting the SSO tag called SIGN_OFF in the DELEGATE element of your application configuration.

**Note:** If you do not specify SSO log out options in the application configuration, on closing the application the user will not be logged out from the IdP. This is the equivalent of specifying `<SIGN_OFF>FALSE</SIGN_OFF>` in the DELEGATE element.

The delegation SSO log-out feature is provided in FGLGWS and GAS version 3.20.

1. Add a `DELEGATE` element in your application configuration (xcf) file.

   ```
   <APPLICATION Parent="defaultgwc">
     <EXECUTION>
       <PATH>$(res.path.mypath)/myapplication</PATH>
       <MODULE>myapp.42r</MODULE>
       <DELEGATE service="services/SAMLServiceProvider">
         <IDP>https://idp.ssocircle.com</IDP>
         <IDFORMAT>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</IDFORMAT>
         <SIGN_OFF>TRUE</SIGN_OFF>
       </DELEGATE>
     </EXECUTION>
   </APPLICATION>
   ```

2. Add a `SIGN_OFF` parameter.

   This parameter allows for three possible log out methods from the IdP authentication server:
   - **TRUE.** Closing the application will perform SSO log out. Restarting the application, the user is prompted for SSO login.
   - **FALSE.** Closing the application will not perform SSO log out. Restarting the application, the user is not prompted for SSO login.
   - **QUERY.** (default value) Closing the application, the user is prompted with the options to log out of the SSO or not.

   **Note:** If the user elects to log out, the IdP needs to redirect back to the user agent too, so it is recommended to also specify an application `END_URL`.

   For more information on the other delegation parameters:
   - For IDP, see Select the SAML server (Identity Provider) on page 156.
   - For IDFORMAT, see Define the SAML ID format on page 157.

**Genero SAML configuration**
Specify entries in the FGLPROFILE file to configure the Genero SAML service provider.

The Genero SAML implementation provides a list of FGLPROFILE file entries to configure the Genero SAML service provider. The configuration file is located in `$FGLDIR/web_utilities/services/saml/res`. 
Table 12: SAML-related FGLPROFILE entries

<table>
<thead>
<tr>
<th>FGLPROFILE file 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 GAS is represented to other SAML partners. Mandatory. Default is urn:genero.</td>
</tr>
<tr>
<td>saml.allowUnsecure</td>
<td>Defines whether the GAS accepts unsecured authentication mechanisms. Default is false (recommended). A SAML authentication mechanism is unsecured if communication between the Identity Provider (IdP) and the GAS is not performed either over HTTPS or with XML encryption. To secure a SAML communication, use HTTPS (via ISAPI or FastCGI) or use XML-Encryption by setting the xml.saml_encryption entries as described in Assertion encryption on page 162.</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 GAS returns an access denied HTML page. This entry also adds the wantAssertionsSigned attribute to the SAML metadata describing the SAML needs of the GAS.</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 GAS 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 GAS 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`.

1. Create a new or use an existing database, essentially on a dedicated machine. If several Genero Application Server 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 the FGLPROFILE file 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 allow the Genero Application Server modification of 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:**  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.

Related concepts

Genero SAML FGLPROFILE on page 164
Genero SAML Single sign-on (SSO) implementation uses its own FGLPROFILE file.

Connect to the application database with SSO on page 166
There are several solutions for automatically connecting to the database server, after starting an application program with a Single sign-on (SSO) delegation.

**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 GAS installation requires proxy configuration to connect to a 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/3.20/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>
     <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.
SSO custom sample (simplesso)

The Four Js Genero GitHub repository contains a demo of how to delegate the start of a Genero application to another service, in order to handle the authentication via a REST service.

If you are looking for source files, and examples, you are encouraged to download from https://github.com/FourjsGenero/ex_simplesso the sample implementation of Single Sign On (SSO) that has been created for you. This example can be adapted for your own application needs.

It provides a sample implementation of Single Sign On (SSO) for Genero where the end user is required to enter a login and password. Once connected, the end user does not need to enter any credentials for 24 hours. There is also support for re-login after a time of inactivity.

As a demo, the session management is simple and sessions are managed via a SQLite database.

**Important:** On production sites, the security must be improved. Please consider the recommendations mentioned in the readme file in the download.

**Related concepts**

- How autologout prompt is implemented on SSO on page 165

The prompt feature can authenticate the user and resume the application after an auto-log-out event.

**How autologout prompt is implemented on SSO**

The prompt feature can authenticate the user and resume the application after an auto-log-out event.

If the auto_logout PROMPT element is configured, when an AUTO_LOGOUT timeout is reached, the user-agent displays a screen or page to notify the user that a re-login is required if the user wants to continue. If the user clicks on the re-login button, the user agent is redirected to a URL specified by the PROMPT configuration element.

**The URL query string**

The uaproxy adds a query string to the URL specified by the PROMPT that may contain the following:

- Application session id. (session=ua_session_id in the URL examples)
- Timeout value. (timeout=prompt_timeout in the URL examples)
- An entry from the FGL_AUTO_LOGOUT_PROMPT_QUERY environment variable, if set at uaproxy start up.

Examples of the prompt and query strings are shown for both URL and DELEGATE types of re-login:

**URL re-login**

The URL type represents an absolute URL, that the user-agent is redirected to when the user wants to re-log in.

**Warning:** When using PROMPT of Type URL, no security check is done to ensure that the request to resume the application comes from a valid user. Use prompt of type URL only for prototyping purposes.

```
http://myserver/reauthenticate?
session=ua_session_id&timeout=prompt_timeout&pump=prompt_id
```

**DELEGATE re-login**

The DELEGATE type represents the delegate service the user-agent is redirected to when the user wants to re-log in.

**Note:** The delegate URL is directed to the Genero Application Server SSO OpenID Connect Service provided in the FGLGWS package. The Web services URL is concatenated with Prompt, which provides this re-login service.

```
http://localhost:6394/gas/ws/r/services/OpenIDConnectServiceProvider/Prompt?
session=ua_session_id&timeout=prompt_timeout&pump=prompt_id
```
Depending on the URL, you must handle the authentication of the user based on the query string provided. Once the user is authenticated, you must redirect the user agent back to the application on the GAS with a URL of the form:

```
http://host:port/gas/ua/resume/session_id
```

### Application restart via URL

The uaproy will remove the screen hiding the application, and the user resumes the application.

### Application restart via DELEGATE

When an application stops, a new cookie named Genero-END is set by the GAS. If you need to detect at application restart how the application previously stopped, the cookie value will indicate this as follows:

- **Disconnected**: indicates the application stopped due to the auto logout timeout.
- **Closed**: indicates the application was stopped by user-action.

The cookie allows you to choose (via your delegate application start mechanism, for instance) if you want to force new credentials query (in case of disconnection) or not (in case of normal close).

**Note**: Genero-END is not generated on first starting the application. As the cookie is not persistent, if you close the browser, the cookie is discarded.

Handling the application restart involves the following:

- The dispatcher forwards the `/ua/resume` request (with all cookies) to the delegate service via the delegate mechanism in order to validate user tokens that may have been set during the credential validation process.
- If the user credentials are valid, the delegate service responds with the HTTP code 307 and an HTTP response phrase of `_GENERO_INTERNAL_DELEGATE_` in order to resume the `ua` application. See [How to implement delegation](#) for more details.
- If the user credentials are not valid, the delegate service returns an HTTP response that is displayed in the user-agent window to indicate the errors.

### Related concepts

**PROMPT (for auto logout)** on page 412

The PROMPT element provides a feature that allows a user to resume an application after an auto logout event.

### Related tasks

**Authorize re-log in with OpenID Connect SSO** on page 147

Use this procedure to configure re-log in to a Genero application authenticated by OpenID Connect SSO after an auto logout event.

**Authorize re-log in with SAML SSO** on page 160

Use this procedure to configure re-log in to a Genero application authenticated by SAML SSO after an auto logout event.

### 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
MAIN
    DEFINE uname, upswd VARCHAR(50)
    ...
    CONNECT TO "dbsource" USER uname USING upswd
    ...
END MAIN
```

For more details about the `CONNECT TO` instruction, see *Database Connections* chapter in the *Genero Business Development Language User 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 that a password is 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®:

```
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 without requiring the database administrators to create database users. End users enter an application login and password, that is checked and stored by the application in a dedicated table of the database.

In this solution, an SSO technology is used where end users can create credentials from an open identity provider (IdP), as, for example, with the OpenID Connect SSO on page 142 implementation. Since anyone can freely register with the Web application, there is no application administrator task required to create an application user.

To access application data, a set of predefined database users with a fixed name and password must exist in the database. The database users are hidden from the application or end users. Each predefined database user is 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:** As application programs 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 to start by the GAS SSO mechanism, it needs to connect to the database, and therefore the program must get application user information (login name, password, and type of user), from the GAS SSO procedure.

For example, when using OpenID Connect, the Web application must get the end user login, password and user type through the corresponding environment variables:

```
LET user_name = FGL_GETENV("OIDC_user_name")
LET user_pswd = FGL_GETENV("OIDC_user_pswd")
LET user_type = FGL_GETENV("OIDC_user_type")
```

For an example using SAML, see [Retrieve identity attributes with SAML](#) on page 158.

The user type identifies the predefined database user that is 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 for the user type got from the SSO attributes. For example, the program can get the database user name and password from an 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:

```sql
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
    SELECT last_login INTO ts FROM app_users
        WHERE app_users.u_name = user_name
    IF SQLCA.SQLCODE == 100 THEN
        -- Application user is not registered
        ...
    ELSE
        -- Password is invalid
        ...
    END IF
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 a 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.

**Related concepts**

- [OpenID Connect SSO](#) on page 142
  - OpenID Connect is a Single sign-on (SSO) protocol supported by the Genero Application Server.

- [SAML SSO](#) on page 152
  - 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`.

**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, such as 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 Log files on page 185) 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 config -z` command, see gasadmin tool.

### Enabling and disabling GAS compression

Both the `gasadmin config -z` compression command and the GAS use the `imt.cfg` file to identify the resources that can be compressed.

```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 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
```

**Related concepts**
gasadmin tool on page 348
The gasadmin tool is provided as an administrative command for the Genero Application Server.

Configure the GBC development environment

Configure a development environment to troubleshoot an Genero Browser Client (GBC) 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 by performing the following steps.

**Warning:** The **--development** option forces the GBC into debug mode. This may effect the behavior of the GBC in the browser. For example, the prompt that is usually displayed when the user presses the browser's back button, is no longer displayed:

Are you sure you want to leave this page?

1. In the GAS configuration file (default \$FGLASDIR/\etc/as.xcf), change the res.uaproxy.param resource:

   **From:**
   ```xml
   <RESOURCE Id="res.uaproxy.param" Source="INTERNAL"></RESOURCE>
   ```

   **To:**
   ```xml
   <RESOURCE Id="res.uaproxy.param" Source="INTERNAL">--development</RESOURCE>
   ```

   **Tip:** 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. For more information, see the **Genero Browser Client User Guide**.

**Related concepts**

GAS configuration file (as.xcf) on page 360

The Genero Application Server is configured through a configuration file. The default configuration file is as.xcf, located in the \$FGLASDIR/\etc directory.

Configure multiple dispatchers

If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

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.

Each dispatcher will have its own uniquely-named configuration file, and each configuration file must have different values for the elements shown in **Table 13: Elements defined for each dispatcher** on page 172:
Table 13: Elements defined for each dispatcher

<table>
<thead>
<tr>
<th>Element</th>
<th>Values to be set</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_BASE_PORT on page 429</td>
<td>The base port.</td>
</tr>
<tr>
<td>TCP_PORT_OFFSET on page 430</td>
<td>The offset that is added to the base port as the listening port. The combination of base port+offset must be unique to each dispatcher.</td>
</tr>
<tr>
<td>TCP_ADMIN_PORT on page 429</td>
<td>The admin port for GAS administration tasks must be unique to each dispatcher.</td>
</tr>
<tr>
<td>Application data directory</td>
<td>Path to the application data directory for data files. The recommended way is to change the value of the res.appdata.path resource to a directory value unique for each dispatcher. An alternative approach is described in Shared application data directory on page 172.</td>
</tr>
</tbody>
</table>

Tip: You can share the same configuration file provided you run the dispatcher with the –E option to override the resources shown in Table 14: Override resources on page 172 for each instance with different values:

Table 14: Override resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>res.ic.base.port</td>
<td>httpdispatch -E res.ic.base.port=dispatcher1-base-port</td>
</tr>
<tr>
<td>res.ic.port.offset</td>
<td>httpdispatch -E res.ic.port.offset=myOffset</td>
</tr>
<tr>
<td>res.ic.admin.port</td>
<td>httpdispatch -E res.ic.admin.port=dispatcher1-admin-port</td>
</tr>
<tr>
<td>res.appdata.path</td>
<td>httpdispatch -E res.appdata.path=/work/tmp/gas/appdata1</td>
</tr>
</tbody>
</table>

Shared application data directory

If you want to use the same application data directory, specified by the resource res.appdata.path, for each dispatcher, it is important to ensure that directories located under the directory remain independent.

For more information on why you need to be cautious about this, see Why do my applications fail using multiple dispatchers? on page 107 and Error taking session ownership on page 64.

The values for the following directories need to be unique for each dispatcher. You can change these directly in their configuration files:

- LOG on page 403 directory (res.log.output.path)
- TEMPORARY_DIRECTORY (res.path.tmp)
- SESSION_DIRECTORY (res.appdata.path)/session

These examples are extracted from the default application server configuration file:

```
-- The res.appdata.path resource specifies the directory for application data
-- for the dispatcher
```
Related concepts

GAS configuration file (as.xcf) on page 360
The Genero Application Server is configured through a configuration file. The default configuration file is as.xcf, located in the $FGLASDIR/etc directory.

Dispatcher on page 38
Understand what the dispatcher does and identify which dispatcher to use with a specific Web server.

Configuring Genero Report Engine on GAS

Understand the different options available for configuring the GAS when a Genero Report Engine is operating in distributed mode locally or on a remote server.

Configure for GRE in distributed mode on local machine

When a Genero Report Engine (GRE) is operating in distributed mode 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 Report Viewer for HTML5 and resources.

- ${connector.uri}/ua/report/viewer
- ${connector.uri}/ua/report/public
- ${connector.uri}/ua/report/private/${session.id}

Table 15: 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/dispenser_name/$(session.id)/reports.

<table>
<thead>
<tr>
<th>URL prefix</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>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.</td>
</tr>
<tr>
<td>private</td>
<td>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/dispenser_name/$(session.id)/reports.</td>
</tr>
</tbody>
</table>

The GAS provides several environment variables to the started application. These environment variables are used with reports for the GRE operating in distributed mode locally.

**Table 16: Environment variables dedicated to GRE in distributed mode locally**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRE_PRIVATE_URL_PREFIX</td>
<td>Private URL 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_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>

**Related concepts**

*Configure GRE in distributed mode on remote machine* on page 174

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.

**Configure GRE in distributed mode on remote machine**

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.

In the `INTERFACE_TO_CONNECTOR` on page 400 configuration element in the GAS configuration file (as.xcf), specify a `REPORT_REMOTE_URL_PREFIX` on page 416 element and set the URL of the GRE:

```
<REPORT_REMOTE_URL_PREFIX>http://remotehost:12345</REPORT_REMOTE_URL_PREFIX>
```

The GAS provides three URLs with the report-r prefix to locate the Genero Report Viewer for HTML5 and resources.

**Note:** The URL prefix used is report-r as distinct from report used in GRE in distributed mode locally. 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 $(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/
   ebf19aba4d09e80cd19cfe47868a9a2a/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/
   ebf19aba4d09e80cd19cfe47868a9a2a/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/ebf19aba4d09e80cd19cfe47868a9a2a
```

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 17: 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_PRIVATE_PREFIX</td>
<td>Prefix private URL prefix</td>
<td><a href="http://localhost:6394/ua/report-r/private/89be6b3867148e738fc6bd1daad1ff2e">http://localhost:6394/ua/report-r/private/89be6b3867148e738fc6bd1daad1ff2e</a></td>
</tr>
<tr>
<td>GRE_PRIVATE_R_DIR_Private directory</td>
<td>Private resource directory.</td>
<td>private/89be6b3867148e738fc6bd1daad1ff2e</td>
</tr>
<tr>
<td>GRE_PUBLIC_R_DIR_Private URL prefix</td>
<td>Private URL prefix</td>
<td>public</td>
</tr>
</tbody>
</table>

Related concepts

Configure for GRE in distributed mode on local machine on page 173

When a Genero Report Engine (GRE) is operating in distributed mode on the GAS locally, the GAS is configured with URLs and environment variables to handle the processing of reports.

Administering the GAS

Understand the options available for the administration of the Genero Application Server.

Monitoring

Use the /monitor URL to view information on the current status of GAS dispatcher and on active applications.

Important: The monitor page is not accessible and needs to be configured. As the monitor page provides access to dispatcher and proxy / VM log files, which may contain some sensitive information, access to this page must be restricted via the MONITOR element. For an example of this configuration see Configure monitor access.

Related concepts

URI Examples on page 48
Several URI examples with ways to help you launch applications.

MONITOR on page 405
The MONITOR element specifies the hosts allowed to access the monitor page of the GAS.

Usage

Enter a URL to access the GAS monitor.

Accessing the monitor page

Important: The monitor page is not accessible and needs to be configured. As the monitor page provides access to dispatcher and proxy / VM log files, which may contain some sensitive information, access to this page must be
restricted via the MONITOR on page 405 element. For an example of this configuration see Configure monitor access.

Open the monitor page from the URL http://appserver:port/monitor.

For example, to open the monitor page on your local machine, enter this URL in your browser: http://localhost:6394/monitor

If you are using a Web server, you enter in a URL http://appserver:port/<gas>/monitor. The <gas> is the name of a directory or virtual directory on the Web Server.

Related concepts
Application Web Address on page 48
To access an application, you specify the necessary information in the browser's address bar by entering in the appropriate application URI.

Monitor URL
Statistics computed by the Genero Application Server (GAS) dispatcher are fed into the monitor URL page. Use this page to monitor and troubleshoot GAS operations.

Viewing information about the GAS
When you perform a monitor request, the dispatcher gathers information from all the proxies it has launched such as: number of DVM started by a proxy, the process IDs (PIDs) of each DVM, date and time of the last request, etc. This information is then put together and displayed in the browser.

Note: The information displayed represents the state of the dispatcher and proxies at the time of the request. This may change over time if you do the same request one second later for instance.

Tip: Data is displayed in XML format. You can modify the way statistics are presented by customizing the monitor.xslt file located in $FGLASDIR/web/fjs.

Use this list to check the current status and operations of the GAS:

• If, for example, you are troubleshooting HTTP request failures, use the Dispatcher logs file link found at the top of the page to open a page where logs are displayed.
• If you need statistics about the CPU usage of the dispatcher, go to the Dispatcher resources usage section of the page. See Dispatcher resources usage on page 179.
• If you need to manage Genero Archives and Genero Browser Clients deployed on the GAS, use the Deployment portal link to access a tool for this purpose.
• Your GAS may have invalid configuration files for Web services. You can check for this by using a link to the Invalid configuration files tool in the General information section.
• If you are monitoring or troubleshooting Web services, go to the Monitoring for GWS proxies section of the page. See Monitoring for GWS proxies on page 178.
• If you are monitoring or troubleshooting Web applications, go to the Monitoring for UA proxies section of the page. See Monitoring for UA proxies on page 179.
• If you are monitoring statistics by HTTP type requests, go to the Monitoring for HTTP requests section of the page. See Monitoring for HTTP requests on page 179.
Figure 38: Monitoring: GAS

General information

Use this section to check details about the dispatcher in use:

- Dispatcher name (for example, httpdispatch, isapidispatch, etc.)
- Date and time launched
- Version
- Build date

A link to Invalid configuration files provides you with access to a tool to check for invalid configuration files for Web services in the GAS.

Monitoring for GWS proxies

If you are monitoring Web services running on the GAS, the number of active Web services (GWS) sessions is shown. If sessions are active they are listed by session
id, and within that by application ids and number of connections. If you need to troubleshoot or view log details, you can access the logs by clicking on links:

- **Detailed monitoring**... opens a Genero Session Monitoring page where you can monitor information about the session, such as the proxy, the environment, the DVM, etc.
- **Proxy logging**... opens a page showing the gwsproxy log, which contains a chronology of activity since the proxy started.

If sessions are not active, you see the following message. No links are shown:

**Number of active sessions:** 0

### Monitoring for UA proxies

Use this section to get information about the number of active Web application sessions.

If sessions are active, a list of the session ids are shown. You can see what applications (application id) are running and how many connections are used in a session. If you need further details, you can access the relevant logs by clicking on these links:

- **Detailed monitoring**... opens a Genero Session Monitoring page where you can monitor information about the session: the proxy, the environment, the DVM, etc.
- **Proxy logging**... opens a page showing the uaproxy logs containing a chronology of activity since the proxy started.

If sessions are not active, you see the following message. No links are shown:

**Number of active sessions:** 0

### Dispatcher resources usage

Monitoring the dispatcher resource usage provides you with the following details:

- Process id
- Resources used since startup
- Resources used since last check
- Number of running threads:
- Process environment variables

### Monitoring for HTTP requests

If you need to know, for example, how many requests for Web services are being handled by the GAS, monitoring HTTP requests for */ws/r* in this section, provides you with the following information:

- Handled requests
- In-progress requests
- Successful requests
- Average time
- Last request time
Use this section to monitor the HTTP requests for Web applications, Web services, and monitoring.

**Related concepts**

- **Log files** on page 185
  When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

- **Troubleshooting application Issues** on page 121
  Troubleshooting tips are provided for the most common issues encountered.

**Related tasks**

- **Monitor dispatcher logs** on page 180
  Follow this procedure to monitor dispatcher logs in the GAS.

- **Monitor proxy logs** on page 181
  Follow this procedure to monitor proxies running in the GAS.

- **Monitor proxies and DVM** on page 182
  Follow this procedure if you need to troubleshoot or get a more detailed view of the current status and operations of the proxy and the DVM for a selected session.

- **Monitor HTTP requests** on page 184
  Information on the different types of HTTP requests performed by the GAS running Web applications, and Web services, may provide you with useful information for analysis and troubleshooting. Follow this procedure to monitor HTTP requests.

- **Find invalid Web service configuration files** on page 180
  Follow this procedure to check for invalid configuration files for Web services in the GAS.

**Find invalid Web service configuration files**

Follow this procedure to check for invalid configuration files for Web services in the GAS.

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 message informing an end user of the error. This tool provides a means of troubleshooting and monitoring Web services.

1. Open the monitor page.
   
   For example, on your machine enter this URL in your browser: `http://localhost:6394/monitor`.

2. Select the link **Invalid configuration files**
   
   The page (`http://localhost:6394/monitor/configuration/service/invalid`) opens in a browser tab.

   A list of invalid configuration files (if any) are displayed. If there are no invalid services, the page displays this message:

   **There are no invalid configuration files.**

**Related concepts**

- **Monitor URL** on page 177
  Statistics computed by the Genero Application Server (GAS) dispatcher are fed into the monitor URL page. Use this page to monitor and troubleshoot GAS operations.

- **Web service invalidation** on page 268
  The Web service invalidation feature identifies an invalid Web service configuration and informs the user agent with an HTTP error and message. Once identified, the invalid configuration must be corrected; in some instances, the Genero Application Server will need to be restarted.

**Monitor dispatcher logs**

Follow this procedure to monitor dispatcher logs in the GAS.

While the dispatcher is running, you can access the current logs to troubleshoot and/or monitor the GAS.
Note: You can also access dispatcher log files in your appdata/log/<dispatcher-name>/date directory.

Important: Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

1. Open the monitor page.
   For example, on your machine enter this URL in your browser: http://localhost:6394/monitor.

2. Select the link Dispatcher logs file
   The page (http://localhost:6394/monitor/log/<dispatcher-name>) opens in a browser tab.
   A list of HTTP requests sent and received since the dispatcher started is displayed.

3. If, for example, you are troubleshooting HTTP request failures, search for HTTP code 400 errors where normally code 200 appears.
   To further investigate, view the proxy logs for application running at the time when the error occurred.

Related concepts
Monitor URL on page 177
Statistics computed by the Genero Application Server (GAS) dispatcher are fed into the monitor URL page. Use this page to monitor and troubleshoot GAS operations.

Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

Related tasks
Monitor proxy logs on page 181
Follow this procedure to monitor proxies running in the GAS.

Monitor proxy logs
Follow this procedure to monitor proxies running in the GAS.

While the dispatcher is running, you can view the current proxy activity to troubleshoot and/or monitor the GAS.

Note: You can also access proxy log files in your appdata/log/<dispatcher-name>/date directory.

Important: Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

1. Open the monitor page.
   For example, on your machine enter this URL in your browser: http://localhost:6394/monitor.

2. Select the Proxy logging... link depending on whether you are monitoring GWS or UA proxies.
   The page (http://localhost:6394/monitor/log/<proxy-name>-<session-id>) opens in a browser tab. A chronology of activity since the proxy started, is displayed.

3. If you are troubleshooting, search for error messages.
   If, for example, the error identified is a DVM error, the process ID is given in the error message. To further investigate, view the VM log running at the time when the error occurred.

Related concepts
Monitor URL on page 177
Statistics computed by the Genero Application Server (GAS) dispatcher are fed into the monitor URL page. Use this page to monitor and troubleshoot GAS operations.

Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**Monitor proxies and DVM**

Follow this procedure if you need to troubleshoot or get a more detailed view of the current status and operations of the proxy and the DVM for a selected session.

In addition to viewing summary statistics about the dispatcher and proxies in the monitor page, you may also need to view more detailed statistics about a session.

This level of detail is provided by the Detailed monitoring... tool. You will find links to this in the Monitoring for UA proxies and Monitoring for GWS proxies sections.

1. Open the monitor page.
   
   For example, on your machine enter this URL in your browser: http://localhost:6394/monitor.

2. Select the Detailed monitoring... link for the proxy and session you wish to monitor.

   The page (http://localhost:6394/monitor/session_id) opens in a browser tab. Details are shown for the proxy, the environment, and the applications running in the session.

   Use Table 18: Proxy Details on page 183 as a reference to help you with analysis of the session data.

![Figure 39: Viewing session statistics](image-url)
### Table 18: Proxy Details

<table>
<thead>
<tr>
<th>Proxy details</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatcher</td>
<td>Names the proxy (for example, uaproxy, gwsproxy)</td>
</tr>
<tr>
<td>Version</td>
<td>Gives the proxy version number.</td>
</tr>
<tr>
<td>Build date</td>
<td>Shows the proxy build date.</td>
</tr>
<tr>
<td>Application</td>
<td>Shows the name of the application running</td>
</tr>
<tr>
<td>User agent</td>
<td>Gives details of user agent, for example, browser type and version.</td>
</tr>
<tr>
<td>Remote IP</td>
<td>Shows the IP address of the server.</td>
</tr>
<tr>
<td>Remote User</td>
<td>Gives the name of the user on the server.</td>
</tr>
<tr>
<td>Proxy port</td>
<td>Shows the port number proxy is listening on.</td>
</tr>
<tr>
<td>FGLSERVER port</td>
<td>Shows the port number FGLSERVER is listening on.</td>
</tr>
<tr>
<td>Launched at</td>
<td>Shows the date proxy was launched.</td>
</tr>
<tr>
<td>Command line</td>
<td>Gives the path to the DVM.</td>
</tr>
<tr>
<td>Process id</td>
<td>Shows the DVM process ID.</td>
</tr>
<tr>
<td>Proxy Resources usage</td>
<td>The resources used by the proxy process (in seconds) since start-up and last check is given. Details are grouped by usage in:</td>
</tr>
<tr>
<td></td>
<td>• Real time (total time process took)</td>
</tr>
<tr>
<td></td>
<td>• User time (CPU time used in executing the process)</td>
</tr>
<tr>
<td></td>
<td>• System time (CPU time spent in system calls to the kernel used in executing the process)</td>
</tr>
<tr>
<td>Process environment variables</td>
<td>Provides a list of the environment variables set on the system.</td>
</tr>
<tr>
<td>DVM (list of sessions identified by IDs)</td>
<td>Details of applications (identified by ID) running in the session are given. These are grouped by:</td>
</tr>
<tr>
<td></td>
<td>• Start url (URL used to start application)</td>
</tr>
<tr>
<td></td>
<td>• Command line (path to the DVM)</td>
</tr>
<tr>
<td></td>
<td>• Connection string: meta Connection (to include language encoding, runtime version, frontEndID, procId, etc.)</td>
</tr>
<tr>
<td></td>
<td>• Launch date</td>
</tr>
<tr>
<td></td>
<td>• Last /ua/sua date (date and time of last submitted user action request)</td>
</tr>
<tr>
<td></td>
<td>• Last /ua/sua url (URL of last submitted user action request)</td>
</tr>
</tbody>
</table>

### Related concepts

Monitor URL on page 177
Statistics computed by the Genero Application Server (GAS) dispatcher are fed into the monitor URL page. Use this page to monitor and troubleshoot GAS operations.

Monitor HTTP requests

Information on the different types of HTTP requests performed by the GAS running Web applications, and Web services, may provide you with useful information for analysis and troubleshooting. Follow this procedure to monitor HTTP requests.

The Monitoring for http requests section of the monitor page provides the following statistics per request type:

- Handled requests
- In-progress requests
- Successful requests
- Average time
- Last request time

1. Open the monitor page.
   For example, on your machine enter this URL in your browser: http://localhost:6394/monitor.

2. Scroll to the Monitoring for http requests section.

   If you are monitoring Web applications, view the ua type requests. Use Table 19: Web applications (ua) Request Types on page 184 as a reference to help you with analysis of the data.

   **Table 19: Web applications (ua) Request Types**

<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ua/r</td>
<td>User requests to start applications.</td>
</tr>
<tr>
<td>/ua/sua</td>
<td>Front-end requests submitting user actions.</td>
</tr>
<tr>
<td>/ua/close</td>
<td>Front-end requests to stop the DVM.</td>
</tr>
<tr>
<td>/ua/ft</td>
<td>File transfer requests</td>
</tr>
<tr>
<td>/ua/i</td>
<td>Image requests</td>
</tr>
<tr>
<td>/ua/newtask</td>
<td>Front-end request for pending applications in the DVM not yet notified to the front-end.</td>
</tr>
<tr>
<td>/ua/interrupt</td>
<td>Interrupt requests sent by the front-end to the DVM.</td>
</tr>
<tr>
<td>/ua/wait, /ua/ping</td>
<td>Timeout/keep-alive requests sent by the front-end.</td>
</tr>
<tr>
<td>/ua/w</td>
<td>Requests to access files in the Genero Browser Client directory.</td>
</tr>
<tr>
<td>/ua/report</td>
<td>Requests to a Genero Report Engine (GRE) running locally.</td>
</tr>
<tr>
<td>/ua/report-r</td>
<td>Requests to a GRE running on a remote server.</td>
</tr>
</tbody>
</table>

If you are monitoring Web services, monitoring statistics for /ws/r HTTP requests will provide this information.

If you are monitoring applications launched via Genero Desktop Client, viewing statistics for the /da/r HTTP request will provide this information.

If you require statistics about monitoring, viewing the following request types will give you details:

- /monitor
- /monitor/log
• /monitor/configuration

If requests of "unknown" type are received, this information is summarized under the **unknown** heading.

**Related concepts**

**Monitor URL** on page 177
Statistics computed by the Genero Application Server (GAS) dispatcher are fed into the monitor URL page. Use this page to monitor and troubleshoot GAS operations.

**Troubleshooting HTTP errors** on page 121
Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

**Log files**

When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**Important:** Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

The following log files are created:

- **Dispatcher**
  A log file is generated for each dispatcher. Incoming requests, starting of proxies, responses sent, and system error messages are captured.

- **Proxy**
  A separate log file is generated for each proxy started.

- **DVM**
  A log file is generated for each DVM started. DVM standard error and standard output are sent to the dedicated DVM log files.

**Accessing log files in appdata**

You can access log files in the `appdata/log/dispatcher_name/date` directory, where `appdata` refers to GAS application data directory. To learn about the `appdata` directory (to include where to find it), see **GAS installation and application data directories** on page 40.

In a Windows® platform, for example, log files are found in the `C:\ProgramData\vendor\gas\gas_version\log\dispatcher_name\current_date` directory.

**Accessing log files from monitor URL**

While the dispatcher is running, access the logs from the GAS monitor URL `http://appserver:port/monitor`. Use the links found in the monitor page to view current dispatcher and proxy logs.

If in development mode, use the link (Get the VM logs) in the ended page for a Genero Browser Client (GBC) Web application to access the VM logs.
Log file names

Dispatcher log
The name specifies the type of dispatcher. Examples:
- `httpdispatch.log`
- `isapidispatch.log`
- `fastcgidispatch.log`

Proxy log
The name indicates the type of proxy. Examples:
- `uaproxy-<session-id>.log`
- `gwsproxy-<group>-<app>.log`

DVM logs
- Web applications
  `vm-<session-id>.log`
- Web Services
  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 `MAX_AVAILABLE` on page 404. For example, `vm-<group>-<app>-<number>.log`
  - `vm-demo-Calculator-0.log`
  - `vm-demo-Calculator-1.log`

  Note: A log file is reused for new DVM logs if the previous DVM has finished, to avoid the accumulation of log files on disk.

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, it will start and stop the Apache server. When Apache restarts, it also starts a new fastcgidispatch process (see Apache: mod_fastcgi on page 98) 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.

Related concepts
GAS installation and application data directories on page 40
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.

LOG on page 403
The LOG element specifies details of the log files the Genero Application Server creates.

Monitoring on page 176
Use the /monitor URL to view information on the current status of GAS dispatcher and on active applications.

Sensitive data in logs
The GAS logs may include some sensitive or personal user data that is gathered during the normal course of running applications.

Important: Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

HTTP messages.
Proxy logs contain HTTP messages, including the request and response content. Messages may include full documents. HTTP headers may contain query strings and cookies. Cookies may contain user credentials.

VM protocol
The proxy communicates with the DVM and can log all information exchanged. On the client side (GDC, GBC), for example, the contents of a log-in screen, including user name and password, may appear as clear text in the AUI tree and in the logs.

Note: Passwords appearing in the logs may be avoided if authentication and authorization is kept separate from the application by using a Single Sign On solution. The recommended practice for applications requiring user login is to implement the Genero SSO solution. See How to implement Single sign-on (SSO) on page 139.

Command lines
If proxies start DVM processes, all the command lines with their parameters and working directories may be logged. For example:

"Start process" Directory: /opt/fourjs/fglgws/demo,
    Command line: "/opt/fourjs/fglgws/bin/fglrun" demo.42m

VM output
All VM content written to the standard output (stdout and stderr) is redirected to the session log files (vm-<session-id>.log).

FastCGI dispatcher
If you are using the fastcgi dispatcher, all the FastCGI protocol parameters can be logged, including
Environment and Configuration

The GAS may log all environment at start up, including all system environment variables, system limits, etc. In the proxy log, some information on where the DVM will connect is also logged. For example, "FGLSERVER" 127.0.0.1:49913:

Clients send additional environment information to the logs at start up. For example, the GDC sends device MAC address, user name, host name, etc.

Monitor page

The monitor page, though not directly related to the log files, provides access to dispatcher and proxies / VM log files. It also provides a list of active sessions, with detailed information on the user agent, IP addresses, command lines used to start applications, authenticated user (CGI REMOTE_USER), environment variables, and access to configuration of available applications and services.

Reviewing the management of logs

Addressing the following questions can help towards a strategy for managing logs and protecting sensitive data.

- What are you using the data for?
  Gas log files are essential for debugging and support response, and for that purpose are used for collecting status information on the VMs and the proxies.
- Where is the data being stored?
  The data is stored on the server running the GAS. Consider who has access to the log files, and ensure access is restricted to administrators and users who need to work with the GAS log files.
- Do you still need the data?
  Consider how long you need to keep logs. Review your log file management system and remove log files after a predefined period. See Manage the Genero Application Server log files on page 186.

Related concepts

- GAS installation and application data directories on page 40
  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.
- LOG on page 403
  The LOG element specifies details of the log files the Genero Application Server creates.
- Monitoring on page 176
  Use the /monitor URL to view information on the current status of GAS dispatcher and on active applications.

Using the debugger

To debug an application you can use the FGL debugger (fgldb) with the dispatcher.

The debugger is built in to the runtime system (fglrun). It allows you to control the execution of a program step by step, so that you can find logical and runtime errors. For more information on this feature, see the Integrated debugger pages in the Genero Business Development Language User Guide.
Tip: The graphical debugger of Genero Studio allows you to start the fgldb tool on another computer through ssh, so that it is possible to debug a program running on a remote machine. For more information see the Genero Studio User Guide.

Run the debugger for the GAS on the Windows® platform

Debug an application by running the FGL debugger.

The dispatcher must open a DOS command or an xterm window and then run fglrun -d.

1. In the GAS configuration file (default $FGLASDIR/etc/as.xcf):
   Change
   ```
   <RESOURCE Id="res.dvm.wa" Source="INTERNAL">
   $(res.fgldir)\bin\fglrun.exe</RESOURCE>
   ```
   to:
   ```
   <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:
   ```
   <DVM_AVAILABLE>10</DVM_AVAILABLE>
   ```
   to:
   ```
   <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 <<< Opens the debugger tool and sets it on program test.42r.
   (fgldb)b test:20 <<< Sets a break point at line 20.
   (fgldb)run <<< Runs the application.
   ```
   This refreshes the browser.

   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™

Debug an application by running the FGL debugger.

Before you begin:
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.

1. To run the dispatcher, enter the following:

   ```
   httpdispatch -E res.dvm.wa="/home/test/xterm.sh"
   ```

   Where:
   - a. In the `xterm.sh` shell file, you have: `/usr/bin/xterm` (the complete path to xterm).
   - b. The command removes all of the options given by the dispatcher along with all error messages.

   The dispatcher starts and a new xterm is opened.

2. Type the commands to run and debug the application; 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**

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
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 \texttt{ROOT\_URL\_PREFIX} 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

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 41: 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 kind 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 42: Sessionless Request Processing

Figure 42: Sessionless Request Processing on page 193 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.
Figure 43: Cookie-based starting-session request processing

Figure 43: Cookie-based starting-session request processing on page 194 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 $G_1$.
3. The request is forwarded to $G_1$.
4. $G_1$ creates the session $S_1$.
5. The response is returned to the load balancer server.
6. A session cookie that holds the identity of $G_1$ 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 44: Cookie-based session-bound request processing on page 195 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 45: Path-based starting-session request processing on page 196 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 46: Path-based session-bound request processing on page 197 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 47: 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 48: Load Balancing Sample Architecture 2

- It is the same configuration as Figure 47: Load Balancing Sample Architecture 1 on page 198, 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
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.

```bash
# 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.

```bash
# 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/wa/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.

```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>
    </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.

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.

```apache
# 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:
  ```text
  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.

```apache
# 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 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/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 for 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 193.

```
<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 193. The example illustrates cookie-based request routing.

```
<IfModule mod_proxy.c>
```
Genero Identity Provider (GIP)

Genero provides its own Identity Provider for securing applications and RESTful Web services.

Why would I use the Genero Identity Provider?

With the Genero Identity Provider, you can:

• Secure Web applications and Web services.
• Manage users and groups.

During the initial configuration of the Genero Identity Provider, you can install the optional deployment service to help deploy and secure applications. See Genero Identity Platform components on page 213 to learn more about which components make up the Genero Identity Platform, and which services can be added during initial configuration.

How is security managed?

Security is managed by scopes. You can think of a scope as being a permission. There are two categories of scopes:

• An authorization scope allows access to an application.
• A scope allows access to a Web service. The term scope comes from the Web service world, where a function uses a scope to secure itself against unauthorized users.

Securing a Web service

The Genero Identity Provider can secure RESTful Web services. To understand how to secure a Web service, you must understand the architecture of a Web service, as it relates to permissioning.
A RESTful Web service is an application that contains one or more resources, and a resource can support multiple operations, as illustrated in Figure 49: Architecture of a RESTful Web service on page 205.

Access is secured by scopes. A scope can be set on a resource or an operation. The Web service sets the scopes in its code, implementing the scope as a string. For example, Figure 49: Architecture of a RESTful Web service on page 205 shows two resources. With the first resource, two of the three operations have a lock displayed. These locks represent scopes set on the operation level. With the second resource, the scope is set on the resource level.

In order to access a resource or operation with a scope assigned, a user must have the required scopes. That scope (string) must be embedded (encrypted) in the access token that the user gets from the Genero Identity Provider, granting them access to the resource or operation. Examine the diagram one more time, you should see that only one operation does not require the user to have a scope assigned.

**Figure 49: Architecture of a RESTful Web service**

**Securing an application**

When you secure an application, you specify which authorization scopes will allow access to the application. In order to access the application, a user must have the required authorization scope.

**Figure 50: Securing an application**

If the application makes calls to a Web service, and the Web service is secured by scopes, the user must also have the scope demanded by that Web service in order to use the operation protected by that scope.

**Figure 51: Securing an application that calls services**

When defining application access, you can specify that a Web service scope is required or optional.

- If a scope is required, then a user *must* have the scope to initially access the application.
- If a scope is optional, the user can initially access the application, even if the user does not have the optional scope. When a user tries to use a protected resource or operation, and that user does not have the necessary scope, the service sends back an error. It is up to the application to handle the error.
Managing user access

To recap:

- If the user needs to access an application, then they need to have the **authorization scope** necessary to access the application.
- If an application then interacts with a Web service, the user must have the **scopes** required by that service in order to use the protected resources and operations in that Web service. If a scope has been marked as required, the user must have the scope to initially access the application.

Scopes and authorization scopes are often provided via groups. A **group** is a collection of authorization scopes and scopes. When a user is made a member of a group, that user inherits all authorization scopes and scopes assigned to that group. In addition to inheriting scopes via group membership, users can be granted scopes directly.

Genero Identity Provider server URLs

The Genero Identity Provider is a Web service that, once set up, is available when the Genero Application Server (GAS) is started. Certain tools used for application deployment via scripts may need to access Genero Identity Provider services on a remote server.

Obtain access to the Genero Identity Provider at the GAS base URL for the **services** group:

https://host:port/gas/ws/r/services/GeneroIdentityProvider

Obtain Genero Identity Provider metadata at the following URL:

https://host:port/gas/ws/r/services/GeneroIdentityProvider/.well-known/openid-configuration

Related concepts

**Genero Identity Platform components** on page 213
Services, applications, and tools work together to secure applications and services and perform single sign-on (SSO) for applications delivered by a Genero Application Server (GAS).

**Genero Identity Platform StarterApp reference** on page 222
The StarterApp configures the Genero Identity Platform (GIP) running on a Genero Application Server (GAS).

Related tasks

**Setting up the Genero Identity Provider** on page 215
The Genero Identity Provider (GIP) must be configured and enabled before it can be used. This procedure configures and enables the GIP on a Genero Application Server (GAS).

**Configure for an external GIP** on page 220
Complete this procedure to configure the Genero Application Server (GAS) to use an external Genero Identity Provider (GIP) installed on another GAS.

Genero Identity Provider scenario

This scenario provides an overview of the process for securing applications and Web services resources with the Genero Identity Provider (GIP).

ABC Farmers is implementing a system to help them sell apples and oranges. They have written a Web service to manage their sales of fruit. Access to the Web service is provided by a browser-based application, to be made available to all ABC Farmers customers, workers, and managers. This scenario takes you through the process of securing both the Web service and the application, while setting up access for users based on their roles (customer, worker, or manager).
Create the groups

It is more efficient to secure an application for a group of users than on a user-by-user basis. Towards this end, the administrator starts by creating user groups based on the different roles, as shown in Figure 52: Groups created for ABC Farmers on page 207.

For each group, the administrator selects the scope of Role.User (for the Authorization API), as each member of these groups is a “standard user of Genero Identity Platform”. In addition, the default scope openid (for the OpenID API) is kept, as it is required for the Genero Identity Provider and single sign-on. Figure 53: Scopes of the ABC Farmers Customer group on page 207 shows the initial scopes selected for the ABC Farmers Customer group; the scopes for the Worker and Manager groups would be identical at this stage.

Secure the Web service

With the groups created, the administrator continues to secure the Web service. ABC Farmers have written a Web service to manage their resources:

- Customers can find out what fruit is available and purchase the fruit.
- Farm workers can update the inventory when new produce arrives.
- Farm managers can set prices.

In the Genero Web service application, scopes were set to secure the resources and operations, as seen in Figure 54: ABC Farmers Web Service resources and operations (with scopes defined) on page 208.
Figure 54: ABC Farmers Web Service resources and operations (with scopes defined)

The administrator must register these scopes with the Genero Identity Provider. From the ConsoleApp, The administrator registers the scopes for the Web service, as seen in Figure 55: Registered scopes for the ABC Farmers Web service. on page 208.

Figure 55: Registered scopes for the ABC Farmers Web service.

Secure the application

A secured browser-based application provides the interface to the Web services resources and operations. Customers use the application to search for and purchase the fruits, farm workers use it to update the quantities on hand, and farm managers use it to update prices. To secure this application, the administrator sets the application’s authorization scope to `Role.User`. A user must have the `Role.User` scope to access the application.

Figure 56: Authorization scopes for the ABC Farmers Portal application

The application is an interface to the ABC Farmers Web service. When considering which scopes a user needs to use the application, the administrator must also account for the scopes set in the Web service, as shown in Figure 57: ABC Farmers scopes on page 209.
Figure 57: ABC Farmers scopes

The administrator must determine which scopes are required and which are optional. For the ABC Farmers Web service, if a user does not have the abcFarmers scope, they will not be able to access any of the resources or operations. Customers, however, will not need the update.apples, update.oranges, or setprice scopes. Based on this analysis, the administrator sets the required and optional scopes for the ABC Farmers Portal application, as shown in Figure 58: Required scopes the ABC Farmers Portal application on page 209 and Figure 59: Optional scopes for the ABC Farmers Portal application on page 210.

Figure 58: Required scopes the ABC Farmers Portal application

Users without the abcFarmers scope will not be allowed to start the application, as it is a minimum scope requirement. The openid scope is required for authentication purposes.
Figure 59: Optional scopes for the ABC Farmers Portal application

Users without the optional scopes will still be allowed to start the application, however they will not be able to access the operations protected by those scopes.

Set scopes for groups

The administrator has defined the user groups, registered the scopes for the Web service, and secured the browser-based ABC Farmers Portal application. Next, the administrator selects the scopes for each group.

A customer needs the abcFarmers scope to access the resources. This scope is added to the ABC Farmers Customer group, as shown in Figure 60: Scopes added for the ABC Farmers Customer group on page 210.

Figure 60: Scopes added for the ABC Farmers Customer group

A worker also needs the abcFarmers scope to access the resources. In addition, the worker needs the update.apples and update.oranges scopes to update the inventory of oranges and apples as new shipments arrive. These scopes are added to the ABC Farmers Worker group, as shown in Figure 61: Scopes added for the ABC Farmers Worker group on page 211.
The manager group also needs the `abcFarmers` scope to access the resources. In addition, the manager needs the `setprice` scope to update the prices of apples and oranges. These scopes are added to the ABC Farmers Manager group, as shown in Figure 62: Scopes added for the ABC Farmers Manager group on page 211.

The scopes for each of the three groups are set.

### Assign groups to users

With the scopes for the groups set, the administrator must now select which group or groups each individual user belongs to. A user can belong to multiple groups; for example, a person can be both a customer (viewing and purchasing fruit) and a manager (setting fruit prices). The administrator selects the groups for each user, as shown in Figure 63: Example: List of groups user "fred" belongs to on page 211.

The user inherits the scopes assigned to the groups in which they belong. In the ConsoleApp interface, scopes inherited by group membership display as selected but with a faded gray font, as shown in Figure 64: Example: List of scopes for user "fred" on page 212.
Figure 64: Example: List of scopes for user "fred"

If groups are used well, there should never be a need to select scopes directly for a user. When a new user is registered with the Genero Identity Provider, the administrator only needs to select which groups are a match for that user, and user setup is complete.

Secure a service (script) application

The ABC Farmers managers do not always want to log in to the application to update pricing; they have already entered prices into another system. An application – a back-end service with no user interface – has been created to update the pricing of fruit based on data pulled from other systems. The script is scheduled to run daily. The script will need access to the secured Web service to complete the pricing updates.

The administrator creates a service to service application entry for the back-end service, and selects the scopes needed by the back-end service to update the prices. The scopes needed include the abcFarmers scope required to access the resources and the setprice scope required to access the operation to update the prices. These scopes are added to the service, as shown in Figure 65: Service to Service app on page 212.

Now, the script will be able to use the GetToken tool on page 251 to retrieve an access token to be used to update the prices, as it has the required scopes.

Figure 65: Service to Service app

Scenario summary

This has been an example of using the Genero Identity Provider to complete its two main goals:

- Securing applications and RESTful Web services.
- Managing users and groups.

For more information on writing a RESTful Web service, refer to the Genero Business Development Language User Guide.
Genero Identity Platform components

Services, applications, and tools work together to secure applications and services and perform single sign-on (SSO) for applications delivered by a Genero Application Server (GAS).

Figure 66: Genero Identity Platform components on page 213 represents a high level view of the components that make up the Genero Identity Platform.

Main components

The main components make up the Genero Identity Provider (GIP). The GIP uses the OAuth 2.0 protocol via the OpenID Connect SSO on page 142 to manage user access to your secure Web applications and services. When a user types the URL of a secure application into a browser, the GIP performs an identity check on the user from a login. If the user's identity is verified and they are allowed access, the user is redirected to the application.

Note: Some application and service components of the GIP are secured by the GIP and access to them requires user login. These are shown in Figure 66: Genero Identity Platform components on page 213 with the padlock symbol.

The main components are needed to secure applications and Web services and manage users. They include services, administrative applications, and interoperability with Genero command line tools.

Core Services (1)

The core services of the GIP handles:

- User authentication and Single Sign-on (SSO)
- The creation of access tokens and identity tokens
- The registration of applications
Delegate Services (2)

The delegate services validate the identity token and access tokens. There are two separate delegate services that provide each function:

1. The OpenIDConnectServiceProvider service manages all SSO delegated requests for applications. It redirects the start of a Genero application to the GIP and checks the validity of the identity token returned by the GIP via the callback URL. For more details about the SSO workflow, see Single sign-on workflow on page 141.

   This service also provides the application with an access token; the application needs to have the access token when sending an HTTP request to a REST service. The delegate service sets the environment variable (OIDC_ACCESS_TOKEN) with the access token received from the GIP and starts the application.

2. The Access delegate service manages all requests between an application and a REST web service. It decodes the access token, and gets the scope (or list of scopes) from it. It checks the token's signature and if it is valid, it forwards the access token with the scope (or scopes) to the REST service. Access is granted if scopes correspond with what is defined in the WSScope attribute for access to the resource.

   For more information on Web service scopes, see the WSScope page in Genero Business Development Language User Guide and Manage Web service access scopes on page 234.

Profile Service (3)

The profile service manages user profiles for SSO. A user profile includes a name, date of birth, phone number, email address, and more. This is a secure service you access through the Console App. The Starter App installs the profile service, or it allows you to specify an alternate profile service.

Console App (4)

The Console App provides a secure interface for registering applications and Web services to be secured by the GIP, and managing users and groups. Administrators can also view current tokens, and revoke a token's ability to renew. See Managing GIP components on page 230.

Tools (5)

The GIP integrates with command line tools such as the GetToken on page 251, DeployGar on page 253, and DeployGbc on page 256 services.

Starter App (6)

The Starter App is for the initial configuration of the GIP. Run this application once to specify the initial configuration of the GIP. It sets the administrator login and password. See Setting up the Genero Identity Provider on page 215.

The Starter App is also used when setting up a distributed environment. For the external GAS servers,
run this application and specify the host where the GIP is located. See Configure for an external GIP on page 220.

Additional components
When you install the GIP, you are given the option to install additional microservices and applications.

Deployment Service
This microservice can deploy, secure, and manage applications and Web services, and deploy and manage Genero Browser Client (GBC) customizations. Install this service on each GAS where you plan to deploy an application or Web service using the Deployment App.

Tip: The DeployGar on page 253 and DeployGbc on page 256 command line tools can also be used with the deployment service. They work with the GetToken on page 251 tool to get an access token for the service. See Automatize application deployment via scripts on page 242.

Deployment App
This application provides a secure interface to the deployment service. For instructions on using the Deployment App, see Deploying and securing applications and Web services on page 236.

SharedFileDemo service
This microservice can push and share files between users in the system. You currently must install the service and its corresponding application on the same GAS in order to use the microservice.

SharedFileDemo app
This application provides a secure interface to the SharedFileDemo service. You currently must install the service and its corresponding application on the same GAS in order to use the microservice.

Related concepts
Genero Identity Platform StarterApp reference on page 222
The StarterApp configures the Genero Identity Platform (GIP) running on a Genero Application Server (GAS).

Setting up the Genero Identity Provider
The Genero Identity Provider (GIP) must be configured and enabled before it can be used. This procedure configures and enables the GIP on a Genero Application Server (GAS).

Before you begin
- Ensure that users using the ConsoleApp and DeploymentApp have write access to the openid-connect directory. See Provide access to the openid-connect directory on page 218.
- If in HTTPS, ensure that all certificates are installed in $FGLDIR/web_utilities/certs. For instance, during an SSO handshake, fglrun will fetch the GIP metadata that may be in HTTPS, thus all appropriate certificate authority must be set.

Apache Users
Apache discards the Authorization header if it is not a base64-encoded user/password combination. A rewrite rule can be used to rewrite it from the server variable to set HTTP Authorization for requests.

For an example configuration, see Apache 2.4: Configure mod_proxy_fcgi for remote server on page 101.
For more information on Apache, see the Apache documentation.

**IIS users**

Ensure that your IIS has the appropriate rights to access the GIP home directory.

**NGINX users**

Ensure that your FastCGI Params configuration has the following directives for GIP:

- Add the HTTP authorization header:
  
  ```
  fastcgi_param HTTP_AUTHORIZATION $http_authorization;
  ```

- As GIP requires a fully qualified name, the nginx SERVER_NAME must be configured as follows:

  ```
  fastcgi_param SERVER_NAME $host;
  ```

**GIP working directory**

The default GIP working directory is set to the user's home directory at $(home)/.genero-sso. It can be changed by setting the res.path.idp resource in the GAS configuration file.

**Warning:** If the GIP is started behind an Apache or IIS server, the user's home directory is not set. In a production environment (behind Apache, nginx, or IIS), we recommend you set res.path.idp to a directory that is accessible when the GAS (and therefore the fgirun command) is started from the Web server.

**Enable the GIP on a GAS**

These steps enable the local GIP on a GAS. You follow this procedure one time only. To enable an external GIP - where a GAS connects to the GIP already enabled on another GAS - see Configure for an external GIP on page 220.

**Warning:** Any URL used with the GIP must contain a valid hostname; it cannot contain "localhost". The GIP uses cookies, and cookies do not work well with "localhost". It is recommended to use the name of the machine for the hostname. Using the IP address is not recommended, and if used at installation, then it must be used thereafter in all its endpoints.

1. **Start the StarterApp application** located at `http://host:port/gas/ua/r/idp/StarterApp`

   If you are using httpdispatch, the gas entry in the URL is not required.

   **Tip:** You can access the StarterApp from the demos page for the GAS, however you must have started the demos page using a valid hostname instead of localhost.

   The StarterApp is only accessible on the localhost and will only start once. It allows you to configure the GIP. For more information on the following configuration steps, see Genero Identity Platform StarterApp reference on page 222.

   a) Select the local GIP.
   b) Enter the new GIP administrator login and password.
   c) Enter the new OAuth secret generator password.
   d) Enter your GIP base URL.
      
      **Note:** HTTPS is recommended for the base URL, however HTTPS is not valid when using httpdispatch for the GAS.

      By default, the GIP base URL will be the GAS base URL. If the GAS is configured with a Web server in HTTPS, you can accept the default.
   e) Select whether you want the User Info Profile Service.
      
      The default is Integrated.
   f) Select whether you want the Deployment Service.
You must install the Deployment Service on the GAS, otherwise you won’t be able to deploy on that GAS protected by the GIP.

By default, Deployment Service is selected.

g) Select whether you want the Deployment App.

The Deployment App requires the Deployment Service to be installed.

The default is no. For more information, see the Genero Identity Platform StarterApp reference on page 222.

h) Select whether you want the shared file demo.

The default is no.

i) Click Install.

Check that everything installed properly. You can check in the Output panel.

The installation creates the file $\text{idp.xml}$ with endpoints to GIP services and creates and initializes databases in the GIP working directory at $\$(\text{home})/\genero\text{-ss}$.

To restart your Starter App, remove this directory completely.

2. Update the defaults for the admin user.

   a) Start the ConsoleApp as the administrator, using the login set in the previous step.
   
   The ConsoleApp is located at $http://\text{host:port/gas/ua/r/admin/ConsoleApp}$.

   b) If you have installed the User Info Profile Service, update your profile.
   
   If you kept the default of ”Integrated”, then you have installed the User Info Profile Service.
   
   Navigate to Users > Update me and click Profile. Update your avatar and any additional profile details.

   Note: First Name, Last Name and Email are mandatory.

   Click Update to save your changes.

   c) Click Groups.

   By default, the administrator belongs to group Administrator. You can add the administrator to any additional groups, if desired. As the administrator, you can always return to this page to update group membership at a later time.

   When you have finished your modifications, click Register to save your changes.

   d) Click Permissions.

   You can only add or remove scopes that are not inherited based on your group memberships.

   When you have finished your modifications, click Change to save your changes.

   e) When you have finished with your changes, click Back until you reach the front page of the ConsoleApp.


   By default, only users with the Role.Admin authorization scope can access these apps.

   a) Modify the authorization scopes for the ConsoleApp.

   Adding the authorization scopes mentioned below should allow all users and supervisors to access the ConsoleApp (to update their profile details or change their password, for example). It may still be possible that you have a user that does not have one of the three default authorization scopes, however that can be managed by ensuring that each user has at least one of these default scopes.

   Navigate to Applications > Manage apps.

   Select the row containing ConsoleApp and click Modify.

   The authorization scopes are listed at the bottom of the page. Select the Role.Supervisor and Role.User authorization scopes.

   When you have finished your modifications, click Save to save your changes.

   b) Modify the authorization scopes for the DeploymentApp.

   This step assumes you want users with the Role.Supervisor scope to access the DeploymentApp.

   Select the row containing DeploymentApp and click Modify.
The authorization scopes are listed at the bottom of the page. Select the Role.Supervisor authorization scope.

When you have finished your modifications, click Save to save your changes.

4. Create a dedicated user that is allowed to deploy new apps on any GAS protected by this GIP.

   Note: It is recommended that you create a dedicated user, however it is not a requirement.

   a) Using the ConsoleApp, create a deploy user.

      See Manage users on page 230.

   b) Assign the user to the Supervisor group.

      Click Groups. Select the Supervisor group. Click Save.

   c) Add scopes for this user.

      Click Permissions.

      Select the permissions:

      • register (OpenIDRegister service)
      • deployment (Deployment service)

      Click Save.

      The deploy user is now able to deploy and secure applications using the DeploymentApp.

5. Create an account for each user of your GIP.

   See Manage users on page 230.

6. Deploy and secure your applications.

   See Deploying and securing applications and Web services on page 236.

Provide access to the openid-connect directory

Users must have write access to the database in the openid-connect directory.

Users running the ConsoleApp or DeploymentApp must have write access to the $FGLDIR\web_utilities \services\openid-connect directory. You can grant access for a specific user or for a group of users.

If access is denied, users receive the Genero OpenIDConnect Error page and an error is written to the OIDC.log file. See the Permissions error in OICD service errors on page 249.

Related concepts

Troubleshoot GIP errors in logs on page 247

Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

Configure multiple GAS within one GIP

Configure one GIP to work for multiple distributed Genero Application Server (GAS).

The GIP can provide a federated infrastructure when in a distributed GAS environment. You might find a distributed GAS environment supporting a cloud-based solution, or where applications and services are on different servers.

Figure 67: Distributed GAS/GIP on page 219 shows an example of the GIP in a distributed GAS environment. In this distributed environment, applications running on GAS-2 access REST services on GAS-3 and GAS-4. The core services of the GIP install on GAS-1. The delegate services of the GIP install on GAS-2, GAS-3, and GAS-4. These services redirect all requests to the GIP core services on GAS-1.
Figure 67: Distributed GAS/GIP

The diagram represents the process of starting an application by performing SSO with the GAS hosting the GIP. The workflow is illustrated at a high level showing the servers involved. The communication paths are explained:

1. The user requests the start of an application (previously deployed and secured using the Genero Deployment App).
2. The SSO delegate service on the GAS where the application is deployed redirects the user agent to the GIP.
3. The GIP queries the user directly for user login and password.
4. If the login is ok, the GIP creates an ID token and an access token in its database and forwards them to the SSO delegate service callback URL (previously registered). On the callback, the delegate service requests the ID token directly from the GIP and checks its signature validity via the GIP's public key.
5. If the ID token is valid, the delegate service starts the application on behalf of the authenticated user and redirects the user agent to the initial URL for the application (/ua/r/app1).

Note: If the application needs to access resources in REST services on another GAS, the access token received from the GIP in step 4 is forwarded in HTTP requests to authenticate the access.

Configuration

In this type of configuration the GIP is hosted on one GAS. Applications on other GAS servers that require authentication are directed to the GAS hosting the GIP. The installation is a two step process:

1. Install local GIP on the host. This involves using the StarterApp to install the core components, as described in Genero Identity Platform components on page 213.
2. Install external GIPs on other GAS servers. This involves using the StarterApp to install the delegate service component only, and to set the address of the GIP host (from the previous step). See Configure for an external GIP on page 220.
Deployment services

You use the **Deployment services** to deploy, secure, and manage applications and Web services, and deploy and manage Genero Browser Client (GBC) customizations. On your distributed GAS installation, you can configure one GAS server to provide the deployment service to the other GAS servers.

The Deployment App is installed on a single GAS and this interacts with the deployment services (if installed) on each GAS that requires application or GBC deployment services.

You can specify a configuration of the deployment services with the **StarterApp** when installing the GIP:

1. On the GAS designated to host deployment:
   a. Select the option to install the Deployment App.
   b. Select the option to install the Deployment service.

   **Note:** The Deployment service option is selected by default.

2. On other GAS servers:
   a. Select the option to install the Deployment service only. See **Genero Identity Platform StarterApp reference** on page 222 and **Configure for an external GIP** on page 220.

When you run the Deployment App to deploy to a GAS on another server, you select the **Deploy > Switch GAS** option and enter its base URL endpoint in the form:

```
https://host:port/gas
```

Configure for an external GIP

Complete this procedure to configure the Genero Application Server (GAS) to use an external Genero Identity Provider (GIP) installed on another GAS.

**Before you begin**

- Ensure that users using the ConsoleApp and DeploymentApp have write access to the `openid-connect` directory. See **Provide access to the openid-connect directory** on page 218.
- If in HTTPS, ensure that all certificates are installed in `$FGLDIR/web_utilities/certs`. For instance, during an SSO handshake, `fglrun` will fetch the GIP metadata that may be in HTTPS, thus all appropriate certificate authority must be set.

**Apache Users**

Apache discards the Authorization header if it is not a base64-encoded user/password combination. A rewrite rule can be used to rewrite it from the server variable to set HTTP Authorization for requests.

For an example configuration, see **Apache 2.4: Configure mod_proxy_fcgi for remote server** on page 101.

For more information on Apache, see the [Apache documentation](#).  

**IIS users**

Ensure that your IIS has the appropriate rights to access the GIP home directory.

**NGINX users**

Ensure that your FastCGI Params configuration has the following directives for GIP:

- Add the HTTP authorization header:
  ```
  fastcgi_param HTTP_AUTHORIZATION $http_authorization;
  ```

- As GIP requires a fully qualified name, the nginx `SERVER_NAME` must be configured as follows:
  ```
  fastcgi_param SERVER_NAME $host;
  ```

**GIP working directory**
The default GIP working directory is set to the user's home directory at $\$(home)/.genero-sso. It can be changed by setting the res.path.idp resource in the GAS configuration file.

**Warning:** If the GIP is started behind an Apache or IIS server, the user's home directory is not set. In a production environment (behind Apache, nginx, or IIS), we recommend you set res.path.idp to a directory that is accessible when the GAS (and therefore the fglrun command) is started from the Web server.

**Steps to enable an external GIP**

This procedure is for a multi-GAS environment architecture, where the initial GIP configuration has already occurred on a different Genero Application Server. You follow this procedure one time only.

**Warning:** Any URL used with the GIP must contain a valid hostname; it cannot contain "localhost". The GIP uses cookies, and cookies do not work well with "localhost". It is recommended to use the name of the machine for the hostname. Using the IP address is not recommended, and if used at installation, then it must be used thereafter in all its endpoints.


   The application is only accessible on the localhost and will only start once.

   **Tip:** There is a link to start it from the GAS welcome page.

   a) Select External IdP.

   b) Provide the external GIP Issuer URL of the GAS.

   ```
   https://host:port/gas/ws/r/services/GeneroIdentityProvider
   ```

   **Tip:** The StarterApp concatenates the URL with the ./well-known/openid-configuration string to fetch the GIP metadata. You can view the metadata at:

   ```
   https://host:port/gas/ws/r/services/GeneroIdentityProvider/.well-known/openid-configuration
   ```

   c) Select whether you want to install the Deployment Service.

      You must install the Deployment Service on the GAS, otherwise you won't be able to deploy on that GAS protected by the GIP.

      The default is yes.

   d) Select whether you want to install the Deployment App.

      The Deployment App requires the Deployment Service to be installed.

      The default is no. For further information, see Table 22: Genero Identity Platform Installation (external mode) on page 224.

   e) Select whether you want the shared file demo.

      The default is no.

   f) Click **Install**.

      You will be queried for the credentials of an user authorized to register applications on the GIP.

      The installation creates an idp.xml file in the GIP working directory.

      Check that everything installed properly. You can check in the Output console.

2. Deploy applications created with fglgar on a GAS protected by an external GIP.

   a) Start the Deployment App as an authorized user.

   b) When deploying a Genero Archive (gar) file, enter the roles and scopes for each xcf of the Genero Archive.

   c) Click **Register All**.

      The GIP registers the Genero Archive.

   d) If the same Genero Archive (gar) file must be deployed on several GAS, repeat the procedure; however the role will not be updated. Only the initial deployment can set the role.
Genero Identity Platform StarterApp reference

The StarterApp configures the Genero Identity Platform (GIP) running on a Genero Application Server (GAS).

Before you can use the GIP, you must configure the GIP. The StarterApp configures the GIP. You run the application one time to configure the GIP.

If you are setting up the GIP for the first time, follow the instructions here: Setting up the Genero Identity Provider on page 215.

If your architecture involves multiple GAS servers, you initially set up the GIP on one GAS. After the initial configuration, you then configure the other GAS servers to link to that initial setup. Find the instructions for configuring those GAS servers here: Configure for an external GIP on page 220.

The remainder of this topic serves as a complete reference to the StarterApp. Its purpose is to provide you with the detail behind each decision you are asked to make while using the StarterApp.

Starting the StarterApp

To start the StarterApp, enter the StarterApp URL:

```
http://host:port/[gas]/ua/r/idp/StarterApp
```

In development, you may be using the standalone dispatcher. For the standalone dispatcher, the StarterApp URL is:

```
http://host:port/ua/r/idp/StarterApp
```

Completing the StarterApp forms

The StarterApp consists of two forms.

- The Genero IDP first installation form. See Table 20: Genero IDP first installation on page 222.
- The Genero Identity Provider Installation form is based on the installation mode selected.
  - For a local mode installation, see Table 21: Genero Identity Platform Installation (local mode) on page 222.
  - For an external mode installation, see Table 22: Genero Identity Platform Installation (external mode) on page 224

Table 20: Genero IDP first installation

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Instruction</th>
</tr>
</thead>
</table>
| Select Genero IDP installation mode | • Select "Local" if you are configuring the GIP:  
  - for a GAS sitting behind a Web server, when you are on the same machine as the Web server and GAS.  
  - for a standalone GAS dispatcher (httpdispatch).  
  • Select "External" if you are in a multi-GAS environment, and the GIP has already been configured for a GAS on an external machine. |

Table 21: Genero Identity Platform Installation (local mode)

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create the administrator account</td>
<td>Create the GIP administrator account. This account has access to GIP components, such as the ConsoleApp and DeploymentApp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Login</th>
<th>Enter your login</th>
</tr>
</thead>
</table>
| Password | Choose a password  
  re-enter the same password |

| Password | Choose a password  
  re-enter the same password |
The GIP uses the OAuth password when generating OAuth secret IDs. For example, when deploying and securing applications, the GIP uses this password to compute the `SECRET ID`. Safely store the OAuth password entered here, in case of system failure. Restoring it later will allow you to recreate all secret ids of the OAuth protocol.

The Base URL is the base URL for the Genero Application Server. The StarterApp computes the default URL from the address of the GAS running the StarterApp. If the GAS is configured with a Web server in HTTPS, the base URL will use the HTTPS protocol. HTTPS is preferred for GIP. You can typically accept the default URL. Edit this setting if you intend the GIP to use a different GAS.

The profile service is a component of GIP that maintains user data such as name, date of birth, email address, etc., for registered users on the GIP:

- Select "None" if you do not want to install the user profile service.
- Select "Integrated" if installing the user profile service on the same host as the GIP. This is the default option.
- Select "External" if you have installed the user profile service on another machine, and need to provide the URL to the service.

"Deployment service" are a set of services required by the GAS to manage Genero Archive (gar), and GBC. This option is selected by default. The service is used by the DeploymentApp and the command line tools that use its micro services to deploy.

- Select "Deployment app" to install the GUI tool that connects to the Deployment services. It provides an alternative to using command line tools for deploying GAR and GBC.

**Tip:** In a multi-GAS set up, the DeploymentApp can be on one machine. You just need to ensure the Deployment services are running on the machines you want it to deploy to.

- Select "Shared file demo" to view a demo of the shared file microservice.
Table 22: Genero Identity Platform Installation (external mode)

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAS base URL</td>
<td>Typically, you can accept the default base URL as the StarterApp automatically computes this from the address of the GAS you run the app from.</td>
</tr>
<tr>
<td></td>
<td>If the GAS is configured with a Web server in HTTPS, the base URL will use the HTTPS protocol. HTTPS is preferred for GIP.</td>
</tr>
<tr>
<td></td>
<td>Edit this setting if you intend the GIP to use a different GAS.</td>
</tr>
</tbody>
</table>

- "Deployment service" are a set of services required by the GAS to manage Genero Archive (gar), and GBC. This option is selected by default. The service is used by the DeploymentApp and the command line tools that use its micro services to deploy.
- Select "Deployment app" to install the GUI tool that connects to the Deployment services. It provides an alternative to using command line tools for deploying GAR and GBC.

**Tip:** In a multi-GAS set up, the DeploymentApp can be on one machine. You just need to ensure the Deployment services are running on the machines you want it to deploy to.

- Select "Shared file demo" to view a demo of the shared file microservice.

Related tasks

Setting up the Genero Identity Provider on page 215

The Genero Identity Provider (GIP) must be configured and enabled before it can be used. This procedure configures and enables the GIP on a Genero Application Server (GAS).

Quick Start: Set authorization for group

This Quick Start illustrates the steps needed to authorize access for a group of users.

Enable the GIP on a GAS

The Genero Identity Provider (GIP) must be configured and enabled before it can be used - see Setting up the Genero Identity Provider on page 215.

Authorize access to a group

Figure 68: The workflow for configuring authorization for a group of users on page 225 shows a high level view of the different steps needed to configure access to applications and Web service deployed and secured by the GIP. To use the GIP efficiently, you need to create groups with the scopes (authentication and access permissions) needed by users. The topics provide details of this configuration.
Create a user group

The ConsoleApp provides a graphical user interface to work with user and groups. It is a component of Genero Identity Provider.

From the ConsoleApp, you can:

- Create users.
- Add users to a group.

The Genero Identity Provider (GIP) must be configured and enabled before it can be used - see Setting up the Genero Identity Provider on page 215.

1. Log in to the ConsoleApp as administrator

   The Console App is located at http://host:port/gas/ua/r/admin/ConsoleApp.

   **Tip:** Select the Console application link from the GAS demos page at http://host:port/gas/demos.html page.

2. Select Groups > New

3. Enter the group name and description and click Create.

   It is good practice to name groups to reflect the role or function its users will perform. For example, the GIP has three default groups for users using the GIP, Administrator, Supervisor, and User. For other examples of creating groups effectively see Genero Identity Provider scenario on page 206. For the purpose of this quick start, name the group, "MyGroup".

4. Add a user to the group.

   a) Select Users > New

   b) In the New user page, enter a user name and password.

      For example, for the purpose of this quick start, call your user "UserA". Choose a simple password that is easy to remember for the user. The user will have to change the password later.

   c) When finished, click Register to save your changes.

      The user is registered with GIP and the User page opens.

      **Note:** The fields First name, Last name, and Email in this page need to be completed by the user logging in to the ConsoleApp to create a profile. See the Set group scopes for profile API on page 228 for details of the scopes they need.

   d) Click Groups.

      In the List of groups user (UserA) belongs to page, select the group created in step 3 on page 225.

      When finished, click Save.
A user is created and as a member of the group they inherit its scopes from the group. Click on Permissions to open the **Select scopes for user (UserA)** page. Scopes inherited from the group appear selected but grayed out to indicate these are inherited scopes.

**Related concepts**
- **Manage users** on page 230
- Each user trying to access an application or Web service protected by the GIP must be a user recognized by the GIP.
- **Manage groups** on page 231
- Groups provide an easy mechanism for gathering a set of scopes required by a group of users.

**Related reference**
- **GIP API scopes reference** on page 258
- Reference information for Genero Identity Platform API scopes.

### Set group authorization scopes

Authorization scopes allow users access to application protected by the GIP.

Authorization scopes are scopes to manage access to applications protected by the GIP. They are used to establish the user's identity and define the access level. For more information on authorization scopes, see **Groups are not Authorization Scopes** on page 231.

From the ConsoleApp, you can set authorization scopes for users by assigning scopes to the group.

1. **Log in to the ConsoleApp as administrator**
   - **Tip:** Select the **Console application** link from the GAS demos page at `http://host:port/gas/demos.html` page.

2. **Select Groups > Manage groups.**

3. **From the Group List page, select the group name you created previously in Create a user group** on page 225.

4. **Double click (or click **Modify** to open the **Group** page.**

5. **Select the scopes to give to the group.**
   - Select **Role.User** from the row under the **API, Scope, and Description** heading. By default, the **openid** scope is selected. When finished click **Save**.

   **Note:** **Authorization scopes: openid, Role.User**

   These are default authorization scopes:
   - **openid**, is the scope required by the Openid API to support OpenID-Connect authentication and Single sign-on.
   - **Role.User**, is a scope defining the access level for a standard user of the Genero Identity Platform required by the Authorization API.

   Your group needs to have at least these scopes. Later you will return to this page to select more scopes required for the applications and services users access.

**Related concepts**
- **Manage authorization scopes** on page 234
- Authorization scopes determine application access.

**Related reference**
- **GIP API scopes reference** on page 258
- Reference information for Genero Identity Platform API scopes.

### Deploy and secure an app

Deploy and secure an application using the Genero Deployment App. This registers the application in the GIP.

**Before you begin**
You must have packaged your applications and services into a Genero Archive. For more information about packaging a Genero Archive, see Understanding packaging with GAS on page 271.

**Tip:** A simple application such as the "price" application located in $FGLDIR/testing_utilities/ggc/src/quick-start/ggc-quick-start.gar can be used for the purpose of this quick-start.

1. Start the Genero Deployment App as an authorized user.
   **Tip:** Select the Deployment application link from the GAS demos page at http://host:port/gas/demos.html.

2. Deploy the .gar file.
   a) Select Deploy > Genero Archives ....
   b) Click Deploy.
   c) Use the Select your Genero Archive to deploy dialog to select your Genero Archive (.gar).
   The Genero archive deploys and is listed under Genero Archive(s).

3. Secure the Genero Archive.
   a) Select the Genero Archive row.
   b) Click Secure.
   The Genero Archive is secured, and the individual applications are ready for security configuration.

4. Set authorization and required scope for the app:
   a) In the Secured panel, select the row for an application.
   b) Click Configure.
   The configuration page appears.
   c) Under the Authorization tab, select the Role.User scope.
   d) Under the Required tab check that the default scope, openid, is selected.
   At this stage no more scopes are needed by the application. A user with the Role.User scope can access your app. The app has the Openid API scope to handle user access via OpenID-Connect authentication and Single sign-on.
   **Tip:** If later you need to configure more user scopes, or scopes for services or APIs used by the app, you can return to this page, or from now on you can configure the registered app using the ConsoleApp.
   e) When finished, click Save.
   A dialog box will ask you to confirm your changes, click Yes.
   The application is secured by the Genero Identity Provider (GIP).

5. Change the status of the deployed Genero Archive to Active.
   a) Select the row containing the deployed app.
   b) Click Enable.
   The status changes from Inactive to Active.

**Test user access to the application:**

At the browser, enter the application URL.

**Note:** The application deployed is named with the .xcf, not the Genero Archive (.gar) file name. To get details of deployed applications run the gasadmin gar --list-archives command where the GAS is running.

For example, if you deployed the "ggc-quick-start.gar", the "price" application was deployed. You access it at http://host:port/gas/ua/r/price:

- Check that access to the application requires user login.
- Check that the authorized user ("UserA") you created in Create a user group on page 225 can access the application.

**Related concepts**

Manage applications on page 232
Use the Genero Console App to manage the access to the applications and service applications whose access is to be managed by the Genero Identity Provider (GIP) on the selected Genero Application Server.

Manage Web services on page 233
Scopes provide access to Web services. You will use the Console App to register the scopes, but you will not explicitly register a Web services application.

Set group scopes for profile API
GIP users may need to have access to the Profile API in order to create a user profile, update their details, and avatar.

Before you begin
The Profile API has several scopes. In this task we focus on the profile.me scope, which is required by users to access the ConsoleApp. In the next task, Set group scopes for app using a service on page 228, you will get to see how to use other profile scopes.

In this task you will configure user access to the Genero Console App:
• you test for access to the ConsoleApp using the application URL.
• you set the profile.me scope for your user group.
• you complete the task by testing again for access to the ConsoleApp.

1. Start the Genero Console App
   Tip: Select the Console Application link from the GAS demos page at http://host:port/gas/demos.html.
   At the log-in page enter the log-in details for the user ("UserA") you created in Create a user group on page 225.
   You get an access is denied message. This is expected.
   Access denied: some scopes are required
2. Log in to the ConsoleApp as administrator.
3. Select Groups > Manage groups.
4. From the Group List page, select the group name you created previously in Create a user group on page 225.
5. Double click (or click Modify) to open the Group page.
6. Select the profile.me scope.
   Scopes are listed in the rows under the API, Scope, and Description headings. When finished click Save.

Test user access to the Console App:
Repeat step 1 on page 228 to check that the authorized user ("UserA") you created in Create a user group on page 225 can access the Console App. You should be able to create a user profile (Users > Update me menu) now by entering details in the First name, Last name, and Email fields.

Related concepts
Genero Identity Platform components on page 213
Services, applications, and tools work together to secure applications and services and perform single sign-on (SSO) for applications delivered by a Genero Application Server (GAS).

Set group scopes for app using a service
Access to an application using a Web service requires user scope for the Web service.

Before you begin
This procedure is provided to give you a quick start, or overview, of configuring user access to applications that rely on Web services. For example, the Genero Identity Provider (GIP) comes with a share file application that allows registered GIP users to share files. The "SharedFileApp" uses the ShareFile Web service. For the purpose of this quick start you will configure user access to this application and Web service. In this task:
you test for access to the Genero SharedFile App using the application URL.
• you set the required scopes for your user group to provide access.
• you complete the task by testing again for access to the SharedFile App.

1. Start the Genero SharedFile App

   **Tip:** Select the Shared File Application link from the GAS demos page at http://host:port/gas/demos.html.

   At the login page enter the login details for the user ("UserA") you created in Create a user group on page 225.
   You get an access is denied message:
   
   Access denied: some scopes are required

2. Log in to the ConsoleApp as administrator.
3. Select Groups > Manage groups.
4. From the Group List page, select the group name you created previously in Create a user group on page 225.
5. Double click (or click Modify) to open the Group page.
6. Select the scopes to give to the group.

Scopes are listed in the rows under the API, Scope, and Description headings. Your group needs to have these scopes to use the SharedFile App:

a) Select shareFile
   This scope is required and provides user access to the SharedFile Web service.

b) Select profile.
   This scope is required by the user to provide their profile information, making it visible to other users.

c) Select profile.query.
   This scope is required by users sharing files with other users. It allows them to see a list of users and select users for file sharing.

d) When finished click Save.

**Test user access to the SharedFile App:**

Repeat step 1 on page 229 to check that the authorized user ("UserA") you created in Create a user group on page 225 can access the Genero SharedFile App. See Share files using the Genero SharedFile App on page 245 for details of how to share files.

**What to do next**

Having completed this quick start you have a basic configuration of GIP working, allowing registered users access to secured applications and services. There is more to learn. For example, learn how to define and manage scopes for a Web service. See Manage Web services on page 233 and Manage Web service access scopes on page 234. For examples, see Genero Identity Provider scenario on page 206.

Explore the Managing GIP components section for details of what to do when managing users, groups, service-to-service apps, etc., or see the GIP components overview topic.

There are different ways to deploy and secure apps, including using scripts that also need to be registered on the GIP. See the topics in the Deploying and securing applications and Web services on page 236 section.

**Related reference**

GIP API scopes reference on page 258
Reference information for Genero Identity Platform API scopes.

Managing GIP components

The Genero Identity Provider (GIP) can manage users, groups, applications, tokens, authorization scopes, and Web service access scopes. These are managed using the Console App.

To access the Console App, open a browser and enter: `host:port/ua/r/admin/ConsoleApp`

A link to the Console App is also provided on the Genero Application Server demos page: `host:port/demos.html`

Who can access the Genero Console App

Within the GIP, access to an application is provided by authorization scopes. The Console App is managed by the GIP, therefore a user must have the required authorization scope to access the Console App.

By default, only members of the `Role.Admin` authorization scope can access the Console App. You can change this by adding and removing authorization scopes for the Console App; see Manage applications on page 232 to learn more.

Manage users

Each user trying to access an application or Web service protected by the GIP must be a user recognized by the GIP.

To manage users, you must be able to access the Console App. Access to the Console App is managed by authorization scopes; see Managing GIP components on page 230 to learn more.

From the Console App, use the Users menu to:

- Add a user.
- Update or delete a user.
- Update your personal profile.

Who can manage users?

Access to the various user management functions is controlled by scopes set by the Profile Service, the Web service that manages user profiles. See Genero Identity Platform components on page 213 to learn more about the Profile Service.

To add, update, or delete users, you must:

- Be a member of the Administrator group.
- Have the `profile.mgr` scope.

To update your personal profile, you must:

- Have the `profile.me` scope. Having this scope enables the Users > Update me menu item.

Add a user

To add a user, select Users > New.

In the New user form, you simply provide the user login and a new password. Choose a simple password for each user. The user will have to change the password later. You are not permitted to provide any further user profile information; it is expected that each user will log in to the Genero Console App and change their password, as well as update their own profile details. Once you have provided the login and password details, click Register.

With the user registered, you will need to add the user to one or more groups. Click Groups. Select one or more groups and click Save.
To add additional scopes for the user, click Permissions. Permissions that are inherited due to group membership are selected and grayed out; they cannot be changed from this page. For the remaining permissions, you can select or de-select scopes. Click Save to save your modifications.

Once you have finished adding the user to groups and giving the user additional scopes, click Back to return to the main page of the Genero Console App.

**Manage a user**

For users that exist, you can choose to reset the user password, add or remove permissions, or add or remove membership to groups.

To manage a user, select Users > Manage users. Select the row of the user to manage and click Modify. The User form opens with the user name shown in the read-only Login field.

To reset the password, click Password and complete the form.

To add or remove permissions, click Permissions. Permissions that are inherited due to group membership are selected and grayed out; they cannot be changed from this page. For the remaining permissions, you can select or de-select scopes. Click Save to save your modifications.

To add or remove membership from a group, click Groups. Select one or more groups and click Save.

**Remove a user**

To remove a user, select Users > Manage users. Select the row of the user to remove and click Remove. A confirmation dialog appears asking you to confirm your selection, as the removal cannot be undone.

**Manage groups**

Groups provide an easy mechanism for gathering a set of scopes required by a group of users.

To manage groups, you must be able to access the Console App. Access to the Console App is managed by authorization scopes; see Managing GIP components on page 230 to learn more.

From the Console App, use the Groups menu to:

- Add or delete a group.
- Add scopes to a group.
- Remove scopes from a group.

**Groups are not Authorization Scopes**

The three default groups are Administrator, Supervisor, and User. The three default authorization scopes are Role.Admin, Role.Supervisor, and Role.User. They are not the same thing!

A distinction must be made between groups and authorization scopes:

- Groups are a collection of scopes. When a user becomes a member of a group, they inherit its scopes.
- Authorization Scopes provide access to applications. An application has one or more authorization scopes; members of the assigned authorization scopes can access the application.

You can assign authorization scopes to groups. For example, the Administrator group has the Role.Admin scope selected by default. It can, however, be deselected, or you can add the authorization scopes Role.Supervisor and Role.User to the Administration group. In addition, the Administrator group can have scopes selected that are not authorization scopes.

**Who can manage groups**

To add, manage, or remove groups, you must have the Role.Admin authorization scope.
Add a group
To add a group, select **Groups > New**. Enter the group name and description and click **Create**.
Once the group is created, select which scopes to give to the group. By default, the `openid` scope for the OpenID API is selected; this scope is necessary in order for the group to support OpenID-Connect authentication. Select any additional scopes and click **Save**.

Manage a group
To manage a group, select **Groups > Manage groups**. Select the row of the group to manage and click **Modify**. The **Group** form opens to display the group name, description and scope selections.
You cannot alter the group name or description.
You can select or remove scopes. Click **Save** to save your changes.

Remove a group
To remove a group, select **Groups > Manage groups**. Select the row of the group to remove and click **Remove**.

Manage applications
Use the Genero Console App to manage the access to the applications and service applications whose access is to be managed by the Genero Identity Provider (GIP) on the selected Genero Application Server.

Applications: apps and service apps
The Genero Identity Provider (GIP) can manage the access to:
- Genero applications that are run on behalf of a user (called **apps** in the Genero Console App interface)
- Genero applications that do not have to be run on behalf of a user (called **service to service apps** in the Genero Console App interface). This type of app often refers to a script that runs without user interaction, often on a timed schedule. One example of a service to service app would be a monitoring tool that runs at night; it could be a simple Genero script or a graphical tool. Another example could be a bash application with no user behind it.

How applications get in the list
Two methods can secure an application by the GIP:
- Using the Deployment App to add an application.
- Using the Console App to register an application, then modifying the application configuration file included in the package.

Manage apps
Select **Applications > Manage apps** to view the list of apps and service to service apps managed by the Genero Identity Provider (GIP). Select the row of the app you want to manage and click **Modify**.

Manage app info
You can update the information regarding the app or service to service app, to include the:

**Description**
Use this field to describe the app or service app.

**Client ID**
The Client ID is generated by the GIP.
The Client ID displayed in the Genero Console App must match the `<CLIENT_PUBLIC_ID>` in the application configuration file for the app or service app.
If you click **Generate**, it updates the Client ID.

**Secret ID**
The Secret ID is generated by the GIP.
The Secret ID displayed in the Genero Console App must match the `<CLIENT_SECRET_ID>` in the application configuration file for the app or service app.

Redirect URLs

The redirect URL (or URLs) is the URL of the OpenIDConnect service on the GAS where the app has been installed. It is the GAS URL + address of the OpenIDConnect service (provided by default in GWS, and called OpenIDConnectServiceProvider.xcf) + the entry point of the service (by default, "oauth2callback") where the IdP will redirect the user-agent to provide the OAuth2 ID token.

Following this explanation, unless some URL rewriting has occurred and is configured in the web server, the URL will be:

http[s]://host:port/[gas/]ws/r/services/penIDConnectServiceProvider/oauth2callback.

Manage scopes

You can also update the authorization, required and optional scopes (for applications) or scopes (for service to service applications).

For the managed app:

- The authorization scope identifies the scopes that will permit to access the app. If you need a new authorization scope that is not listed, see Manage authorization scopes on page 234.
- The required scopes are those scopes required to use the application.
- The optional scopes are those scopes required by some parts of the application; however the application can still be run without the user having these scopes; it may not be fully functional.

For the managed service to service app, the scopes identify those scopes that the service application needs to run its scripts.

Related reference

GIP API scopes reference on page 258
Reference information for Genero Identity Platform API scopes.

Manage Web services

Scopes provide access to Web services. You will use the Console App to register the scopes, but you will not explicitly register a Web services application.

Register scopes

While you do not explicitly register a Web service application, you must use the Genero Console App to register the scopes that the Web service uses to provide access, and to then assign those scopes to the correct groups and users.

For more information, refer to these procedures:

- Manage Web service access scopes on page 234
- Manage groups on page 231
- Manage users on page 230

Deploying a Web service

Refer to these procedures for instructions on deploying a Web service.

- Deploy a Web service using the Genero Deployment App on page 239
- Deploy a Web service NOT using the Genero Deployment App on page 241

**Manage authorization scopes**

Authorization scopes determine application access.

**Default authorization scopes**

The Genero Identity Provider (GIP) comes with three default application scopes:

- **Role.Admin**
  The Role.Admin authorization scope is intended for your GIP admin. An admin manages the users and provides their permissions.

- **Role.Supervisor**
  The Role.Supervisor authorization scope is intended for supervisors. A supervisor installs new applications for a set of users and/or groups.

- **Role.User**
  The Role.User authorization scope is intended for your standard user. A user runs the installed applications.

**Add an authorization scope**

You can add new authorization scopes.

To add an authorization scope, select Security > Authorization, then click Create. Enter the new scope name (it must start with "Role.") and description, and click OK to save your changes.

Once added:

- It appears as a scope (under the Authorization API) when you are working with Groups (you can add it to a group)
- It appears in the user-specific scope list when you go to manage a user.
- It appears as a possible authorization scope for an application.

**Modify an authorization scope**

You can change the name and/or description of an existing authorization scope.

To modify an authorization scope, select Security > Authorization. Select the row containing the scope to modify and click Change. The fields for the selected scope become editable.

The Name field must start with the preface "Role."

When you have completed your edits, click OK to save your changes.

**Remove an authorization scope**

To remove an authorization scope, select Security > Authorization. Select the row containing the scope to remove and click Remove. A confirmation dialog appears asking you to confirm your selection, as the removal cannot be undone.

**Related reference**

GIP API scopes reference on page 258
Reference information for Genero Identity Platform API scopes.

**Manage Web service access scopes**

Access to services is managed by scopes. In order to grant users or groups scopes, they must be entered into the Genero Identity Provider. It is from this interface that you provide the list of scopes for a service.

Select Security > Services to access the Service access list. From this page, you can create a new service, modify the name and description of an existing service, remove a service, or view/modify the scopes of a service.
The **Service access list** is where you enter all scopes, with the intention to later assign the scopes to users or groups. These scopes are organized under a **Service name**. This name is not necessarily a direct correlation to any specific service; it is simply a heading that you can use to organize a set of scopes. You may find that you have one set of scopes that are being used to provide or restrict access to a complex system of services and operations - the service name you choose to enter those scopes in the service access list will likely be a generic term representing the group of services.

**Add a new set of scopes**

To add a new set of scopes:

1. Select **Security > Services**.
2. Click **Create**.
   
   A row appends the service access list.
3. Enter in a service name and description and click **OK**.
   
   **Note:** The service name can be any name of your choosing. While it does not have to match the name of your Web service, it would be advantageous to pick a name that is meaningful.

   The service is created with a single default scope, matching the name of the service. This default scope name can be changed if needed.
4. To further view or modify the scopes of your new service, see [View or modify scopes of a service](#) on page 235.

**Modify a service name or description**

To modify a service name or description:

1. select **Security > Services**.
2. From the **Service access list**, select the row containing the service name or description to modify.
3. Click **Change**.
   
   The fields for the selected row are made editable.
4. Make your modifications and click **OK**.

**Remove a service and its scopes**

To remove a service and its scopes:

1. Select **Security > Services**.
2. From the **Service access list**, select the row containing the service name to remove.
3. Click **Remove**.
   
   A confirmation dialog appears asking you to confirm your selection, as the removal cannot be reversed.

**View or modify scopes of a service**

By default, a service is created with a single scope matching the service name. The default scope name can be changed, if needed, and additional scopes can be added, modified, or removed.

To add, modify or remove scopes to a service:

1. Select **Security > Services**.
2. From the **Service access list**, select the row containing the service name.
3. Click **Scopes**.
   
   The scope list for the selected service appears.
4. Using the buttons at the top of the **Scope list** page, you can create, modify, or remove scope entries.
Manage refresh tokens

View the list of active tokens generated by the Genero Identity Provider (GIP), and revoke tokens if necessary.

The GIP manages tokens. When a user accesses a secure application, the GIP issues an access token and a refresh token. Access tokens are valid for a specific period of time, after which they must be renewed. The refresh token allows the access token to be renewed at intervals, without prompting the user to log in again. The access token continues to be refreshed until the user closes the application. At that time, the refresh token is removed by the GIP.

Note: If the user does not close the application properly, the refresh token is not removed. The GIP will discard it after one week.

From the Console App, you can revoke the refresh token.

Tip: The GetToken on page 251 program provides a set of command line options for working with access tokens.

Token details

Select Security > Tokens to view the list of tokens.

For each active token, you can view the following:

- Token - The refresh token string.
- Application - The application that the token was issued for.
- User - The user that requested access to the application.
- Expires - The expiry date of the token.
- URI - The redirect URI. This is the callback URI to the GAS where the application is started. In the case of a distributed environment, it specifies which GAS the application is started on. It identifies where the access token corresponding to the refresh token has been delivered.
- Scopes - The list of scopes encoded in the access token.

Revoke a refresh token

Revoking means you remove the ability to renew the access token granted in the refresh token. After revoking a token, if the user is logged in to the application, they will be able to continue to use the application until the next time a refresh is needed. The refresh will not occur, and the user will need to restart the application and log in again.

To revoke a token, select the token (by using the checkbox) and click Revoke.

To prevent the user from restarting the application, revoking the refresh token is not enough. You must remove the scopes that allowed the user to access the application. See Manage users on page 230.

Related concepts

Genero Identity Platform components on page 213
Services, applications, and tools work together to secure applications and services and perform single sign-on (SSO) for applications delivered by a Genero Application Server (GAS).

Troubleshoot GIP errors in logs on page 247
Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

Deploying and securing applications and Web services

You have two methods for deploying and securing applications and Web services.

The Deployment App provides you with the interface for deploying and securing applications.

The Console App allows you to register the application and generate a CLIENT_ID and SECRET_ID for that application. These are added to the application's configuration file, and the application configuration file is included in the final package (.gar).
Deploy and secure an application using the Genero Deployment App

Deploy and secure an application using the Genero Deployment App. Start with the Genero Archive (.gar) for the application.

Before you begin

- Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.
- You must have packaged your applications and services into a Genero Archive.

Note: When you create a Genero Archive (.gar) with fglgar gar using the --service wsname.42m option, a Web service configuration (.xcf) file is automatically generated from the app. If you have an existing Web service configuration (.xcf) that you would rather use, you must create a MANIFEST file pointing to your Web service configuration file.

For more information about packaging a Genero Archive, see Understanding packaging with GAS on page 271.

About this task

When you deploy the contents of a Genero Archive using the Genero Deployment App, the registration of the application is managed for you, to include the creation of a CLIENT_ID and SERVICE_ID for the deployed Genero Archive and the addition of the GIP details to the generated application configuration file.

1. Start the Genero Deployment App as an authorized user.


2. Deploy the .gar file.
   a) Select Deploy > Genero Archives ....
   b) Click Deploy.
   c) Use the Select your Genero Archive to deploy dialog to select your Genero Archive (.gar).

   The Genero archive deploys and is listed under Genero Archive(s).

3. Secure the Genero Archive.
   a) Select the Genero Archive row.
   b) Click Secure.

   The Genero Archive is secured, and the individual applications are ready for security configuration.

4. For an app:
   a) In the Secured panel, select the row for an application.
   b) Click Configure.

   The configuration page appears, where you can select authorization and required scopes for the selected application.
   c) Under the Authorization tab, select the authorization scopes required to access the application.
   d) Under the Required tab, select those scopes required by the application.
   e) When you have selected the scopes needed by the application, click Save.

   A dialog box will ask you to confirm your changes, click Yes.

   The application is secured by the Genero Identity Provider (GIP).

5. Change the status of the deployed Genero Archive to Active.
   a) Select the row containing the deployed app.
   b) Click Enable.

   The status changes from Inactive to Active.
6. If the same Genero Archive (.gar) must be deployed on several GAS, repeat the procedure; however the role will not be updated. Only the initial deployment can set the role.

Related tasks
Deploy a Web service NOT using the Genero Deployment App on page 241
Deploy and secure an Web service without using the Genero Deployment App.

Deploy an application NOT using the Genero Deployment App
Deploy and secure an application without using the Genero Deployment App.

About this task
When you deploy and secure an application without using the Genero Deployment App, you must first register the app using the Genero Console App, then add the configuration information required by the Genero Application Server to start the application using the security provided by the Genero Identity Provider.

1. Register the app in the GIP using the Genero Console App.
   a) Open the Genero Console App.
      See Managing GIP components on page 230.
   b) Select Applications > New app or Applications > New service to service app.
      Note: The New service to service app option is not related to a Web service! It refers to a script application (an application running without an identified user and with no user interface). For example, you may have a batch process that runs nightly to gather information from different services. As there is no user or user interface, you must first register the script application and assign it scopes. The interface will return the CLIENT_ID and SECRET_ID that you will have to use in the GetToken on page 251 command line tool to retrieve a valid access token with the authorized scope. The script can then use the access token to access the resources. See Automatize application deployment via scripts on page 242 for an example that uses the GetToken command line tool.
      The Registering a new application form appears.
   c) In the Info section of the form, complete the fields.
      The Name must match the name of the application configuration file.
      The Client ID and Secret ID are generated for you. You will need these values later in this procedure.
      The Redirect URLs can use "localhost" when all components – the Genero Identity Provider (GIP), the User Info Profile Service, and the application – are installed on the same GAS. Otherwise, you must specify the exact callback URL where the OpenIDConnectServiceProvider is set to receive a notification from the GIP.
      Important: Input the redirect URL for a new app only. There is no redirect URL for a new service app.
   d) If you selected New app, the interface includes panels where you can select authorization scopes, required scopes, and optional scopes.
      When you select an authorization scope, any user with that authorization scope will be able to start the application. You can select multiple authorization scopes.
      Required scopes are those scopes that the user must have to successfully use the application.
      Optional scopes are those scopes that a user would need to use specific parts of the application, however a user without those scopes can still be successful using parts of the application.
   e) If you selected New service to service app, the interface includes a panel where you can select scopes.
      For a service to service app, you do not specify Authorization scopes, as these apps are not run by a user. You can only select scopes; you can select multiple scopes.
   f) Click Create.
      The app is registered.

2. Create your app XCF using the appropriate delegate service for an application as described here: Quick start: Set up OpenID Connect in the GAS on page 142.
The application configuration file should resemble this example:

```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/3.20/cfextwa.xsd">
<EXECUTION>
  <PATH>${res.deployment.path}</PATH>
  <MODULE>HelloWorld.42r</MODULE>
  <DELEGATE service="services/OpenIDConnectServiceProvider">
    <IDP>localhost</IDP>
    <CLIENT_PUBLIC_ID>C4805A1E-1766-4945-BEC4-E54F115D9E24</CLIENT_PUBLIC_ID>
    <CLIENT_SECRET_ID>rv6sQN/DjNfGYcE+4wq+VVVAFwoj8u7i</CLIENT_SECRET_ID>
  </DELEGATE>
</EXECUTION>
</APPLICATION>
```

The `<PATH>` entry must point to the executable of your app, you will likely have to modify this entry.

The `OpenIDConnectServiceProvider` delegation protects applications.

The `<IDP>` entry of `localhost` is valid when everything runs on the same Genero Application Server. Otherwise, you must provide the GIP URL. For example: `http://othermachine.com:6394/ws/r/services/GeneroIdentityProvider`

3. Install the application configuration file (.xcf) on the GAS without the deploymentApp, in a dedicated GAS group or in the default app group.

   For instructions on configuring applications and adding them to the GAS without the use of the deployment portal, read the topics in the section Configuring applications on GAS on page 111.

4. Access you application; it is protected by the GIP.

   To determine the URL needed to access your application, review the topic URI Examples on page 48.

**Related tasks**

Deploy and secure an application using the Genero Deployment App on page 237

Deploy and secure an application using the Genero Deployment App. Start with the Genero Archive (.gar) for the application.

**Deploy a Web service using the Genero Deployment App**

Deploy and secure an Web service using the Genero Deployment App. Start with the Genero Archive (.gar) for the Web Service.

**Before you begin**

- Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.
- You must have packaged your applications and services into a Genero Archive.

**Note:** When you create a Genero Archive (.gar) with `fglgar gar` using the --service `wsname.42m` option, a Web service configuration (.xcf) file is automatically generated from the app. If you have an existing Web service configuration (.xcf) that you would rather use, you must create a MANIFEST file pointing to your Web service configuration file.

For more information about packaging a Genero Archive, see Understanding packaging with GAS on page 271.
The REST Web service, in its source code, should already use the \texttt{WSScope} attribute to define the scopes that will provide security globally for the service (on the \texttt{WSInfo} record) or for its functions. You should have a list of such \texttt{WSScope} entries.

\textbf{About this task}

When you deploy the contents of a Genero Archive using the Genero Deployment App, the registration of the Web service is managed for you. Unlike the Genero application, however, the deployment does not include the generation of a \texttt{CLIENT_ID} or \texttt{SERVER_ID} for the Web service.

1. Start the Genero Deployment App as an authorized user.
   The Deployment App application is located at \url{http://host:port/gas/ua/r/admin/DeploymentApp}.
2. Deploy the \texttt{.gar} file.
   a) Click \texttt{Deploy > Genero Archives...}.
      The \texttt{Manage Genero Archives} form displays.
   b) Click \texttt{Deploy}.
   c) Use the \texttt{Select your Genero Archive to deploy} dialog to select your Genero Archive (\texttt{.gar}).
      The application deploys and is listed under \texttt{Genero Archive(s)}.
3. Secure the Genero Archive.
   a) Select the row containing the deployed Genero Archive.
   b) Click \texttt{Secure}.
      All services contained by the archive are secured.
4. Change the status of the deployed Genero Archive to Active.
   a) Select the row containing the deployed Genero Archive.
   b) Click \texttt{Enable}.
      The status changes from Inactive to Active.
5. If the same Genero Archive (\texttt{.gar}) must be deployed on several GAS, repeat the procedure.
6. If not already registered, register the Web service scopes in the GIP using the Genero Console App.
   Instructions are provided here, or you can see Manage Web service access scopes on page 234.
   a) Open the Genero Console App.
      See Managing GIP components on page 230.
   b) Select \texttt{Security > Services}.
      The \texttt{Service access list} form appears.
   c) Click \texttt{Create}.
      A new, empty row is added to the service access list, with the cursor in the \texttt{Service name} field.
   d) Enter a name in the \texttt{Service name} field, and a description in the \texttt{Description} field.
      The name entered can be any name you wish.
      When you have finished, click \texttt{OK}.
   e) With the new service highlighted in the list, click \texttt{Select}.
      The \texttt{Scope list} form displays for the new service. By default, a single scope was created using the name of the service.
   f) Add any additional scopes required by the service.
      It is the \texttt{WSScope} entries in the Genero source that determine what additional scopes need to be entered.
      To add a scope, click \texttt{Create} and enter the information requested.
   g) When you have finished entering all necessary scopes, click \texttt{Back} until you return to the front page of the Genero Console App.

\textbf{Related tasks}

Deploy an application NOT using the Genero Deployment App on page 238
Deploy and secure an application without using the Genero Deployment App.

**Deploy a Web service NOT using the Genero Deployment App**

Deploy and secure an Web service without using the Genero Deployment App.

**Before you begin**

The REST Web service, in its source code, should already use the `WSScope` attribute to define the scopes that will provide security globally for the service (on the `WSInfo` record) or for its functions. You should have a list of such `WSScope` entries.

For more information about `WSScope` and how it is implemented in a Genero REST Web service, see the `WSScope` topic in the *Genero Business Development Language User Guide*.

**About this task**

When you deploy and secure a Web service without using the Genero Deployment App, you must first register the service using the Genero Console App, then add the configuration information required by the Genero Application Server to start the application using the security provided by the Genero Identity Provider.

1. If not already registered, register the Web service scopes in the GIP using the Genero Console App.
   
   Instructions are provided here, or you can see Manage Web service access scopes on page 234.
   
   a) Open the Genero Console App.
      
      See Managing GIP components on page 230.
   
   b) Select Security > Services.
      
      The Service access list form appears.
   
   c) Click Create.
      
      A new, empty row is added to the service access list, with the cursor in the Service name field.
   
   d) Enter a name in the Service name field, and a description in the Description field.
      
      The name entered can be any name you wish.
      
      When you have finished, click OK.
   
   e) With the new service highlighted in the list, click Select.
      
      The Scope list form displays for the new service. By default, a single scope was created using the name of the service.
   
   f) Add any additional scopes required by the service.
      
      It is the `WSScope` entries in the Genero source that determine what additional scopes need to be entered.
      
      To add a scope, click Create and enter the information requested.
   
   g) When you have finished entering all necessary scopes, click Back until you return to the front page of the Genero Console App.

2. Create your app XCF using the appropriate delegate service for an application as described here: Quick start: Set up OpenID Connect in the GAS on page 142.

   The application configuration file should resemble this example:

   ```xml
   <?xml version="1.0" encoding="UTF-8" ?>
   <APPLICATION Parent="ws.default">
   <EXECUTION>
      <PATH>$(res.deployment.path)</PATH>
      <MODULE>MyService.42r</MODULE>
      <DELEGATE service="services/GeneroAccessService">
      <IDP=localhost</IDP>
      </DELEGATE>
   </EXECUTION>
   </APPLICATION>
   ```

   The `<PATH>` entry must point to the executable of your app, you will likely have to modify this entry.
   
   The GeneroAccessService delegation protects Web service applications.
The <IDP> entry of localhost is valid when everything runs on the same Genero Application Server. Otherwise, you must provide the GIP URL. For example: http://othermachine.com:6394/ws/r/services/GeneroIdentityProvider

3. Install the application configuration file (.xcf) on the GAS without the deploymentApp, in a dedicated GAS group or in the default app group.
   For instructions on configuring applications and adding them to the GAS without the use of the deployment portal, read the topics in the section Configuring applications on GAS on page 111.

4. Access your application; it is protected by the IdP.
   It will require a valid access Token with a scope set in WSScope, otherwise you will get an access denied.

Automatize application deployment via scripts

To automatize application deployment via scripts, you must use the Genero Identity Provider (GIP) GetToken and deploy command line tools.

About this task

This procedure is intended to register applications that have to be executed not on behalf of a user, but as a scripting tool. For example, a batch app that gathers some monitoring data from several services.

You need first to register your scripting application using the ConsoleApp (the New service to service app menu option) in order to generate a CLIENT_ID and SECRET_ID for the application.

1. Register a new service application in GIP via ConsoleApp, and select deployment and register scope.
   a) Open the ConsoleApp.
      See Managing GIP components on page 230.
   b) Select Applications > New service to service app.
      The Registering a new service to service application form appears.
   c) In the Info section of the form, complete the fields.
      The Name must match the name of the application configuration file.
      The Client ID and Secret ID are generated for you. They will be required later to access that service app.
   d) In the Scopes section of the form, select the register and deployment scopes.
      Note: register and deployment are scopes needed to be able to get tokens and deploy using the console or script.
   e) Click Create.

2. Use a script to get an access token for the deployment and register service.
   Note: The gettoken tool is located in the $FGLDIR/web_utilities/services/gip/bin/gettoken directory.

   Run this command:
   GetToken client_credentials --idp base_URL_of_idp --savetofile token.json --client_id client_id --secret_id secret_id deployment register

   where:
   • base_URL_of_idp is the base URL of the Genero Identity Provider:
     https://host:port/gas/ws/r/services/GeneroIdentityProvider
   • client_id is the Client ID generated in the previous step.
   • secret_id is the Secret ID generated in the previous step.

   This command saves the needed access token in a file named token.json. The token will be valid for a period of 10 minutes. After ten minutes, you will need to query for a new access token.

3. Use a script to deploy a Genero Archive (.gar) using the generated access token file.
**Note:** The DeployGar tool is located in your $FGLDIR/web_utilities/services/gip/bin/deploy directory.

Run the command:

```
DeployGar deploy --tokenfile token.json myGarfile.gar GAS_base_URL
```

Where:

a. The `--tokenfile` gives the path to a file name where the access token got from the IdP is saved.
b. The Genero archive file name is specified.
c. `GAS_base_URL` is the URL for the Genero Application Server (GAS) where you want to deploy the application.

```
https://host:port/gas
```

This command deploys the Genero Archive and returns the list of deployed Genero Archive (.gar) files on the specified GAS.

**Related concepts**

- **GetToken** on page 251
  The GetToken program implements a set of command line options for getting access tokens.

- **DeployGar** on page 253
  The DeployGar program implements a set of command line options for working with Genero archive (.gar) files on a GAS secured by a Genero Identity Provider.

- **DeployGbc** on page 256
  The DeployGbc program implements a set of command line options for working with Genero Browser Client (GBC) files on a GAS secured by a Genero Identity Provider.

**Automatize application configuration via scripts**

Management of application scopes via scripts uses the Genero Identity Provider (GIP) GetToken and deploy command line tools.

**About this task**

This procedure allows you to manage scopes on a deployed application on a remote GAS via the console or a script. For example, you need to update the scopes for users accessing an application. For more information on managing scopes, see Manage authorization scopes on page 234.

**Note:** This procedure does not apply to Web services. There the scopes are provided in the code via the WSScope attribute and can not be changed without recompiling the service application. For more information about WSScope and how it is implemented in a Genero REST Web service, see the WSScope topic in the Genero Business Development Language User Guide.

**Before you begin**

- If not already done, you need to register a scripting application using the ConsoleApp (the New service to service app menu option) in order to generate a CLIENT_ID and SECRET_ID for the application. See step 1 on page 243.
- You will also need to have the CLIENT_ID of the secure application you are configuring. You can get this using the ConsoleApp (the Applications > Manage apps menu). For details, see Manage applications on page 232.

1. Register a new service application in GIP via ConsoleApp, and select deployment and register scope.
   a) Open the ConsoleApp.
      See Managing GIP components on page 230.
   b) Select Applications > New service to service app.
      The Registering a new service to service application form appears.
   c) In the Info section of the form, complete the fields.
The **Name** must match the name of the application configuration file. The **Client ID** and **Secret ID** are generated for you. They will be required later to access that service app.

d) **In the** **Scopes** **section** of the form, select the **register** and **deployment** scopes.

**Note:** register and deployment are scopes needed to be able to get tokens and deploy using the console or script.
e) Click **Create**.

2. **Use a script to get an access token for the deployment and register service.**

**Note:** The `gettoken` tool is located in the `$FGLDIR/web_utilities/services/gip/bin/gettoken` directory.

Run this command:

```
GetToken client_credentials --idp base_URL_of_idp --savetofile token.json --
client_id client_id --secret_id secret_id deployment register
```

where:

- `base_URL_of_idp` is the base URL of the Genero Identity Provider:

  ```
  https://host:port/gas/ws/r/services/GeneroIdentityProvider
  ```

- `client_id` is the Client ID generated in the previous step.
- `secret_id` is the Secret ID generated in the previous step.

This command saves the needed access token in a file named `token.json`. The token will be valid for a period of 10 minutes. After ten minutes, you will need to query for a new access token.

3. **Use a script to get the configuration of the deployed app.**

**Note:** The `DeployGar` tool is located in your `$FGLDIR/web_utilities/services/gip/bin/deploy` directory.

Run the command:

```
DeployGar config -f token.json -c config.json get app_client_id GAS_base_URL
```

where:

- The `-f` option (short option for `--tokenfile`) gives the path to a file name where the access token got from the IdP is saved.
- The `-c` option (short option for `--config`) needs a name for the configuration file that is created.
- The `get` option takes the client id of the application you are configuring. For details see the prerequisites section.
- `GAS_base_URL` is the URL of the Genero Application Server (GAS)

  ```
  https://host:port/gas
  ```

The sample shows the output saved to the configuration file, `config.json`.

```
"client_id": "3410B13A-06EB-4B13-A4F9-FB78B180D9BC",
"client_secret": "jnImCC+Qq/6P/EdMtudv+/LRGwgB5tqs",
"grant_types": ["authorization_code"],
"redirect_uris": ["http://helios:6394/ws/r/services/OpenIDConnectServiceProvider/oauth2callback"],
"client_name": "edit",
"client_description": "Dynamic registered edit application",
"scope": "Role.Admin openid"
```

4. **Use a script to update the scope in the configuration of the deployed app.**
a) Edit the configuration file (config.json in the example)
The file is in the JSON format. You can open it with a text editor and edit the line with "scopes". For example, you can add a new scope, making sure to follow the JSON format:

```
"scope": "Role.Admin openid deployment"
```
b) When finished, save your changes.
c) Run this command to upload your changes:

```
DeployGar config -f token.json -c config.json set app_client_id GAS_base_URL
```
Where the set option is used to set your configuration changes.

**What to do next:**

Test the application functions as expected. Check if users from the added scope ("deployment" in our example) can access the application.

**Related concepts**

- **Manage authorization scopes** on page 234
  Authorization scopes determine application access.
- **GetToken** on page 251
  The GetToken program implements a set of command line options for getting access tokens.
- **DeployGar** on page 253
  The DeployGar program implements a set of command line options for working with Genero archive (.gar) files on a GAS secured by a Genero Identity Provider.
- **DeployGbc** on page 256
  The DeployGbc program implements a set of command line options for working with Genero Browser Client (GBC) files on a GAS secured by a Genero Identity Provider.

**Sharing files**

The Genero Identity Provider shared file microservice can be used for sharing files. It is possible for registered users to share files.

The SharedFileApp provides you with the interface for pushing and sharing files with other users.

**Share files using the Genero SharedFile App**

The Genero SharedFile App offers a secure way of managing file sharing with other registered users.

**Enable the GIP**

Before you can use the Genero SharedFile App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. For further details, see **Setting up the Genero Identity Provider** on page 215.

**Provide the permissions to the users**

To use the SharedFile app, a user must have the following permissions (scopes):

- **profile.query** scope (for the Profile service). This allows users sharing files to select users.
- **profile** scope (for the Profile service). This allows users to provide their profile information, making it visible to other users.
- **profile.me** scope (for the Profile service). This allows users to log in to the ConsoleApp and update their personal profile (Users > Update me menu).

See **Manage users** on page 230 for details of managing user scopes (permissions).
Create the SharedFile profiles for the users

Even if a registered GIP user has the correct permissions to access the SharedFile app, each user must create a profile within the SharedFile app. This is in addition to being registered within the GIP. When a user first accesses the SharedFile app, they will be asked to create a profile. After creating a profile, they can then use the application. They can only share files with other users who have profiles within the SharedFile app.

At a minimum, users need to have created a profile with at least their First name, Last name, and Email before sharing files with users.

Steps

1. Start the Genero SharedFile App as an authorized user.

2. Upload the file.
   - a) Click Upload.
   - b) Use the Select your file to upload dialog to select a file from your system.
   - The file is uploaded to a directory in your $(home)/.genero-sso/.shared directory. The file is listed in the My Files panel.

3. Allow users access to the file.
   - a) Select the Access rights panel on the right.
   - b) Click New access.
   - The Select users page appears.
   - To select multiple users, hold the CTRL key as you select.
   - d) With the users selected, click OK to close the page and return to the SharedFileApp page.
   - The Access rights panel on the right displays the users to whom you have selected for access.

4. Share the file URL
   - a) Select the file from the My files panel.
   - The Public URL button enables.
   - b) Click Public URL.
   - c) Use the Public URL dialog to copy the URL
   - The URL can be sent to users by email.

5. The receiving user can use the file URL to access the file.
   - As the user receiving the URL, you have two options for accessing the file:
     - Log in to the SharedFile app and access the file by entering the URL in the Get Shared dialog.
     - Use the GetToken tool to retrieve the access token, then add the token to the shared file URL using the query string access_token=value:

```
http://host:port/gas/.../file/share/fileref?access_token=token_xxx
```

GIP Troubleshooting

Examining the Genero Identity Provider logs for error messages, will help resolve issues. You can also report issues to support.

Issues with the GIP are logged in the GAS appdata/log directory. The IdentityProviderService.log is the entry point for troubleshooting GIP services.
GIP log files

When you run an application, the Genero Identity Provider (GIP) writes messages to log files. These log files can assist in troubleshooting GIP issues.

**Important:** Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

Each GIP component writes its messages to its own log file. Requests for services can involve multiple components, and the messages may be logged in one or more logs.

You can access log files in the `appdata/log` directory, where `appdata` refers to GAS application data directory. To learn about the `appdata` directory (to include where to find it), see GAS installation and application data directories on page 40.

The log files include:

- **AccessService.log**  
  Logs messages generated by the GIP ConsoleApp or DeploymentApp.

- **DeploymentService.log**  
  Logs messages generated by the deployment service.

- **IdentityProviderService.log**  
  Logs messages generated by the Genero identity provider service. This includes requests for authentication (SSO), and authorization (user permissions to open apps).

- **OIDC.log**  
  Logs messages generated by requests for the OpenID Connect delegation service.

- **ProfileService.log**  
  Logs messages generated by the profile service.

- **RegisterService.log**  
  Logs messages generated by the registration service.

**Related concepts**

- Troubleshoot GIP errors in logs on page 247
- Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

  - **IdP errors** on page 248
    - Describes some common Genero Identity Provider (GIP) errors and how to resolve them.

  - **Access service errors** on page 249
    - Describes some common GIP access errors and how to resolve them.

**Troubleshoot GIP errors in logs**

Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

You need to have access to the GIP logs:

1. Check the GIP logs for standard errors or error messages.
   - It is recommended that you check your `IdentityProviderService.log` file first as it is the entry point for most IDP services. If there is any indication that there is an error, it will appear here. For examples of how to resolve some known errors. See **IdP errors** on page 248, **Access service errors** on page 249, and **OIDC service errors** on page 249.

2. Check for HTTP and HTTPS errors. See HTTP errors in GAS logs and HTTPS errors on page 250.
   - HTTP response codes indicate the success or failure of an API request.

3. If nothing unusual is found, contact your local Four Js support center for assistance. If requested to send the logs, copy all the logs you have to a zip file and email it to your support contact. Typically these logs are needed:
   - `appdata/log/*.log`, where `appdata` refers to GAS application data directory.
• Related dispatcher daily logs (dispatcher, proxy and vm) for the date the issue arose.

**Related concepts**

**GIP log files** on page 247

When you run an application, the Genero Identity Provider (GIP) writes messages to log files. These log files can assist in troubleshooting GIP issues.

**IdP errors**

Describes some common Genero Identity Provider (GIP) errors and how to resolve them.

**Table 23: Errors when trying to connect to GIP ConsoleApp or DeploymentApp**

<table>
<thead>
<tr>
<th>Error type</th>
<th>IdentityProviderService.log message</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWS error</td>
<td>ERROR : pid - [Server] &quot;Request&quot; -15575</td>
<td>Gas disconnected the Web service program because an invalid or unsupported HTTP request was sent.</td>
</tr>
<tr>
<td>Endpoint error</td>
<td>ERROR : pid - [Server] &quot;CheckIdPConsistency&quot; Detected idp.xml inconsistency between endpoint : '<a href="http://x">http://x</a>' and request : '<a href="http://y">http://y</a>'</td>
<td>The URL in the request is incorrect. For instance, if the URL in the request to the ConsoleApp is not the same as the one registered by the StarterApp in the idp.xml file, this error is thrown.</td>
</tr>
<tr>
<td>Authentication error</td>
<td>[AuthService] &quot;DoLogin&quot; Error sent: no application matching gid, gid=X</td>
<td>Incorrect application name.</td>
</tr>
<tr>
<td>User scope error</td>
<td>ACCESS : pid - [AuthService] &quot;CheckUserAuthorizationAndScopes&quot; Authorization denied to user=1 for access to application=2 due to missing required scope=10</td>
<td>Check the user scopes in the ConsoleApp. See <strong>Manage users</strong> on page 230.</td>
</tr>
</tbody>
</table>

**Related concepts**

**Troubleshoot GIP errors in logs** on page 247
Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

Access service errors

Describes some common GIP access errors and how to resolve them.

Table 24: Errors using GIP ConsoleApp or DeploymentApp

<table>
<thead>
<tr>
<th>Error type</th>
<th>AccessService log message</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>401 unauthorized error</td>
<td>ACCESS : pid - [AccessService] &quot;DoDelegate&quot; Invalid grant, no access token found</td>
<td>No valid token provided. Apache server discards the authorization header if it is not a base64 encoded user/password combination. To resolve, check your Apache configuration for ReWriteRule. See the &quot;Apache users&quot; section in the Setting up the Genero Identity Provider on page 215 topic.</td>
</tr>
<tr>
<td>Invalid delegate</td>
<td>ERROR : pid - [AccessService] &quot;DoDelegate&quot; invalid delegate</td>
<td>This is probably due to a misuse of the delegate feature in the GeneroAccessService.xcf. Check the access log file (AccessService.log) for the exact cause of the problem.</td>
</tr>
</tbody>
</table>

Related concepts

Troubleshoot GIP errors in logs on page 247

Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

OIDC service errors

You may receive an error when using the Console App or Deployment App. This topic lists the most common OpenID Connect errors and how you can resolve them.

Table 25: Errors using GIP ConsoleApp or DeploymentApp

<table>
<thead>
<tr>
<th>Error type</th>
<th>OIDC log message</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration parameter errors</td>
<td>ERROR : pid - [Discovery] &quot;GetOpenIDConfiguration&quot; ERROR :-15553</td>
<td>OpenID configuration errors arise when the OpenID service:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Does not exist at the endpoint provided</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Your request has invalid or missing parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you are using HTTPS, see HTTPS errors on page 250.</td>
</tr>
<tr>
<td>Metadata error</td>
<td>SQLERR : pid - [IdPManager] &quot;GetIdPIdFromEntityID&quot; could not retrieve metadata</td>
<td>(See above)</td>
</tr>
</tbody>
</table>
### Error type | OIDC log message | Resolution
--- | --- | ---
Provider error | ERROR: pid - [SPManager] "StartAuthentication" Issuer is missing | (See above) 
Permissions error | Program error at 'RelayState.4gl', line number 95. SQL statement error number -6372 (-8). attempt to write a readonly database | This error is triggered when FGLGWS is installed in the C: \Program Files path. Grant user write permissions on $FGLDIR \web_utilities\services \openid-connect directory.
License error | Program error at 'Server.4gl', line number 77. FORMS statement error number -6016. Cannot get information for license (Error 00K000). Check your environment and the license (run 'fglWrt/ greWrt/grxWrt -a info') | Check your GAS environment, and run fglWrt/greWrt/grxWrt -a info to ensure your license is valid.

### Related concepts
Troubleshoot GIP errors in logs on page 247
Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

### HTTPS errors
This topic lists the most common HTTPS errors and how you can resolve them.

#### Table 26: Errors using HTTP and HTTPS

<table>
<thead>
<tr>
<th>Error type</th>
<th>Browser error</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>License error</td>
<td>Application or service has been stopped due to fatal error</td>
<td>Check your GAS environment, and run the fglWrt/grxWrt --a info to ensure your license is valid. Check also your OICD.log. See OICD service errors.</td>
</tr>
<tr>
<td>Access forbidden</td>
<td>HTTP Error 403</td>
<td>See ACCESS_CONTROL on page 367</td>
</tr>
<tr>
<td>Error type</td>
<td>Browser error</td>
<td>Resolution</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Certificate issuer errors</td>
<td>Genero OpenIDConnect Error page is shown in the browser. The error may be described as either:</td>
<td>A certificate authority (CA) to verify the certificate is missing. Contact the certificate issuer and ask them to provide the CA certificate.</td>
</tr>
<tr>
<td></td>
<td>• Invalid configuration parameters (The certificate issuer is unknown) or</td>
<td>Important: If CA certificates are stored in $FGLDIR/web_utilities/certs directory, ensure they are named with the .crt extension.</td>
</tr>
<tr>
<td></td>
<td>• Protocol error (the certificate is not in your CA list).</td>
<td>Other alternatives for CA configuration:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• You can configure the CA using FGLPROFILE entries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If on Windows® or macOS™, you also have the option of importing the CA into the operating system keystore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the Certificates in practice section in the Genero Business Development Language User Guide.</td>
</tr>
</tbody>
</table>

Related concepts

Troubleshoot GIP errors in logs on page 247
Problems can be caused by issues as diverse as service request errors, Web server errors, GAS errors, incorrect configurations, invalid tokens, or other issues. Use the GIP logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

Troubleshooting HTTP errors on page 121
Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

GIP Reference

These topics are the reference guides for Genero Identity Provider.

Tools and Commands

Command line tools provided for GIP.

GetToken
The GetToken program implements a set of command line options for getting access tokens.

Syntax

```
GetToken [client_credentials | password] [options] scopes
```

1. There are two commands:
   a. `client_credentials` gets an access token using the service application client id and secret id.
   b. `password` gets an access token using the GIP user's username and password.
2. `options` are described in the tables.
3. *scopes* are the list of scopes required in the access token.

**Table 27: GetToken options specific to the `client_credentials` command**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h or --help</td>
<td>Displays help for the <code>client_credentials</code> command.</td>
</tr>
<tr>
<td>-c, --client_id code</td>
<td>Specifies the service application client id.</td>
</tr>
<tr>
<td>-s, --secret_id code</td>
<td>Specifies the service application secret id.</td>
</tr>
<tr>
<td>-f, --savetofile filename</td>
<td>Save access token to file.</td>
</tr>
<tr>
<td>-i, --idp URL</td>
<td>URL of IdP</td>
</tr>
</tbody>
</table>

**Table 28: GetToken options specific to the `password` command**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h or --help</td>
<td>Displays help for the <code>password</code> command.</td>
</tr>
<tr>
<td>-u, --username username</td>
<td>Specifies the name of the IdP user.</td>
</tr>
<tr>
<td>-p, --password password</td>
<td>Specifies the password of the IdP user</td>
</tr>
<tr>
<td>-f, --savetofile filename</td>
<td>Save access token to file.</td>
</tr>
<tr>
<td>-i, --idp URL</td>
<td>URL of IdP</td>
</tr>
</tbody>
</table>

**Usage**

**Note:** The `gettoken` tool is located in the `$FGLDIR/web_utilities/services/gip/bin/gettoken` directory.

The `GetToken` tool provides a way to get access tokens at the command line. For instance, you can use this tool to work with applications or services secured by the GIP via a script. See Automatize application deployment via scripts on page 242.

**Get an access token for a Web service secured by the GIP**

For example, you use the following command if you need to get an access token for a Web service secured by the GIP (line breaks have been added to the command example to improve readability):

```
GetToken password -u user -p mypw
--idp https://host:port/gas/ws/r/services/GeneroIdentityProvider
--savetofile mytoken.json myWSScope
```

Where:

1. The `password` command is used.
   **Note:** Web services have no credentials (client or secret ids) and with this command you can access the IdP with a user’s login details to get its token.
2. The IdP user’s name and password is provided in `-u user -p mypw`.
3. The URL of the Genero Identity Provider is specified.
4. `mytoken.json` is the file name where the access token is saved.
5. `myWSScope` is the scope of the service required in the token.

**Note:** **Access token generated is valid for a limited time:**

The access token got via this tool is valid for 10 minutes only. An access token got from the GIP when starting an application is valid for one hour and continues to be refreshed automatically till the application is closed.

Use the commands `password` and `client_credentials` with the help option to display available options. For example:

```
GetToken client_credentials --help
```

**Related concepts**

**Deploying and securing applications and Web services** on page 236

You have two methods for deploying and securing applications and Web services.

**DeployGar**

The DeployGar program implements a set of command line options for working with Genero archive (.gar) files on a GAS secured by a Genero Identity Provider.

**Syntax**

```
DeployGar command \ options \ args
```

1. **command.** There are nine commands to perform actions on Genero archive (.gar) files on the GAS:
   a. **list** lists all deployed gar files.
   b. **deploy** deploys a gar file.
   c. **undeploy** undeploys a gar file.
   d. **enable** enables a gar file.
   e. **disable** disables a gar file.
   f. **secure** secures a gar file.
   g. **unsecure** unsecures a gar file.
   h. **delegate** configures delegation in a deployed Genero archive.
   i. **config** configures application security on the Identity Provider (IdP).

2. **options.** Some options are specific to commands and these are described in the tables in **DeployGar command options** on page 254.

3. **args** include the URL of the GAS. Some arguments are specific to commands and these are described in the next paragraphs.

**Syntax 1: list archives**

```
DeployGar list \ options \ url
```

1. **options** are described in **Table 29: DeployGar options specific to the list/deploy/enable/secure commands** on page 254

2. **url** is the GAS base URL

**Syntax 2: deploy/enable/secure archives**

```
DeployGar \ deploy \ undeploy \ enable \ disable \ secure \ unsecure \ options \ archive \ url
```

1. **options** are described in **Table 29: DeployGar options specific to the list/deploy/enable/secure commands** on page 254
2. *archive* is the Genero archive file name
3. *url* is the GAS base URL

**Syntax 3: delegate**

```plaintext
DeployGar delegate [options] archive xcf url
```

1. *options* are described in Table 30: DeployGar options specific to the delegate command on page 254
2. *archive* is the Genero archive file name
3. *xcf* is the file name of the application configuration file (*.xcf) in the archive
4. *url* is the GAS base URL

**Syntax 4: config**

```plaintext
DeployGar config [options] operation client_id url
```

1. *options* are described in Table 31: DeployGar options specific to the config command on page 255
2. *operation* to perform security configuration on IdP must be one of the following:

   - get
   - set

3. *client_id* is the OAuth client ID of the application
4. *url* is the GAS base URL

**DeployGar command options**

**Table 29: DeployGar options specific to the list/deploy/enable/secure commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help for the command.</td>
</tr>
<tr>
<td>-u, --username username</td>
<td>Specifies the HTTP user name.</td>
</tr>
<tr>
<td>-p, --password password</td>
<td>Specifies the password of the HTTP user.</td>
</tr>
<tr>
<td>-t, --token value</td>
<td>Specifies the access token value.</td>
</tr>
<tr>
<td>-f, --tokenfile filename</td>
<td>Specifies the access token file.</td>
</tr>
<tr>
<td>-x, --xml</td>
<td>Output result of command in XML format.</td>
</tr>
</tbody>
</table>

**Table 30: DeployGar options specific to the delegate command**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Displays help for the command.</td>
</tr>
<tr>
<td>-u, --username username</td>
<td>Specifies the HTTP user name.</td>
</tr>
<tr>
<td>-p, --password password</td>
<td>Specifies the password of the HTTP user.</td>
</tr>
<tr>
<td>-t, --token value</td>
<td>Specifies the access token value.</td>
</tr>
<tr>
<td>-f, --tokenfile filename</td>
<td>Specifies the access token file.</td>
</tr>
</tbody>
</table>
Table 31: DeployGar options specific to the `config` command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h --help</code></td>
<td>Displays help for the command.</td>
</tr>
<tr>
<td><code>-u, --username username</code></td>
<td>Specifies the HTTP user name.</td>
</tr>
<tr>
<td><code>-p, --password password</code></td>
<td>Specifies the password of the HTTP user.</td>
</tr>
<tr>
<td><code>-t, --token value</code></td>
<td>Specifies the access token value.</td>
</tr>
<tr>
<td><code>-f, --tokenfile filename</code></td>
<td>Specifies the access token file.</td>
</tr>
<tr>
<td><code>-c, --config filename</code></td>
<td>Specifies the configuration file name.</td>
</tr>
</tbody>
</table>

Usage

**Note**: The DeployGar tool is located in your `$FGLDIR/web_utilities/services/gip/bin/deploy` directory.

The DeployGar tool allows you to manage Genero archive (.gar) files at the command line. For instance, you can use this tool to deploy and configure secure applications or services on a remote GAS or using a script.

Use a command with the help option to display available options. For example:

`DeployGar config --help`

Some examples are given in the next paragraphs using DeployGar.

**Get list of deployed gar files**

Run this command to get a list of deployed gar files on a GAS. Mainly, you would use this to check your access is working; that you have an up-to-date access token, and that the GAS URL is reachable.

`DeployGar list --xml --tokenfile mytoken.json GAS_base_URL`

Where:

1. The `--xml` option is specified for the format of the output.
2. The `--tokenfile` option needs the path to a file name where the access token got from the IdP is saved. See `GetToken` on page 251 for details of getting the access token.
3. `GAS_base_URL` is the URL for the Genero Application Server (GAS) where you want to see a list of deployed applications.

   ```
   https://host:port/gas
   ```

**Deploy a gar file**

Run this command to deploy a Genero archive file.

`DeployGar deploy -f mytoken.json myGarfile.gar GAS_base_URL`

Where:

1. The `--tokenfile` option gives the path to a file name where the access token got from the IdP is saved.
2. `GAS_base_URL` is the URL for the Genero Application Server (GAS) where you want to deploy the application.

   ```
   https://host:port/gas
   ```

For more examples, see Automatize application deployment via scripts on page 242 and Automatize application configuration via scripts on page 243.
Related concepts

Deploying and securing applications and Web services on page 236
You have two methods for deploying and securing applications and Web services.

DeployGbc
The DeployGbc program implements a set of command line options for working with Genero Browser Client (GBC) files on a GAS secured by a Genero Identity Provider.

Syntax

```
DeployGbc command \ options \ args
```

1. **command**. There are six commands to perform actions on GBC on the GAS:
   a. **list** lists all GBC.
   b. **deploy** deploys a GBC.
   c. **undeploy** undeploys a GBC.
   d. **default** sets GBC as default.
   e. **reset** resets GBC to the default in the FGLGWS installation.
   f. **rename** renames a GBC.
2. **options** are described in Table 32: DeployGbc options on page 256.
3. **args** include the URL of the GAS. Some arguments are specific to commands and these are described in the next paragraphs.

Syntax 1: list/reset GBC

```
DeployGbc list reset \ options url
```

1. **options** are described in Table 32: DeployGbc options on page 256
2. **url** is the GAS base URL

Syntax 2: deploy/undeploy/default/rename GBC

```
DeployGbc deploy undeploy default rename \ options name url
```

1. **options** are described in Table 32: DeployGbc options on page 256
2. **name** is the GBC file name
3. **url** is the GAS base URL

Table 32: DeployGbc options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h --help</td>
<td>Displays help for the command.</td>
</tr>
<tr>
<td>-u, --username username</td>
<td>The HTTP user name.</td>
</tr>
<tr>
<td>-p, --password password</td>
<td>The password of the HTTP user.</td>
</tr>
<tr>
<td>-t, --token value</td>
<td>The access token value.</td>
</tr>
<tr>
<td>-f, --tokenfile filename</td>
<td>The access token file.</td>
</tr>
</tbody>
</table>
### Genero Identity Provider (GIP) 257

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x, --xml</td>
<td>Output result of command in XML format.</td>
</tr>
</tbody>
</table>

### Usage

**Note:** The `DeployGbc` tool is located in your `$FGLDIR/web_utilities/services/gip/bin/deploy` directory.

The `DeployGbc` tool allows you to manage Genero Browser Client files at the command line. For instance, you can use this tool to deploy and configure GBC on a remote GAS or using a script.

Use a command with the help option to display available options. For example:

```
DeployGbc reset --help
```

See the next paragraphs for examples using `DeployGbc`.

### Get list of deployed GBC

Run this command to list the deployed GBC on a GAS. Mainly, you would use this to check your access is working; that you have an up-to-date access token, and that the GAS URL is reachable.

```
DeployGbc list --xml --tokenfile mytoken.json GAS_base_URL
```

Where:

1. The `--xml` option is specified for the format of the output.
2. The `--tokenfile` option needs the path to a file name where the access token got from the IdP is saved. See `GetToken` on page 251 for details of getting the access token.
3. `GAS_base_URL` is the URL for the Genero Application Server (GAS) where you want to see the list of deployed GBC.

```
https://host:port/gas
```

### Deploy a GBC file

Run this command to deploy a GBC on a GAS.

```
DeployGbc deploy -f mytoken.json myGBCProject.zip GAS_base_URL
```

Where:

1. The `-f` option (short option for `--tokenfile`) gives the path to a file name where the access token got from the IdP is saved.
2. The GBC file name is specified.
3. `GAS_base_URL` is the URL for the Genero Application Server (GAS) where you want to deploy the GBC.

```
https://host:port/gas
```

### Related concepts

- Deploy GBC with `gasadmin` on page 260
The `gasadmin` tool allows you to deploy a GBC client from the command line.

**GIP API scopes reference**
Reference information for Genero Identity Platform API scopes.

### Table 33: OpenID and OpenIDRegister API scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
<th>Required by</th>
</tr>
</thead>
</table>
| openid  | Support OpenID-Connect Single sign-on authentication                         | • All apps
|         |                                                                             | • All users                                      |
| register| Access the Genero Identity Platform registration service to register applications, Web service scopes, etc. | • Administrators
|         |                                                                             | • Service-to-Service apps (for example, scripts) |

### Table 34: Authorization API scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
<th>Required by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role.Admin</td>
<td>Administrator of Genero Identity Platform</td>
<td>Administrators</td>
</tr>
<tr>
<td>Role.Supervisor</td>
<td>Supervisor of Genero Identity Platform</td>
<td>Select users</td>
</tr>
</tbody>
</table>
| Role.User | Standard user of Genero Identity Platform        | • All apps
|           |                                                 | • All users         |

### Table 35: Profile API scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
<th>Required by</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile</td>
<td>Provide user profile information to other users</td>
<td>File sharing users</td>
</tr>
<tr>
<td>profile.me</td>
<td>Access to manage a user's own personal profile information</td>
<td>All users</td>
</tr>
<tr>
<td>profile.mgr</td>
<td>Manage any user profile information</td>
<td>Administrators</td>
</tr>
<tr>
<td>profile.query</td>
<td>Get user info by query</td>
<td>File sharing users</td>
</tr>
</tbody>
</table>

### Table 36: Deployment API scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
<th>Required by</th>
</tr>
</thead>
</table>
| deployment | Access the Genero Deployment service            | • Service-to-Service apps (for example, scripts deploying apps)
|         |                                                 | • Users deploying apps                           |
Table 37: SharedFile API scopes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
<th>Required by</th>
</tr>
</thead>
<tbody>
<tr>
<td>sharefile</td>
<td>Access to file share service</td>
<td>File sharing users</td>
</tr>
</tbody>
</table>

Related concepts

Managing GIP components on page 230
The Genero Identity Provider (GIP) can manage users, groups, applications, tokens, authorization scopes, and Web service access scopes. These are managed using the Console App.

Developing Web applications and Web services

You can deliver Web applications and Web services with the Genero Application Server (GAS). Delivery of Web applications also involves the Genero Browser Client (GBC).

Working with Genero Browser Client (GBC)

Understand how to deploy and manage customized GBCs on the Genero Application Server (GAS).

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 Browser Client operates can be helpful in the planning and deploying of Web applications.

For more information, see the Genero Browser Client User Guide.

Related concepts

Front-ends and Extensions on page 16
The Genero Application Server can serve applications using various front-ends and extensions.

Create zip file with customization project

Use this procedure to create a zip file with your Genero Browser Client (GBC) customization project for deployment on the GAS.

As a prerequisite, you have prepared a customization project and you have a compiled version of the GBC front-end. For more information, see the Genero Browser Client User Guide.

1. Open the project_dir/custom.json file using a text editor.
   a) Set the value for mode to prod.
   b) Update the customization variable to reference the specific customization project you wish to deploy.

```json
{
    "compile": {
        "mode": "prod",
        "customization": "customization/myCustomProject"
    }
}
```

2. Use grunt --create-zip to compress the contents of your project_dir/dist/customization/custom_project_dir.

   Note: grunt --create-zip builds your current target (as grunt would do) and zips the contents.

   A zip file is created in project_dir/archive.
What to do next
See Deploy GBC with gasadmin on page 260 or Deploy GBC client with Deployment App on page 261.

Deploy GBC with gasadmin
The gasadmin tool allows you to deploy a GBC client from the command line.

The gasadmin tool does not require the Genero Identity Provider.

Tip: If you have installed the Genero Identity Provider, then you have access to the Deployment App. The Deployment App provides a browser-based interface for deploying and managing your GBC clients. See Deploy GBC client with Deployment App on page 261 for more information.

As a prerequisite, you have a compiled version of the GBC front-end in a zip file. For more information, see the Genero Browser Client User Guide.

Deploy a GBC client
To deploy a GBC, you specify the path and filename:

```
gasadmin gbc --deploy <path+filename>
```

Deploy GBC
This example shows how you can use the gasadmin deploy command to deploy a GBC client on the GAS.

```
gasadmin gbc --deploy c:\fjs\gbc-projects\gbc-1.00.53\archive\custA.zip
```

List deployed GBC clients
To list all deployed GBC clients:

```
gasadmin gbc --list
```

All deployed GBC clients are listed, with the default identified.
In addition, the static GBC included with FGLGWS is identified.

Set a GBC client as the default
To set a deployed GBC client as default:

```
gasadmin gbc --default <name>
```

To retrieve the name, and to verify the setting worked, use the --list option.

Set default GBC
These examples show how you can use gasadmin deploy command to list the deployed GBC clients and set a default client on the GAS.

```
gasadmin gbc --list
```

```
gasadmin gbc --default custB
```
Figure 69: List deployed GBC and set default

Related tasks
Reset GBC client to initial installed version on page 266

The GBC can be reset to the initial installed version delivered in the FGLGWS package using the gasadmin tool.

Deploy GBC client with Deployment App

Use this procedure when you need to deploy a customized Genero Browser Client on your Genero Application Server using the Genero Deployment App.

As a prerequisite, you have a compiled version of the GBC client in a zip file. For more information see the Genero Browser Client User Guide.

The Genero Deployment App provides you with a user interface to the action to deploy implemented by the gasadmin gbc command. The zip file is unpacked in the $(res.gbc.deployment) path, which resolves to the $(res.appdata.path)/gbc_deployment directory.

Tip: As an alternative to the Genero Deployment App, you can deploy the GBC using the gasadmin gbc command with the deploy option. For an example see Deploy GBC on page 356.

1. Open the Genero deployment application.

   http://myhost:6394/ua/r/admin/DeploymentApp

   On a remote machine, you need to specify the appropriate URL for the server and port in your browser, for example:

   http://zeus:8090/ua/r/admin/DeploymentApp

2. Select the Deploy > Genero Browser Clients ... menu.
3. In the Manage Genero Browser Clients screen click the Deploy button.
4. Use the dialog that opens to locate the client you wish to deploy.

   Note: It must be in a zip file.

   You can drag and drop your file into the dialog window. The new client is deployed and displayed in the list.

What to do next
Provide GBC in application path

Describes a method for providing a GBC client in your application path.

Before you begin

You created a customization project and you have a compiled version of the GBC front-end. See the *Genero Browser Client User Guide*.

About this task:

In this task you provide the GBC to use with your app in the `gbc` directory of the application path. Regardless of the GAS configuration, the application will use the GBC inside that directory.

If the `gbc` directory is not set, the GAS then uses the standard `GBC_LOOKUP_PATH` lookup mechanism.

Follow these steps to provide the GBC in the application path.

1. Create an empty directory, named `gbc` in your application root directory.
   
   For example, in your `myapps/special/app1/gbc` directory.

2. Copy your GBC to this directory.

3. In the application configuration file, specify the application path in the `PATH` element.

   For instance:

   ```xml
   <APPLICATION version="1.0" encoding="UTF-8" >
   <EXECUTION>
   <PATH>myapps/special/app1/</PATH>
   <MODULE>myModule.42m</MODULE>
   </EXECUTION>
   </APPLICATION>
   ```

   At application startup, the GAS looks for the GBC in the `gbc` directory.

4. To test, save the configuration file in a GAS application group directory.

   You can save the `xcf` in the default application group, `$(res.appdata.path)/app`, directory, or in your own defined group. Alternatively, you can deploy the app.

5. Test the application opens with the GBC as expected.

   View an application in your browser by starting the standalone dispatcher from the command line. Open the application by entering the URL, for example:

   ```
   http://localhost:6394/ua/r/myapp1
   ```

   Click on the **Application information** menu icon to open the **Product identification** pop up window. Details of the GBC in use are displayed.

   If the application fails to load, see the Fix configuration issues page of the *Genero Browser Client User Guide*.

For production environments, it is recommended to deploy the GBC and set the default client on your GAS installation using one of the following methods:
• The Genero deployment portal. It offers a user interface, and access can be configured for remote servers. See Deploy GBC client with Deployment App on page 261.
• The gasadmin gbc command line tool option. You can use it on a local GAS installation.

Related concepts
Configuring GBC client for applications on page 263
There are different methods for specifying the GBC client your applications use depending on whether you want to use the same client for all applications, or a given client for a specific application.

Configuring GBC client for applications
There are different methods for specifying the GBC client your applications use depending on whether you want to use the same client for all applications, or a given client for a specific application.

GBC client configuration
Table 38: GAS configuration on page 263 describes how to configure the GBC client in the Genero Application Server (GAS) and specify a default.

Table 38: GAS configuration

<table>
<thead>
<tr>
<th>Configuration</th>
<th>File</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBC_LOOKUP_PATH</td>
<td>GAS configuration file</td>
<td>Add the path for your GBC on page 394 directory, for example, &lt;GBC_LOOKUP_PATH&gt;c:/project_dir/dist/ customization;...; $(res.fgldir)/ web_utilities/gbc&lt;/GBC_LOOKUP_PATH&gt;</td>
<td>Required. If the client specified in GBC is not found in the path, the default client in the $(res.fgldir)/ web_utilities/gbc is used.</td>
</tr>
<tr>
<td>Your default GBC client</td>
<td>A file called _default</td>
<td>Create a text file with the name &quot;_default&quot; in a root directory where you have GBC clients, for example, c:/project_dir/dist/ customization/_default. The file contains the name or alias of a directory where the GBC client you want to set as default is found.</td>
<td>Optional. _default is at a path found in the GBC_LOOKUP_PATH. Or if using the dispatcher override, _default is specified in the user path. Can be applied for a single application, or for all applications. Tip: A GAS restart is not required when you update the _default file.</td>
</tr>
</tbody>
</table>

Applying GBC client
Table 39: Apply specific GBC client to use for an application on page 264 summarizes two ways to specify the GBC client for a single application.
Table 39: Apply specific GBC client to use for an application

<table>
<thead>
<tr>
<th>Applied to:</th>
<th>Element/Option</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your application</td>
<td>GBC</td>
<td>Specify the client to use in the GBC element, for example <code>&lt;GBC&gt;&lt;my-gbc&gt;&lt;/GBC&gt;</code>, where <code>my-gbc</code> references either a directory name or <code>_default</code> file found in <code>GBC_LOOKUP_PATH</code>.</td>
<td>See Specify in the application configuration file on page 264.</td>
</tr>
<tr>
<td>configuration file (.xcf)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applied to:</th>
<th>Element/Option</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
</table>

Table 40: Apply GBC client with dispatcher override on page 264 shows how to apply the GBC to use for all applications and override GAS defaults.

Table 40: Apply GBC client with dispatcher override

<table>
<thead>
<tr>
<th>Applied to:</th>
<th>Element/Option</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatcher override</td>
<td><code>-E res.path.gbc.user</code></td>
<td>For example, start the standalone dispatcher with override, <code>httpdispatch -E res.path.gbc.user=C:\my_gbc\1.00.40\dist\customization</code></td>
<td>See Specifying default GBC client for applications on page 265.</td>
</tr>
</tbody>
</table>

Specify in the application URL

This method specifies the GBC client to use in the application URL.

Once you have deployed the GBC (using the `gasadmin gbc` command or the deploymentApp), you can use the URL query string (`?gbc=<my-gbc>`) to specify which GBC client to use for the application when you run it:

```
http://myApplicationServer:6394/ua/r/gwc-demo?gbc=<my-gbc>
```

Specify in the application configuration file

This method uses the `GBC_LOOKUP_PATH` and GBC configuration parameters.

In the GAS configuration file, set `GBC_LOOKUP_PATH` to include the path to the folder that contains your customization directories.

**Tip:** The dispatcher searches the paths specified in order, therefore it is recommended to place the path to your GBC directory first in the path list.

In the application configuration file, set `GBC` to your GBC directory name.

**Related tasks**

Provide GBC in application path on page 262
Describes a method for providing a GBC client in your application path.

**Specifying default GBC client for applications**

A method for specifying a default GBC client for your applications using the _default file is described.

**Before you begin**

You created a customization project and you have a compiled version of the GBC front-end. The project sits in a project directory `project_dir`. See the Genero Browser Client User Guide.

**About this task:**

When testing or in development you just need to specify a default GBC client that all your applications can use without needing to alter or update your application configuration files.

All that is required in this case is that you have a text file called "_default" in your customization root directory, which contains the name of the client you want your applications to use. Then you specify the path to the root directory in one of the following ways:

- In an override when starting the dispatcher. This is the method shown in this task.
- Or you specify it in the `GBC_LOOKUP_PATH` element of the GAS configuration file (`as.xcf`).

Follow these steps to specify the default GBC:

1. Create an empty file, named `_default` (with no extension), in your customization root directory.
   
   For example, the file is located in your `project_dir/dist/customization/` directory.

2. Using a text editor, specify the name of the customization to use.
   
   For example, to use a GBC customization named "red", the file would simply contain the following:

   ```
   red
   ```

   In this example, red is the actual name of your `custom_project_dir` directory located in `project_dir/dist/customization/`.

Test the customized GBC is as expected. View an application in your browser by starting the standalone dispatcher from the command line using the override (`-E`) option, and specifying the path of the customization root directory:

```shell
httpdispatch -E res.path.gbc.user=C:\fjs\gbc\1.00.39\dist\customization
```

Open the application by entering the URL, for example:

http://localhost:6394/ua/r/myapp.

If the application fails to load, see the Genero Browser Client User Guide.

For production environments, it is recommended to deploy the GBC and set the default client on your GAS installation using one of the following methods:

- The Genero deployment portal. It offers a user interface, and access can be configured for remote servers. See Deploy GBC client with Deployment App on page 261.
- The `gasadmin gbc` command line tool option. You can use it on a local GAS installation.

**Related concepts**

Configuring GBC client for applications on page 263

There are different methods for specifying the GBC client your applications use depending on whether you want to use the same client for all applications, or a given client for a specific application.

**Set default GBC client with the Genero Deployment App**

Use this procedure when you need to set the default Genero Browser Client on your Genero Application Server using the Genero Deployment App.

**Before you begin:**
As a prerequisite, you have deployed a GBC client on the GAS. See Deploy GBC client with Deployment App on page 261.

**Steps**

1. Open the Genero deployment application.

   \[
   \text{http://myhost:6394/ua/r/admin/DeploymentApp}
   \]

   On a remote machine, you need to specify the appropriate URL for the server and port in your browser, for example:

   \[
   \text{http://zeus:8090/ua/r/admin/DeploymentApp}
   \]

2. Select the **Deploy > Genero Browser Clients ...** menu.

3. In the Manage Genero Browser Clients screen select the client you wish to set as default in the list and click on the **Set as Default** button.

   The default client is updated and the checkbox beside it is ticked.

Test the customized GBC is as expected. Open an application in a browser tab. Click on the **Application information** menu icon to open the **Product identification** pop up window. Details of the GBC in use are displayed.

If the application fails to load, see the troubleshooting topics in the Genero Browser Client User Guide.

**Related concepts**

Deploying and securing applications and Web services on page 236

You have two methods for deploying and securing applications and Web services.

**Reset GBC client to initial installed version**

The GBC can be reset to the initial installed version delivered in the FGLGWS package using the `gasadmin` tool.

If you have deployed and set a default GBC client on the GAS, you can quickly restore the original installed version by following this procedure.

**Steps**

At the command line enter the `gasadmin gbc --reset` command with the reset option.

\[
\text{gasadmin gbc --reset}
\]

The following message is displayed in the standard output: GBC is reset to initial one.

Test the GBC is as expected. View an application in your browser by starting the standalone dispatcher from the command line. Open the application by entering the URL, for example:

\[
\text{http://localhost:6394/ua/r/myapp}
\]

If the application fails to load, see the Troubleshooting customization configuration issues topic in the Genero Browser Client User Guide.

**What to do next**

Alternatively, you can use the deployment application to reset the GBC client. For more details see Deploy GBC client with Deployment App on page 261.
Working with Genero Web Services

Familiarize yourself with the URLs used to access Web services applications deployed on the GAS and to get their WSDL (SOAP) or OpenAPI specification (REST). Learn how to validate they are working and troubleshoot failures, or configure a "sticky" Web services.

For information on Web service configuration see:

- Configure applications for Web service on page 116
- SERVICE_LIST on page 424

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

Examples of the URIs used to access SOAP and REST Web services.

Access REST Web services

- To get the OpenAPI specification file for a specified service:
  
  http://appserver:6394/ws/r/xcf/resource?openapi.json

- To access a Web service resource:
  

- If the Web service uses a group:
  

- Access through a Web server:
  
  http://webserver/gas/ws/r/xcf/resource/resource-endpoint

Access SOAP Web services

In the following examples, "appid" is replaced by the application Id and "service" is 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://webserver/gas/ws/r/appid/service
Related concepts

Web service invalidation on page 268
The Web service invalidation feature identifies an invalid Web service configuration and informs the user agent with an HTTP error and message. Once identified, the invalid configuration must be corrected; in some instances, the Genero Application Server will need to be restarted.

URI Examples on page 48
Several URI examples with ways to help you launch applications.

Web service invalidation

The Web service invalidation feature identifies an invalid Web service configuration and informs the user agent with an HTTP error and message. Once identified, the invalid configuration must be corrected; in some instances, the Genero Application Server will need to be restarted.

Note: This feature is only supported for Genero Web services. It is not supported for Genero Desktop Client or Genero Web Client applications.

Web service invalid configuration - HTTP error 503

A Web service is typically an automated process, called on behalf of a user or process. Services requests can fail when:

- A Web service has an invalid configuration.
- The gwsproxy cannot start the DVM within DVM_AVAILABLE time. This can happen because of the invalid configuration or because of some other system error.

When a Web service fails to start, this feature prevents requests from failing forever. If the service (gwsproxy) can not start within DVM_AVAILABLE time, it informs the dispatcher to stop trying.

The next time a request is received for that Web service, the dispatcher remembers that the gwsproxy for the Web service cannot be started and sends a 503 HTTP service unavailable 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.

Troubleshoot error 503

To resolve an HTTP 503 error:

1. Check for invalid configuration files for Web services in the GAS using the monitor URL.
2. Review the invalid configuration file to find and fix the error.

   Tip: If you can not find an error, you can still try to restart the service by updating the configuration file’s (.xcf) time stamp. The proxy recognizes the update as a change in the configuration, and will attempt to start it. At the command line, type the command for your system:

   - Linux®
     
     touch filename.xcf
   
   - Windows®
     
     copy /b filename.xcf +,

3. If further assistance is required, contact your local Four Js support center. If requested to send the GAS logs, copy the logs for the date the issue arose to a zip file and email it to your support contact. Typically these logs are needed:

   - appdata/log/*.log.
• Related dispatcher daily logs (dispatcher, proxy, and vm) for the date the issue arose.

**Restarting a Web service after reconfiguration**

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.

**Related concepts**

**Troubleshooting HTTP errors** on page 121

Problems can be caused by issues as diverse as system errors, Web server errors, fglrun process errors, incorrect configurations, or other issues. Use the GAS logs to try to isolate the problem and determine if it is an issue that you can resolve or one that needs the assistance of your local Four Js support center.

**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 **Proxy: gwsproxy** on page 347 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 411 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.

**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.

```xml
<APPLICATION Parent="ws.default" mode="sticky">
  <EXECUTION>
    <PATH>$(res.path.app)/services/sticky-ws-service</PATH>
    <MODULE>sticky-main</MODULE>
  </EXECUTION>
  <TIMEOUT>
    <KEEP_ALIVE>60</KEEP_ALIVE>
  </TIMEOUT>
</APPLICATION>
```

You can change the name of the cookie by setting the environment variable **FGL_GWSPROXY_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_GWSPROXY_COOKIE_NAME">new_name</ENVIRONMENT_VARIABLE>
    <PATH>$(res.path.app)/services/sticky-ws-service</PATH>
    <MODULE>sticky-main</MODULE>
  </EXECUTION>
  <TIMEOUT>
```


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.

Related concepts

`APPLICATION (for a service)` on page 373

This `APPLICATION` element defines a service within the Genero Application Server configuration file or in an external application configuration file.

Deploying apps with Genero Archive

You deploy applications with the Genero Archive so that you have a simple process for packaging applications and services into an archive to deploy and manage on a GAS installation.

The deployment framework provides an interface to:

- Deploy applications and services packaged into a Genero Archive (`.gar`) file
- 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.

**Note:** If you are using Genero Studio to deploy with Genero Archive, see the *Packaging, deploying, and distributing apps* section of the *Genero Studio User Guide*.

What is Genero Archive?

A Genero Archive (`.gar`) file is a zip archive containing a `MANIFEST` file providing installation instructions and the list of application and services to make available on the GAS.

The archive name can be any name you wish. It need not reflect the applications or services contained within. You create an archive file using the `fglgar` tool.

The `MANIFEST` file provides deployments instructions, and must be at the root of the archive tree.
It is possible to embed external tools in 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.)

**Related concepts**

- **Steps to packaging applications** on page 273
- This sequence of tasks guides you through preparing an application for deployment; from bringing all of the required files and resources together in a directory to building a package.

## Create Genero Archive

Create a Genero Archive (.gar) file with default application configuration files and MANIFEST file.

If you just need to create a simple archive for one or more applications or services, follow this procedure. MANIFEST file and configuration files are created automatically.

**Before you begin:** Make sure that all required files are in the root directory of your archive.

Create the Genero Archive (.gar) file.

For example, at the command line type the following:

```
fglgar gar --application myApp.42r --service welcomeService.42m
```

**Note:** Each application needs to be specified with an `--application` option and each service with a `--service` option.

A Genero Archive (.gar) file is created that has the same name as your current directory. A MANIFEST file and configuration files are created automatically and included in the archive.

**Important:** If a MANIFEST file already exists when you run the command, errors will be raised.

**What to do next**

When you have created the archive in the above steps, your next task is to deploy the archive file on the GAS as detailed in Deploying and managing applications with GAR on page 285.

**Related tasks**

- **Create Genero Archive from MANIFEST** on page 275
- Create a Genero Archive (.gar) file for applications and services MANIFEST file.

## Understanding packaging with GAS

Before deploying applications on a GAS installation, you need to package the required compiled files and resources in a Genero archive file. The process of packaging applications is described.

**Creating a Genero Archive (.gar) file**

As a prerequisite, all of the required files and resources must be in an archive directory. When you run the `fglgar` tool with the `gar` command, a Genero archive is created taking a single directory as its parameter.

**Note:** If you need a more sophisticated archive tool than `fglgar` (to add only specific files to the archive, for example), you can use any other zip tool to create your GAR archive. For more information on using fglgar, see the `fglgar` topic in the *Genero Business Development Language User Guide*.

**Example gar file**

```
fglgar gar --application helloWorld.42r --service welcomeService.42m
```

The `.gar` file this command creates contains a MANIFEST file, your application modules, form files, configuration files, for example:
• MANIFEST
• helloWorld.42m
• helloWorld.42r
• helloWorld.42f
• welcomeService.42m
• helloWorld.xcf
• welcomeService.xcf

The MANIFEST file

The MANIFEST is an XML file that essentially provides a list of the applications and services in the gar to make available. It can be created automatically by fglgar at the command line for applications specified with the --application and/or --service option.

Or alternatively, if you have many applications to package, you may find it easier to first create the MANIFEST by hand. Running fglgar (without the --application and/or --service option) it checks if a MANIFEST file is present in the directory, and uses it to create the Genero Archive (gar) file.

Application configuration files

You can provide the application configuration (xcf) files if you wish but they are created automatically for you if you provide the --application and/or --service options with the name of the executable (42r or 42m) files instead of xcf files.

The xcf file is created based on default configurations in the GAS configuration file, as.xcf:

• With option --application, fglgar creates a default xcf based on the "defaultwa" configuration.
• With option --service, fglgar creates a default xcf based on "ws.default" configuration.

In both cases, the generated xcf file is given the name of the 42r or 42m provided at the command line.

Deploy the gar file and enable its applications

You deploy the archive on your GAS installation using one of the following methods:

• The Genero Deployment App
• The gasadmin gar command line tool

Note: If the GAS is on a remote Web server, you can not manage applications with gasadmin.

The contents of the archive is unpacked in the GAS $(res.appdata.path) /deployment path. The applications deployed are identified by the name of the xcf.

To make the applications available to end users, enable the archive using one of the above methods.

Related concepts

Deploying and managing applications with GAR on page 285
Deploying and managing applications and Web services using Genero Archives on the GAS.

MANIFEST file on page 280
To provide information about the applications and/or Web services packaged in a Genero Archive (gar) file, a MANIFEST file is created. If you are packaging applications, you will find it helpful to know how to create a MANIFEST file.

Related tasks

Create MANIFEST file on page 275
Create your MANIFEST file.

**Steps to packaging applications**

This sequence of tasks guides you through preparing an application for deployment; from bringing all of the required files and resources together in a directory to building a package.

**Gather files for packaging**

Gather all of the required files and resources and place them in a directory for packaging.

**Important:** Ensure that you have compiled your application for the version of Genero you are deploying to.

All your files must be included under one directory. From the following checklist of files, include those that apply to your needs:

- Application configuration files (.xcf)
- Modules (.42r, .42m)
- Form files (.42f)
- FGLPROFILE file
- Database files
- Images
- Resource files such as topmenu, toolbar, or styles resource files
- Web component files

1. Create a directory.
2. Create a directory structure for your package if required.

   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
   ```

   In the previous example, the files are organized within the root directory by a series of sub-directories. Such directories are not required. You can place all the files directly in the root as in the following archive for the same application:
   
   ```
   ./app.42m
   ./app.42r
   ./app.42f
   ./app.xcf
   ```

   **What to do next**

   When you have completed the above steps, your next task is to Organize the resources (images, logos, etc.) required by applications on page 273.

**Organize the resources (images, logos, etc.) required by applications**

Decide what resources are going to be used by all your applications. You need to create a directory in your archive especially for their deployment.

For example, 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.
1. Put your application's public images in a dedicated directory of your archive directory.
   You can name it, for example, "myAppPublicImages".

2. Put your private images in the root directory.

What to do next

When you have completed the above steps, your next task is to Create application configuration files on page 274.

Related concepts

Resource deployment on page 276
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.

Create application configuration files

Create your configuration files.

When using fgln GAR to deploy applications, configuration files are created automatically for you if you use the --application and/or --service options and provide the names of the executable files (42r or 42m).

Depending on your needs, you may need a specific configuration. For example, you may need to specify the customization project used by the Genero Browser Client (GBC). In this case, create your own configuration files as described in this procedure.

1. Create configuration files (.xcf) for your applications in your archive directory.
   See Application configuration file on page 113.

2. Add the elements that you need, and modify those you need to specify.

   Make sure to set the PATH element to $(res.deployment.path). If your compiled files (forms, modules, and so on) were in the /bin directory of your archive, you would specify the PATH element as shown in the example.

   In this example the configuration file is for a GBC Web application and the GBC element specifies the customization to use. For more information on customization, see the Genero Browser Client User Guide.

```
<APPLICATION Parent="defaultwa"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextwa.xsd">
  <EXECUTION>
    <PATH>${res.deployment.path}/bin</PATH>
  </EXECUTION>
  <!-- Override the default gbc with gbc-custom -->
  <UA_OUTPUT>
    <PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
    <GBC>gbc-custom</GBC>
    <TIMEOUT> Using="cpn.wa.timeout"</TIMEOUT>
  </UA_OUTPUT>
</APPLICATION>
```

In this example the configuration file has defined environment variables to specify where the GAS finds application resources, FGLPROFILE file, and the application database. For example, if your resource files (style files, private images, etc.) were in the /resources directory of your archive, you would specify the FGLRESOURCEPATH variable as shown in the example.

```
<APPLICATION Parent="defaultwa">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="FGLRESOURCEPATH">$(res.deployment.path)/resources</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="FGLPROFILE">$(res.deployment.path)/resources/fglprofile</ENVIRONMENT_VARIABLE>
```

In this example the configuration file has defined environment variables to specify where the GAS finds application resources, FGLPROFILE file, and the application database. For example, if your resource files (style files, private images, etc.) were in the /resources directory of your archive, you would specify the FGLRESOURCEPATH variable as shown in the example.
What to do next

When you have completed the above steps, your next task is to create a MANIFEST file.

Create MANIFEST file

Create your MANIFEST file.

When using fglgar to deploy applications, a MANIFEST file is created automatically for applications specified with the --application and/or --service option.

If you have many applications to package, you may find it easier to first create the MANIFEST by hand as described in this procedure.

1. Create a text file called MANIFEST in the root of your archive directory.
2. Add APPLICATION and SERVICE elements for each application and/or service that you need and specify their configuration files.

   If your configuration files (xcf) were in the /xcf directory of your archive, you would specify the element as shown in the example.

   In this example the RESOURCES element specifies that your applications' public images are in a dedicated directory of your archive directory.

   ```xml
   <MANIFEST>
   <DESCRIPTION>This archive contains one app and one service</DESCRIPTION>
   <RESOURCES>myAppPublicImages</RESOURCES>
   <APPLICATION xcf='xcf/app.xcf'></APPLICATION>
   <SERVICE xcf='xcf/webserviceApp.xcf'></SERVICE>
   </MANIFEST>
   ```

What to do next

When you have completed the above steps, your next task is to create a Genero Archive from MANIFEST.

Related concepts

MANIFEST file
To provide information about the applications and/or Web services packaged in a Genero Archive (gar) file, a MANIFEST file is created. If you are packaging applications, you will find it helpful to know how to create a MANIFEST file.

Create Genero Archive from MANIFEST

Create a Genero Archive (gar) file for applications and services MANIFEST file.

About this task

If you have many applications to package, it is recommended to create the MANIFEST by hand. The fglgar checks if a MANIFEST file is present in the archive directory and uses it to create the Genero Archive (.gar) file.

Before you begin

As a prerequisite, you have:

- Created a MANIFEST file.
- Created application configuration files (xcf) for your applications and/or services
Note: When you use the fglgar tool to create your Genero Archive and you specify --application or --service with an .xcf file as parameter, the fglgar tool automatically creates the MANIFEST for you, pointing to the specified .xcf file. If you specify --application or --service with a .42m or .42r file as parameter, the tool will automatically generate a standard .xcf for that app or service, and create a MANIFEST file that points to the generated .xcf file.

Use the fglgar tool to create a Genero Archive.

If you are in the directory containing your MANIFEST file and your program files, type the command:

```
fglgar gar -s directory
```

where the -s option specifies the source directory. If no other parameters are specified, the -s option is mandatory. However, if you add the --application or --service option, the -s is no longer mandatory and it defaults to current directory.

This creates a Genero 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 a Genero Archive (.gar) 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 a Genero Archive (.gar) file with the name myfuzzy.gar, drawing its content from the ./fuzzy directory.

Note: The Genero Archive (.gar) file name has no importance.

What to do next

When you have created the archive in the above steps, your next task is to deploy the archive file on the GAS as detailed in Deploying and managing applications with GAR on page 285.

Related tasks

Create MANIFEST file on page 275
Create your MANIFEST file.

Resource deployment

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

Sample application and resource deployment

The following example is provided to show the directory structure of the $(res.appdata.path) used by the deployment framework.
### $(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)

**Related reference**

File serving URIs on page 51

You can determine the file system location of resources specific to your deployed applications on the GAS from the application URL of a request. Examples are given for document root, public images, and web components.

**Paths to application resources**

There are specific directories in the appdata path for application resources used by the DVM. Resources are defined in the GAS for these paths.

Starting with Genero 3.00, three new predefined resources have been added:

- $(res.appdata.path)/public
  - This is a public resource path for all applications. At dispatcher start up, the public directory is created with write permissions (if it doesn't already exists). It contains two sub directories:
    - common
    - deployment
  - These directories will contain public images (that is 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.
  - **Note**: To ensure that resources are cacheable by the browser, the default public resource settings, /public and /public/common, are not configurable.

- $(res.public.resources)
  - This is a resource for public or common resources such as images used by applications. It resolves to $(res.appdata.path)/public/common

- $(res.path.tmp)
  - This is a resource where the GAS stores files temporarily during file transfer. It resolves to $(res.appdata.path)/tmp/

**Related reference**

File serving URIs on page 51
You can determine the file system location of resources specific to your deployed applications on the GAS from the application URL of a request. Examples are given for document root, public images, and web components.

**Deploy public images**

Public images are images such as logos, background images, etc., that are common to all or several of your deployed Genero Browser Client (GBC) applications. If your applications use public images, you need to know how to deploy them correctly.

**Public images shared by your applications**

At runtime images found in subdirectory paths of the $(res.public.resources) and the $(res.appdata.path)/public/deployment directory 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.

**Deploying public images**

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.
  - When deploying the Genero Archive (gar) file use the fglgar --resources option to specify the directory in the archive with your public images.
    
    **Tip:** If you create your own MANIFEST file and add the RESOURCES element, public images will be deployed without having to use the --resources option with fglgar.
  - The images contained in the archive's resource directory are copied to a directory in the $(res.appdata.path)/public/deployment/ path, for example:

    ```
    $(res.appdata.path)/public/deployment/myApp1
    ```

- Or if not using fglgar, you must then copy your public images by hand into the directory specified in your PUBLIC_IMAGEPATH.

  **Important:** 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 its subdirectory paths, public/common and public/deployment.

**Related concepts**

*Paths to application resources* on page 277

There are specific directories in the appdata path for application resources used by the DVM. Resources are defined in the GAS for these paths.

*Deploy private images* on page 279

Private images are resources that are only used by one application. There are some considerations for how you deploy them.

**Related tasks**

*Create MANIFEST file* on page 275

Create your MANIFEST file.

*Organize the resources (images, logos, etc.) required by applications* on page 273
Deploying apps with Genero Archive

Decide what resources are going to be used by all your applications. You need to create a directory in your archive especially for their deployment.

**Deploy private images**

Private images are resources that are only used by one application. There are some considerations for how you deploy them.

**Private images**

Before deploying, you can place private images 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.

**Set an environment variable for FGLIMAGEPATH**

If your private images are in subdirectories of the root, you will need to ensure that an environment variable for 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$(res.path.separator)images
$(res.path.separator)private$(res.dir.separator)images</ENVIRONMENT_VARIABLE>
```

**Tip: Use platform independent path and directory resource separators**

- `$(res.path.separator)` is a delimiter resource to separate paths. It resolves to either a colon `:` on a UNIX™-like system including macOS™ or a semi-colon `;` on a Windows® system.
- `$(res.dir.separator)` is a directory path delimiter resource. It resolves to either a slash `/` on a UNIX™-like system including macOS™ or a backward slash `\` on a Windows® system.

If there are resources in, for example, both a parent and its subdirectory, you have to specify each directory separately. For more details on FGLIMAGEPATH, see the *Genero Business Development Language User Guide*.

**Related concepts**

Deploy public images on page 278

Public images are images such as logos, background images, etc., that are common to all or several of your deployed Genero Browser Client (GBC) applications. If your applications use public images, you need to know how to deploy them correctly.

**Related tasks**

Organize the resources (images, logos, etc.) required by applications on page 273

Decide what resources are going to be used by all your applications. You need to create a directory in your archive especially for their deployment.

**Deploy Web components for GBC**

To enable access to Web components for your Genero Browser Client (GBC) applications, you need to add them to specific `$(APPDATA)/public/` directories.

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.

**Related concepts**

Resource deployment on page 276

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.

WEB_COMPONENT_DIRECTORY on page 440
The `WEB_COMPONENT_DIRECTORY` element specifies the path where Web components for an application are located.

**Related tasks**

- Deploying, enabling, and running applications on GAS on page 296
- Use these examples for building archives, deploying applications, enabling, and running them locally on the GAS.

## MANIFEST file

To provide information about the applications and/or Web services packaged in a Genero Archive (gar) file, a MANIFEST file is created. If you are packaging applications, you will find it helpful to know how to create a MANIFEST file.

A MANIFEST file must be created and included in the Genero archive (gar) file. The following are some options to consider depending on your requirements:

- If you have one or more applications, you can run the `fglgar` tool with the `gar` command with an `--application` and/or `--service` option for each application or service you wish to include. An Genero Archive (.gar) file is generated and a MANIFEST file is created automatically and included in it.
- Alternatively, if you have many applications to package, you may find it easier to create the MANIFEST by hand. The `fglgar` checks if a MANIFEST file is present in the archive directory and uses it to create the gar.

**Important:** You must not use `fglgar` command options that provide for the generation of a MANIFEST (such as `--application` and/or `--service`) when a MANIFEST already exists, as errors will be raised.

**Note:** For more information on using `fglgar`, see the `fglgar` topic in the Genero Business Development Language User Guide.

### Syntax

A MANIFEST file is an XML file. It has a MANIFEST element at its root.

```xml
<MANIFEST>
  <DESCRIPTION>This archive contains one app and one service</DESCRIPTION>
  <![TRIGGERS component='comp_name']>
    <![DEPLOY>deploy.sh</DEPLOY>]
    <![UNDEPLOY>undeploy.sh</UNDEPLOY>]
  <![RESOURCES>]
  <![APPLICATION>]
  <![SERVICE>]
</MANIFEST>
```

### Child elements

**Important:** Element order. If child elements are present, they must be set in the order listed or as shown.

Child elements can include the following:

- One `DESCRIPTION (for manifest)` on page 281 element.
- Zero or one `TRIGGERS (for manifest)` on page 281 element.
- Zero or one `RESOURCES (for manifest)` on page 283 element.
- Zero or more `APPLICATION (for manifest)` on page 284 elements.
- Zero or more `SERVICE (for manifest)` on page 284 elements.

## MANIFEST file

```xml
<MANIFEST>
  <DESCRIPTION>This archive contains one app and one service</DESCRIPTION>
  <TRIGGERS component='comp_name'>
    <DEPLOY>deploy.sh</DEPLOY>
    <UNDEPLOY>undeploy.sh</UNDEPLOY>
  </TRIGGERS>
</MANIFEST>
```
DESCRIPTION (for manifest)

The `DESCRIPTION` element provides a textual description of the archive.

Syntax

```
<DESCRIPTION>Description</DESCRIPTION>
```

1. `Description` specifies a description of the archive.

Child elements

There are no child elements.

Usage

Use this element to provide a description of the archive, which is displayed when listing archives with `gasadmin` tool on page 348.

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

Example

```
<DESCRIPTION>This archive contains an application to be deployed</DESCRIPTION>
```

Parent elements

This element is a child of the `MANIFEST` element in `MANIFEST file` on page 280

TRIGGERS (for manifest)

The `TRIGGERS` element defines a set of deployment parameters that can be used when deploying an application with `fglgar`.

It contains elements which specify programs (.42r or .42m) or scripts to execute when deploying and undeploying applications on the GAS when `fglgar` is used.

Note: Deployment triggers are typically not required, you can deploy your applications without them.

Syntax

```
[<TRIGGERS [component = 'comp_name']>
  [<DEPLOY></DEPLOY>]
  [<UNDENDEPLOY></UNDENDEPLOY>]
</TRIGGERS>]
```

1. The `comp_name` specifies a set of environment context for the runtime required by the `DEPLOY` and `UNDEPLOY` programs. If not set, it defaults to the "cpn.gar.execution.local" Web Application component defined in the GAS `as.xcf`

Child elements

The `TRIGGERS` element may contain the following child elements:
1. Zero or one **DEPLOY** element.
2. Zero or one **UNDEPLOY** element.

**Example MANIFEST with triggers**

```
<TRIGGERS component = 'cpn.gar.execution.local'>
  <DEPLOY>fglrun mydeploy.42r</DEPLOY>
  <UNDEPLOY>fglrun myundeploy.42r</UNDEPLOY>
</TRIGGERS>
```

**fglgar with trigger options**

In this example, the `component` value - "cpn.gar.execution.local" - is specified by the `--trigger-component` at the command line when creating an archive:

```
fglgar --gar --application myapp.xcf --trigger-component cpn.gar.execution.local --deploy-trigger "fglrun mydeploy.42r" --undeploy-trigger "fglrun myundeploy.42r"
```

**Note:** If the `--deploy-trigger` fails, the entire deployment will fail. Whereas if the `--undeploy-trigger` fails, the undeployment is still carried out.

The MANIFEST file is created automatically with the trigger information.

**Parent elements**

This element is a child of the MANIFEST element in MANIFEST file on page 280

**DEPLOY (for manifest)**

The **DEPLOY** element specifies a program or script to execute when deploying applications on the GAS.

This is an optional element of the TRIGGERS element.

**Syntax**

```
[<DEPLOY>deployProgramScript</DEPLOY>]
```

1. The `deployProgramScript` deployment program may be a Genero 4GL (.42r or .42m) file or a script file.

**Child elements**

There are no child elements.

**Example DEPLOY defined with Genero 4GL program**

```
<TRIGGERS component = 'cpn.gar.execution.local'>
  <DEPLOY>fglrun mydeploy.42r</DEPLOY>
  <UNDEPLOY></UNDEPLOY>
</TRIGGERS>
```

**Example DEPLOY defined with script**

```
<TRIGGERS component = 'cpn.gar.execution.local'>
  <DEPLOY>mydeploy.sh</DEPLOY>
  <UNDEPLOY></UNDEPLOY>
</TRIGGERS>
```

**Parent elements**

This element is a child of the TRIGGERS (for manifest) on page 281 element.
UNDEPLOY (for manifest)
The UNDEPLOY element defines a command and program or script used when undeploying applications deployed in an archive.

This is an optional element of the TRIGGERS element.

Syntax

\[ \text{<UNDEPLOY> unDeployProgramScript</UNDEPLOY> \]}

1. The *unDeployProgramScript* program may be a Genero 4GL (.42r or .42m) file or a script file.

Child elements

There are no child elements.

Example UNDEPLOY defined with Genero 4GL program

```xml
<TRIGGERS component = 'cpn.gar.execution.local'>
  <DEPLOY></DEPLOY>
  <UNDEPLOY>fglrun myundeploy.42r</UNDEPLOY>
</TRIGGERS>
```

In this example the

Example UNDEPLOY defined with script

```xml
<TRIGGERS component = 'cpn.gar.execution.local'>
  <DEPLOY></DEPLOY>
  <UNDEPLOY>myundeploy.sh</UNDEPLOY>
</TRIGGERS>
```

Parent elements

This element is a child of the TRIGGERS (for manifest) on page 281 element.

RESOURCES (for manifest)
The RESOURCES element specifies the path to your application resources.

The directory specified must be in the archive.

Syntax

\[ \text{<RESOURCES> resource_directory</RESOURCES> \]}

1. The *resource_directory* specifies the directory in your archive file where images for your applications are found.

Usage example

If your images, reports, etc. were in the /resources directory of your archive, you would define it as shown.

```xml
<MANIFEST>
  <DESCRIPTION></DESCRIPTION>
  <TRIGGERS></TRIGGERS>
  <RESOURCES>/resources</RESOURCES>
</MANIFEST>
```
Child elements
There are no child elements.

Parent elements
This element is a child of the MANIFEST element in MANIFEST file on page 280

APPLICATION (for manifest)
The APPLICATION element specifies the location of the configuration file for the application.

Syntax
This element takes a mandatory xcf attribute.

```
[<APPLICATION xcf='xcf_fileName'></APPLICATION>]
```

1. The xcf_fileName specifies the path to an application configuration file (xcf) in an archive file.

Usage
If your application xcf files were in the /xcf directory of your archive, you would define them as shown. The applications specified must be in the archive.

```
<MANIFEST>
  <DESCRIPTION></DESCRIPTION>
  <TRIGGERS></TRIGGERS>
  <RESOURCES></RESOURCES>
  <APPLICATION xcf='xcf/test.xcf'></APPLICATION>
  <APPLICATION xcf='xcf/app2.xcf'></APPLICATION>
</MANIFEST>
```

Child elements
There are no child elements.

Parent elements
This element is a child of the MANIFEST element in MANIFEST file on page 280

SERVICE (for manifest)
The SERVICE element specifies the path to a Web services application configuration file (xcf) in an archive file.

Syntax
This element takes a mandatory xcf attribute.

```
[<SERVICE xcf='xcf_fileName'></SERVICE>]
```

1. The xcf_fileName specifies the configuration file for the Web service.

Child elements
There are no child elements.
### Usage example

If your Web services application `xcf` files were in the `/xcf` directory of your archive, you would define them as shown. The applications specified must be in the archive.

```xml
<MANIFEST>
  <DESCRIPTION/>
  <TRIGGERS/>
  <RESOURCES/>
  <APPLICATION>
    <SERVICE xcf='xcf/webserviceApp1.xcf'/>
    <SERVICE xcf='xcf/webserviceApp2.xcf'/>
  </APPLICATION>
</MANIFEST>
```

### Parent elements

This element is a child of the `MANIFEST` element in **MANIFEST file** on page 280

---

### Deploying and managing applications with GAR

Deploying and managing applications and Web services using Genero Archives on the GAS.

**Related concepts**

Deployment on remote server on page 298

The Genero deployment service can be accessed remotely to perform tasks to deploy applications.

### Genero Archive lifecycle

The lifecycle of a deployed Genero Archive involves six stages: deployed, secured, activated, deactivated, undeployed, and cleaned.

Each stage is defined below.

1. **Deployed** - The archive is deployed but not activated.
2. **Secured** - Applications and services provided by an archive are available to authorized users only. Security is implemented by the Genero Identity Provider (GIP) and configured using the The Genero Deployment App.  
   
   **Note:** Step 2 is not required if the application or service is provided for all users to access. You can unsecure a secured Genero Archive.
3. **Activated** - Applications and services provided by an archive are available to users.
4. **Deactivated** - Applications and services provided by an archive are no longer available to users. A deactivated archive can be reactivated.
5. **Undeployed** - The archive is no longer deployed, and cannot be reactivated. The MANIFEST file is removed. Physical cleanup of the archive is not done.
6. **Cleaned** - The archive is physically removed.

**Related concepts**

Genero Identity Provider (GIP) on page 204

Genero provides its own Identity Provider for securing applications and RESTful Web services.

### 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, but are not yet available to users.

### Deploy options

On your GAS installation use one of the following methods:

- The Genero Deployment App to deploy and secure applications with Genero Identity Provider (GIP).
- The `gasadmin` command line tool. You can use it to deploy applications on a local GAS installation.
- The `DeployGar` tool. This allows you to manage applications on both local and remote servers. Commands can be included in a script file. See Automatize application deployment via scripts on page 242.

**Genero Deployment Application**

Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.

![Figure 70: Genero Deployment Application user interface](image)

By default, only users with the GIP `Role.Admin` authorization scope can access the DeploymentApp. Access to the application needs to be provided by your GIP administrator for other users. For more information see Managing GIP components with the Genero Console App. To access the application:


**Warning:** Any URL used with the GIP must contain a valid hostname; it cannot contain "localhost". The GIP uses cookies, and cookies do not work well with "localhost".

**Deploy an archive with the Genero Deployment App**

Use this procedure when you need to deploy an Genero Archive (gar) file on your Genero Application Server using the Genero Deployment App.

**Before you begin:**

- Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.
- You have created a Genero Archive (gar) file.

**About this task:**

The archive is unpacked in the `$res.appdata.path)/deployment` directory.

**Note:** The application/service deployed is named with the `xcf`, not the Genero Archive (gar) file name.
1. Open the Genero deployment application.

   http://myhost:6394/ua/r/admin/DeploymentApp

   On a remote machine, you need to specify the appropriate URL for the server and port in your browser, for example:

   http://zeus:8090/ua/r/admin/DeploymentApp

2. Select the Deploy > Genero Archives ... menu.
3. In the Manage Genero Archives screen click the Deploy button.
4. Use the dialog that opens to locate the archive you wish to deploy.

   You can drag and drop your file into the dialog window.

   The new archive is deployed and displayed in the list.

All applications and services are deployed in the default group of the application server.

Related concepts
Deploying and securing applications and Web services on page 236
You have two methods for deploying and securing applications and Web services.

Deploy and secure an archive with the Genero Deployment App
There are options for securing applications, depending on whether you are securing using the Genero Identity Provider or a third party IdP.

Typically, security is managed by the Genero Identity Provider (GIP). You must secure the gar file using the Genero Deployment App. For an example of deploying and securing applications, follow the procedure described in Deploy and secure an application using the Genero Deployment App on page 237.

When securing an application using a third party Identity Provider (IdP), such as Google, this is not managed by the Genero Deployment App. See the procedures described in the section How to implement Single sign-on (SSO) on page 139.

Why use Genero Deployment App instead of gasadmin?

Securing and unsecuring applications can only be performed by the Genero Deployment App using the GIP. Security cannot be performed by the gasadmin command tool.

In most other respects the Genero Deployment App provides the same services as the gasadmin command line tool because actually, the Genero Deployment App executes the gasadmin on server side via a REST service.

For example, details of all applications and services that are deployed and/or secured in the application server can be viewed using either the Genero Deployment App, or the gasadmin tool.

If a gar is deployed via gasadmin, it will appear in the Genero Deployment App interface, and vice versa. See List all deployed archives on page 289.

Related concepts
Deploying and securing applications and Web services on page 236
You have two methods for deploying and securing applications and Web services.

Manage applications on page 232
Use the Genero Console App to manage the access to the applications and service applications whose access is to be managed by the Genero Identity Provider (GIP) on the selected Genero Application Server.

Manage Web services on page 233
Scopes provide access to Web services. You will use the Console App to register the scopes, but you will not explicitly register a Web services application.

**Deploy an archive with gasadmin**

Use this procedure to deploy an archive on your Genero Application Server using the gasadmin tool.

**Before you begin:**

As a prerequisite, you have a Genero Archive (gar) file.

**Tip:** It is a good idea to set your environment to access the gasadmin command by running the script $FGLASDIR/envas.

**About this task:**

When you use gasadmin to deploy the Genero Archive (gar) file the archive is unpacked in the $(res.appdata.path)/deployment directory. Static resources are zipped for better performance at runtime.

**Note:** The application deployed is named with the xcf, not the Genero Archive (.gar) file name. To get details of deployed applications run the gasadmin gar --list-archives command where the GAS is running.

At the command line type the gasadmin command with the appropriate command and arguments.

To deploy an archive named fuzzy.gar:

```
gasadmin gar --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 gar --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.
```

**Important:** If you start the dispatcher with the option (-E) to override the $(res.appdata.path) location, you must also override the resource when deploying applications with the gasadmin gar command, in order to deploy to the correct directory.

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 gar -E res.appdata.path=/work/tmp/gas/appdata --deploy-archive
myapp.gar
```

All applications and services are deployed in the default group of the application server.

**File system layout of deployed archive**

A Genero Archive contains application source files organized into subdirectories for modules, forms, etc. includes the **MANIFEST** and application configuration files. The directory structure is recreated when deployed.

When you deploy, the contents of the archive are unpacked in the GAS $(res.appdata.path)/deployment path.

A dedicated directory is created under the $(res.appdata.path)/deployment 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.

**Genero archive file**

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-20190522-123456/MANIFEST
- <APPDATA>/deployment/myapp-20190522-123456/modules/app.42m
- <APPDATA>/deployment/myapp-20190522-123456/modules/app.42r
- <APPDATA>/deployment/myapp-20190522-123456/forms/app.42f
- <APPDATA>/deployment/myapp-20190522-123456/xcf/app.xcf

**List all deployed archives**

You can list all deployed archives with a single command.

**Before you begin:**

Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.

1. View the list of applications using the Genero Deployment App.
   
   The Deployment App application is located at `http://host:port/gas/ua/r/admin/DeploymentApp`. For details of accessing the deployment app as an authorized user, see the Genero Identity Provider (GIP) on page 204.

2. Or alternatively, use the `gasadmin` command.
   
   To list all deployed archives, enter the `gasadmin` command with the appropriate arguments:

```
$ gasadmin gar --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 gar --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 is my first archive
    Application(s):
      - 22223.xcf
      - 22224.xcf
    Service(s):
      - 15233.xcf

  Name : archive2
  State : Disabled
  Description : This is my second archive
    Application(s):
      - 22315.xcf
    Service(s):
      - 17746.xcf
      - 19164.xcf
```

An example of XML output for an archive listing:

```
<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 contain some MESSAGE or ERROR nodes.

**Enable an archive**

When you activate (enable) a deployed archive, the applications and services provided by the archive become available to users.

**Enable options**

On your GAS installation use one of the following methods:

- The Genero Deployment App to deploy and secure applications with Genero Identity Provider (GIP).
- The `gasadmin` command line tool. You can use it to deploy applications on a local GAS installation.
• The DeployGar tool. This allows you to manage applications on both local and remote servers. Commands can be included in a script file. See Automatize application deployment via scripts on page 242.

Activate (enable) a deployed archive with the Genero Deployment App
Use this procedure when you need to enable an archive on your Genero Application Server using the Genero Deployment Application.

Before you begin:
• Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.
• You have created a Genero Archive (gar) file.

About this task:
When you activate (enable) a deployed archive, the applications and services provided by the archive become available to users.

1. Open the Genero deployment application.

   http://myhost:6394/ua/r/admin/DeploymentApp

   On a remote machine, you need to specify the appropriate URL for the server and port in your browser, for example:

   http://zeus:8090/ua/r/admin/DeploymentApp

2. Select the Deploy > Genero Archives ... menu.

3. In the Manage Genero Archives screen select the archive you wish to enable and click on the Enable button.

   The status of the archive is set as Active.

Related tasks
Activate (enable) a deployed archive with gasadmin on page 291
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.

Deactivate (disable) a deployed archive on page 292
When you deactivate (disable) a deployed Genero Archive (gar) file, the applications and services provided by the archive are not exposed (no longer available to users).

Activate (enable) a deployed archive with gasadmin
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 (gar) file.

1. Enter the gasadmin command with the appropriate arguments.

   To show a list of archives and details of their status:

   gasadmin gar --list-archives

   To enable an archive named fuzzy:

   gasadmin gar --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:

```
gasadmin gar --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.

XML Output examples:

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully enabled</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to enable <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure....</ERROR>
</DEPLOYMENT>
```

Related tasks

Activate (enable) a deployed archive with the Genero Deployment App on page 291

Use this procedure when you need to enable an archive on your Genero Application Server using the Genero Deployment Application.

Deactivate (disable) a deployed archive on page 292

When you deactivate (disable) a deployed Genero Archive (gar) file, the applications and services provided by the archive are not exposed (no longer available to users).

Deactivate (disable) a deployed archive

When you deactivate (disable) a deployed Genero Archive (gar) file, the applications and services provided by the archive are not exposed (no longer available to users).

As a prerequisite, you have a deployed Genero Archive (.gar) file that has been enabled.

Disabling an archive means disabling all applications and services provided by the Genero Archive (.gar) file. 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 (gar) file named `fuzzy.gar`:

```
gasadmin gar --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 gar --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:

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully undeployed</MESSAGE>
</DEPLOYMENT>
```
Undeploy an archive

An undeployed archive is no longer deployed and cannot be reactivated.

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.

No files are removed when you undeploy. The cleanup of the undeployed archive directory must be explicitly requested.

Undeploy options

On your GAS installation use one of the following methods:

- The Genero Deployment App to deploy and secure applications with Genero Identity Provider (GIP).
- The gasadmin command line tool. You can use it to deploy applications on a local GAS installation.
- The DeployGar tool. This allows you to manage applications on both local and remote servers. Commands can be included in a script file. See Automatize application deployment via scripts on page 242.

Undeploy an archive with the Genero Deployment App

Use this procedure when you need to undeploy an archive on your Genero Application Server using the Genero Deployment App.

Before you begin:

Before you can use the Genero Deployment App, you need to configure and enable the Genero Identity Provider (GIP) on your GAS. You use the StarterApp to perform the installation of the GIP. The Genero Deployment App utilizes the Deployment Service. You must select these components in your GIP configuration at the time of installation. For further details, see Setting up the Genero Identity Provider on page 215.

1. Open the Genero deployment application.
   
   \[http://myhost:6394/ua/r/admin/DeploymentApp\]

   On a remote machine, you need to specify the appropriate URL for the server and port in your browser, for example:

   \[http://zeus:8090/ua/r/admin/DeploymentApp\]

2. Select the **Deploy > Genero Archives ...** menu.

3. In the Manage Genero Archives screen select the archive you wish to undeploy and click on the **Undeploy** button.

   The archive is removed from the list.

Undeploy an archive with gasadmin

When you undeploy an Genero Archive (gar) file, the archive is no longer deployed and cannot be reactivated.

Enter the `gasadmin` command with the appropriate arguments.
To undeploy a Genero Archive (.gar) file named fuzzy.gar:

```
gasadmin gar --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 gar --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 session -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. Run the `gasadmin` command with the appropriate arguments.

   To clean up all undeployed archives:

   `````
   gasadmin gar --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 gar --clean-archives --xml-output
   ```
To disable confirmation for each archive removal, add the \texttt{-all} option:

```

gasadmin gar --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 \texttt{quit}.

\section*{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.

1. Deploy the new version of the archive. See \textit{Deploy an archive with the Genero Deployment App} on page 286, \textit{Deploy an archive with gasadmin} on page 288.
2. Disable the previously deployed archive. See \textit{Deactivate (disable) a deployed archive} on page 292.
3. Enable the newly deployed archive. See \textit{Activate (enable) a deployed archive with the Genero Deployment App} on page 291, \textit{Activate (enable) a deployed archive with gasadmin} on page 291.
4. Undeploy the previously deployed archive. See \textit{Undeploy an archive with the Genero Deployment App} on page 293, \textit{Undeploy an archive with gasadmin} on page 293.

\section*{Upgrade a Web service while the GAS is running}

If you need to upgrade your Web service while it is still running on the GAS, there are options to consider depending on how you wish to perform this task.

For example, you need to add new operations or provide new parameters for your Web services. These changes should not effect existing clients running your service and avoid interrupting requests coming in for the Web service as you upgrade.

There are two options:

- The recommended procedure is to use Genero Archive to disable the old service and enable a new one as described in \textit{Upgrade a Web service using Genero archive} on page 295. Be aware that this procedure may interrupt the service for a short period of time.
- If the risk of an interruption of service is not acceptable, then you have the option of manually overwriting the configuration file. Using this procedure comes with the risk of corrupting the Web service and is not recommended. See \textit{Upgrade a Web service manually} on page 296.

\section*{Upgrade a Web service using Genero archive}

Use this procedure to use Genero Archive to upgrade a Web service running on the GAS.

As a prerequisite, your services were previously deployed as an archive. You still have the requisite archive files (see \textit{Create Genero Archive} on page 271 or \textit{Create Genero Archive from MANIFEST} on page 275.)

\textbf{Note:} Be aware that this method of upgrading a Web service may interrupt the service for a short period of time while you are disabling the old service and enabling the new one.

1. Make the necessary modifications to the code of your Web services \texttt{4gl} files and compile them.
2. Create a new archive file using the \texttt{fglgar} tool.

```

fglgar gar --output new_archive.gar
```

3. To upgrade the Web service with the new archive, follow the procedures described in \textit{Upgrade an archive} on page 295.

After completing the upgrade procedure, the next request will start the new service.

\section*{Related tasks}

\textit{Upgrade a Web service manually} on page 296
Use this procedure to manually upgrade a Web service running on the GAS.

**Upgrade a Web service manually**

Use this procedure to manually upgrade a Web service running on the GAS.

**Warning:** You need to be aware that this procedure involves manually overwriting the `.xcf` file of the existing service with the configuration of the new service. This can lead to service inconsistency or corruption because the old archive may contain several `.xcf` files, and may still be active for other services or applications. You are advised to proceed with caution.

1. Make the necessary modifications to the code of your Web services `.g1` files and compile them.
2. Create a subdirectory for your service on the GAS. For instance, you can use a directory path like `${res.appdata.path}/my_service`.
   
   **Important:** You must not create a directory in the `${res.appdata.path}/deployment` path as this is reserved for services deployed with Genero archive.
3. Copy the new service files to the created directory.
4. Create a configuration file (`.xcf`) that points to the directory where your service is located.
   
   **Important:** You need to give the configuration file the same name as your existing Web service.
5. Copy the `.xcf` file manually to the `${res.appdata.path}/services` directory, replacing the existing Web service configuration file.

After completing the upgrade procedure, the next request for the Web service will detect the `.xcf` file has changed and the GAS will start a new proxy to use the new service.

**Related concepts**

- **Configure applications for Web service** on page 116
  Create an application configuration file (`.xcf`) for a Web services application.

**Related tasks**

- **Upgrade a Web service using Genero archive** on page 295
  Use this procedure to use Genero Archive to upgrade a Web service running on the GAS.
- **Find invalid Web service configuration files** on page 180
  Follow this procedure to check for invalid configuration files for Web services in the GAS.

**Compile, package, and redeploy after updating software**

Your applications must be compiled using the version of Genero you are deploying to.

After upgrading to a new version of the Genero software, you must recompile, repackage, and redeploy your applications.

Similarly, if you switch between versions of the Genero software, ensure the application you deploy matches the version of the software you are deploying to. This is true whether you are switching from an older version to a new version of the software, or reverting from a new version of the software back to an older version of the software.

For further information on managing different versions of Genero software and applications, see the *Install multiple Genero versions* topic in the *Install and License your Genero Products* manual.

**Deploying, enabling, and running applications on GAS**

Use these examples for building archives, deploying applications, enabling, and running them locally on the GAS.

As a prerequisite, you must have your applications and/or services created, compiled, and tested. Consolidate all the necessary files for your archive under a root directory.

In this page there are examples for building archives, deploying, enabling, and running applications:

- **Overview of the main tasks** on page 297
- **Building archives**
For a full understanding of what Genero archiving offers, please read all archiving topics in this section.

**Overview of the main tasks**

This procedure provides you with a quick overview of the main steps for archiving an application using `fglgar`, and deploying to local and remote servers.

1. Prepare the required program files, and organize those in a directory for packaging.
   
   See [Steps to packaging applications](#) on page 273.

2. Create the archive file using the `fglgar gar` command.
   
   See [Create Genero Archive](#) on page 271 or [Create Genero Archive from MANIFEST](#) on page 275

3. Deploy the archive file.

   You deploy the archive on your GAS installation using one of the following methods:
   
   - The Genero deployment portal. See [Deploy an archive with the Genero Deployment App](#) on page 286.
   - The `gasadmin` command line tool. See [Deploy an archive with gasadmin](#) on page 288.

4. Enable the archive.

   This makes applications in the archive available to end users.

   You enable the archive on your GAS installation using one of the following methods:

   - The Genero deployment App. See [Activate (enable) a deployed archive with the Genero Deployment App](#) on page 291.
   - The `gasadmin` command line tool. See [Activate (enable) a deployed archive with gasadmin](#) on page 291.

   **Note:** If the GAS is on a remote Web server, you can not manage applications with `gasadmin`.

**Build an archive with public resources**

An archive can contain common or public resources such as logos, images, reports, etc., that all deployed applications on the GAS can use. Using `fglgar` as shown in this procedure creates this type of archive.

As a prerequisite, you have put your application's public images in a dedicated directory of your archive directory. You can name it, for example, "myAppPublicImages".

Create the Genero Archive (gar) file.

For example, at the command line type the following command with the `--resource` option specifying the name of the public image directory in the archive directory:

```
fglgar gar --resource myAppPublicImages --application myApp.xcf
```

**Build an archive with deployment triggers**

An archive can be defined with deployment parameters. These are commands that execute when deploying and undeploying an application on the GAS. Using `fglgar` as shown in this procedure creates the archive with deployment trigger options.

**Note:** Deployment triggers are typically not required, you can deploy your applications without them.

Create an Genero Archive (gar) file.

For example:

```
fglgar gar --application myApp.xcf --trigger-component cpn.gar.execution.local
```
--deploy-trigger "fglrun mydeploy.42r" --undeploy-trigger "fglrun myundeploy.42r"

Where:

- The `--trigger-component` option references a trigger component in the GAS as.xcf that defines the runtime environment where triggers are run. If not set, it defaults to "cpn.gar.execution.local".
- The `--deploy-trigger` specifies your DEPLOY command.
- The `--undeploy-trigger` options specifies the UNDEPLOY command.

These commands are saved in a MANIFEST file in the gar. See TRIGGERS (for manifest) on page 281.

**Important:** If a MANIFEST file already exists when you run the command, errors will be raised.

### Deploy your application on your machine

**About this task:**

Once you have created an archive for your application, you can now deploy it locally on your machine where the GAS is installed. If the standalone dispatcher on your GAS installation is not already started, run it from the command line using `httpdispatch`.

Deploy your Genero Archive (gar) file.

You deploy the archive on your GAS installation using one of the following methods:

- The Genero deployment portal. See Deploy an archive with the Genero Deployment App on page 286.
- The `gasadmin` command line tool. See Deploy an archive with gasadmin on page 288.

### Enable your application on your machine

Once you have deployed your application, you can now make it available for end users by enabling it on the machine where the GAS is installed. If the standalone dispatcher on your GAS installation is not already started, run it from the command line using `httpdispatch`.

Enable your application.

You enable the archive on your GAS installation using one of the following methods:

- The Genero deployment portal. See Activate (enable) a deployed archive with the Genero Deployment App on page 291.
- The `gasadmin` command line tool. See Activate (enable) a deployed archive with gasadmin on page 291.

### Run the deployed application

Once the application is enabled, you can now run it. If the standalone dispatcher on your GAS installation is not already started, run it from the command line using `httpdispatch`.

In a browser enter the address of your deployed application.

http://localhost:6394/ua/r/myApp

In this example, the URL is looking for a configuration file named `myApp.xcf`.

You should see your application displayed and be able to interact with it. You have successfully deployed an application.

### Deployment on remote server

The Genero deployment service can be accessed remotely to perform tasks to deploy applications.

You can access it on the GAS at a base URL of `/ws/r/admin/GeneroDeploymentService`. 

--deploy-trigger "fglrun mydeploy.42r" --undeploy-trigger "fglrun myundeploy.42r"
Deploying a Genero Archive (gar) file on a remote server where you do not have access to the command line gasadmin tool, requires you to use the deployment service. The service is protected by Genero Identity Provider (GIP) on page 204 and requires authorization to use it.

The deployment service is based on the gasadmin gar and implements most of its commands for managing Genero Archive (gar). Table 41: Genero Archive deployment service actions over HTTP on page 299 describes the actions you can perform and provides you with examples of using them over HTTP.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>HTTP request details</th>
</tr>
</thead>
</table>
| **deploy** | Deploys a Genero Archive (gar) file on the Genero Application Server by calling:
  - gasadmin gar --deploy-archive
  Returns the command result in XML format. | HTTP PUT request of following form:
  - URL: `http://app_server:port/connector/gas/admin/GeneroDeploymentService/deploy?archive=name`
  - Note: `archive=name` is mandatory to identify the archive on the GAS.
  To send the archive to the Genero Application Server, you can use curl with a `-H` parameter to provide the access token, and a `PUT` instruction:
  ```
  curl -H "Authorization: Bearer ACCESS_TOKEN" 
  -T test.gar 
  http://app_server:port/connector/gas/admin/GeneroDeploymentService/deploy?archive=name
  ```
  - where:
    - The `ACCESS_TOKEN` is provided by the GIP as authorization to use the deployment service.
    - **Tip:** The access token is obtained via the GIP GetToken on page 251 tool. For examples, using this tool see Automatize application deployment via scripts on page 242.
    - `app_server` is the server name or IP address.
    - `port` is the port where the application server or Web server is listening.
    - `connector` is the Genero Application Server connector (`/gas`, for example).

| **enable** | Enables all applications and services of a given archive by calling:
  - gasadmin gar --enable-archive | HTTP GET request of following form:
  - URL: `http://app_server:port/connector/gas/admin/GeneroDeploymentService/enable?archive=name` |
| **disable** | Disables all applications and services of a given archive by calling:
  - gasadmin gar --disable-archive | HTTP GET request of following form:
  - URL: `http://app_server:port/connector/gas/admin/GeneroDeploymentService/disable?archive=name` |
<table>
<thead>
<tr>
<th><strong>Action</strong></th>
<th><strong>Description</strong></th>
<th><strong>HTTP request details</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>undeploy</td>
<td>Undeploys a genero archive on the Genero Application Server by calling: <code>gasadmin gar --undeploy-archive</code></td>
<td>HTTP GET request of following form: URL: <code>/ws/r/admin/GeneroDeploymentService/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 gar --list-archive</code></td>
<td>HTTP GET request of following form: URL: <code>/ws/r/admin/GeneroDeploymentService/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 the command: <code>gasadmin gar --clean-archives</code></td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Related concepts**

gasadmin tool on page 348

The gasadmin tool is provided as an administrative command for the Genero Application Server.

---

**Run Genero Application Server in Docker container**

These topics provide you with the information needed to perform tasks for building a docker image and running the GAS in a docker container.

The FourjsGenero docker_debian_fgl_gas GitHub repository contains scripts and docker files to help you work with creating the docker image. Clone the GitHub repository to a local directory on your system.

**Note:**

Docker is supported on Linux® operating system images. This procedure applies to Linux® platforms only.

If you are working with Docker, you must have an understanding of Docker container technology. Docker is based on open standards that allow Docker instances running on the same machine to share its operating system kernel but isolate applications from one another and from the underlying infrastructure. For more information see the Docker site.

**Install docker**

Follow this process to install docker on your Linux® machine and run the daemon.

1. Download and install docker, as shown in the example.

   ```bash
curl -fsSL https://get.docker.com/ | sh
   `` `

2. Start the docker daemon.

   ```bash
   sudo systemctl start docker
   ```
3. Verify that the docker daemon is running.

```
sudo systemctl status docker
```

4. Enable docker to start each time the system starts.

```
sudo systemctl enable docker
```

---

## Docker scripts

You can use scripts to help you work with Genero GAS docker images.

Sample scripts are available from the FourjsGenero docker_debian_fgl_gas GitHub repository. Table 42: Docker Debian files on page 301 lists the contents of the GitHub repository and describes the function of each file and script.

The scripts (.sh) allow you to work with a Debian® image to create a docker container running Genero Application Server (GAS) with Apache Web server. For more information on using the scripts, see the README.md file, or follow the procedures in the Docker Quick Start Guide on page 301. You are encouraged to customize the scripts to your needs.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>README.md</td>
<td>This file contains the steps to guide you through the process of creating a docker image and running Genero GAS in a docker container.</td>
</tr>
<tr>
<td>entrypoint.sh</td>
<td>This script file contains the commands needed to start Apache Web server and the GAS fastcgidispach dispatcher that runs automatically when the container starts.</td>
</tr>
<tr>
<td>docker_run_genero_image.sh</td>
<td>This short script file contains the command to run the docker container.</td>
</tr>
<tr>
<td>Dockerfile</td>
<td>The Dockerfile contains the commands needed to get the latest updates to build the Debian® image, and install the GAS and FGLGWS. The user (&quot;genero&quot;) and group (&quot;fourjs&quot;) are added. Settings for the Apache Web server are provided with login details for the genero user.</td>
</tr>
<tr>
<td>docker_build.genero_image.sh</td>
<td>This script builds the docker image with the FGLGWS and GAS package installation files. It provides the commands to set up the environment for GAS and Apache Web server.</td>
</tr>
<tr>
<td>000-default.conf</td>
<td>This file contains Apache Web server configuration settings for the GAS.</td>
</tr>
</tbody>
</table>

## Docker Quick Start Guide

Follow the tasks in this quick start guide to complete the process of building a docker image and running applications on a Genero Application Server in a docker container.

Step-by-step instructions are provided showing you how to use the scripts available from the FourjsGenero docker_debian_fgl_gas GitHub repository to perform these tasks.
Download installation packages

Download the latest installation files for Genero Business Development Language (BDL) and Genero Application Server (GAS) from the Four Js web site.

Download the packages to your working directory on your system. For help with downloading the appropriate packages for your Linux® distribution and processor type, see the Supported platforms and databases link in the Support page of the Four Js web site (http://www.4js.com/support/), or contact your Four Js support center.

1. Open a browser and navigate to the Four Js web site, www.4js.com.
2. Log in with your login details or register.
3. Select Download > Products. The Products download area page displays.
4. Expand the Genero BDL and Web Services (BDL) section. Download the required BDL version.
5. Expand the Genero Application Server (GAS) section. Download the required GAS version.

What to do next:
Configure license details for FLM server, see Configure license details on page 302

Configure license details

Provide details of the FLM server and your Genero product's license key and number in an FGLPROFILE file.

About this task: Licensing Genero products installed on a Docker container requires the use of a Four Js License Manager (FLM) server that is hosted outside the container. For information on installing a License Manager, see the Install Four Js License Manager chapter in the Install and License your Genero Products manual.

Note:
If you transfer the license from your Genero product installed on the host machine to a Genero product running in a Docker container, the FLM is not needed. See the License sharing with docker topics in the Install and License your Genero Products manual.

In your docker directory create an FGLPROFILE file with the following configuration:

```
flm.license.number="XXX#XXXXXXXX"
flm.license.key="XXXXXXXXXXXX"
flm.server=hostname
flm.service=port
```

Where:

- The license number and key are those required for a Genero BDL installation.
- The flm.server is the name or IP address of the machine that holds the license service program.

Note: If you are working in a virtual environment such as VMware or VirtualBox, you may have to provide the IP address of the host machine depending on how your network is set up and where your FLM is located.

What to do next:
Create a docker image with GAS and BDL installations and run in a Docker container. See Create image and run docker container on page 302.

Create image and run docker container

Build a docker image with a BDL and GAS installation and run an application in a docker container.

Before you begin:
You must have Docker installed and running. This task uses scripts available from the FourjsGenero
docker_debian_fgl_gas GitHub repository. It is assumed you have downloaded the latest version of the repository.

Tip: Refer to the readme.md file for the most up-to-date instructions. You will likely need to modify these steps accordingly.

1. Copy the BDL and GAS installation packages to your working directory.
2. Create environment variables.
   a) Create an environment variable to set to the BDL installation package. This will be used to copy the Genero BDL package to the image.
      For example:
      ```
      export FGLGWS_PACKAGE=fjs-fglgws-3.20.01-build1486651223-164x1212.run
      ```
   b) Create an environment variable to set to the GAS installation package. This will be used to copy the GAS package to the image.
      For example:
      ```
      export GAS_PACKAGE=fjs-gas-3.20.00-build154169-l64xl212.run
      ```
   c) Create an environment variable to configure access to the Apache Web server in the GAS configuration. Set the value for the host name or the IP address of your host machine. The default is localhost.
      ```
      export HOST_HOSTNAME=localhost
      ```
   d) Create an environment variable and set it to the Apache Web server port on your host machine. The default is 8080.
      ```
      export HOST_APACHE_PORT=8080
      ```
3. Build the docker image by running the `docker_build_genero_image.sh` script.
   A Genero GAS docker image is generated.
4. Start the docker container by running the `docker_run_genero_image.sh` script.
5. Open a browser and type the following URL in the address bar:
   ```
   http://localhost:8080/gas/ua/r/gwc-demo
   ```
   In the pop-up authentication window, enter the user name and password (same for both):
   ```
   gasadmin
   ```
   The demo application launches. You have successfully built a docker image with a GAS and Genero BDL installation and launched an application in a docker container.

### Docker commands

A list of useful docker commands.

<table>
<thead>
<tr>
<th>Table 43: Some Docker commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commands</strong></td>
</tr>
<tr>
<td>docker images</td>
</tr>
<tr>
<td>docker ps</td>
</tr>
<tr>
<td>docker stop genero</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 GAS

These topics provide an look back at the new features introduced with each release of the Genero Application Server (GAS).

GAS 3.20 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 3.20.

Important: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 3.20 upgrade guide on page 323.

Table 44: Single Sign-On (SSO) and delegation

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genero Identity Provider (GIP) integration to the GAS, ready to use with minimal settings. This will bring authentication and authorization mechanism to your apps.</td>
<td>See Identity Provider (IdP) on page 139.</td>
</tr>
</tbody>
</table>

The `DELEGATE` element provides a feature that allows a user to be logged out of the authentication server when a Web application is closed.

- For OpenID Connect two new parameters are added for this configuration, `IDP_LOGOUT_URL` and `SIGN_OFF`.
- For SAML authentication the log-out behavior is specified in the `SIGN_OFF` parameter.

See:

- `DELEGATE` on page 383
- Configure OpenID Connect SSO log out on page 147
- Configure SAML SSO log out on page 161
The **DELEGATE** element provides support for OAuth2 SSO authentication as used by identity providers (IdP) such as Facebook and Instagram.

- There are enhancements to the ImportOAuth tool command. It supports parsing of its command-line arguments with getopt.

The OpenID Connect service has three possible ways of authentication redirect: via an HTML submit form using GET or Post, or using the default HTTP 302.

The GetToken, DeployGar, and DeployGbc are command line tools for working with the GIP. They follow the FGL default for command options using getopt.

In an enhancement for GAS v3.20.14, the command tools are run from a script that can be set in the PATH environment.

The ImportIdP tool for managing SAML identity providers supports parsing of its command-line arguments with getopt.

### Table 45: Web Services and the GAS

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>With <strong>DELEGATE_OPTIONS</strong> you can specify whether to send the body or just HTTP headers for a service using delegation.</td>
<td>See <strong>DELEGATE_OPTIONS (for a service)</strong> on page 384</td>
</tr>
</tbody>
</table>

### Table 46: Engine and Architecture

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The dispatcher configuration is enhanced to listen for incoming requests on a dedicated IP address. The address is specified by the <strong>LISTEN</strong> element.</td>
<td>See <strong>LISTEN</strong> on page 402</td>
</tr>
<tr>
<td>With <strong>TCP_ADMIN_PORT</strong> you can specify a port for GAS administration tasks.</td>
<td>See <strong>TCP_ADMIN_PORT</strong> on page 429</td>
</tr>
<tr>
<td>With <strong>END_URL</strong> you can specify a URL that the user agent redirects to when your Web application ends.</td>
<td>See <strong>END_URL</strong> on page 388</td>
</tr>
<tr>
<td>The <strong>CACHE_CONTROL_MAX_AGE</strong> element allows you to specify the duration files sent by the GAS are held in front-end cache.</td>
<td>See <strong>CACHE_CONTROL_MAX_AGE</strong> on page 378</td>
</tr>
<tr>
<td>The <strong>ENVIRONMENT_VARIABLE</strong> element has a Concat attribute, which allows you to manage how inherited parent configuration settings are handled; appended, prepended, or discarded.</td>
<td>See <strong>ENVIRONMENT_VARIABLE</strong> on page 388</td>
</tr>
<tr>
<td>In an enhancement for GAS v3.20.14, dispatchers create a pid file. You can set the httpdispatch and fastcgidispatch dispatchers to write their process-ids to a file on disk with the --pid-file option of the command.</td>
<td>See <strong>Dispatcher: httpdispatch</strong> on page 344, <strong>Dispatcher: fastcgidispatch</strong> on page 345</td>
</tr>
</tbody>
</table>

The following gasadmin commands are enhanced with list options:

- `gasadmin config --list` lists all applications and services (not just the deployed ones) found in the GAS.
- `gasadmin gbc --list` lists also all static and deployed GBC found.
The `gasadmin session` command is enhanced with the `--close-session` and `--close-all-sessions` options. This provides for closing sessions gracefully, without displaying messages to the user agent.

In an enhancement for GAS v3.20.14, the `$(res.dir.separator)` resource is added. It allows you to specify directory paths for all platforms in the same configuration. It resolves to either a forward slash "/" on a UNIX™-like system including macOS™ or a backward slash "\" on a Windows® system.

### Table 47: Deployment

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The GAS allows you to provide your GBC client in the application path's</td>
<td>See <a href="#">Provide GBC in application path</a> on page 262.</td>
</tr>
<tr>
<td><code>gbc</code> directory.</td>
<td></td>
</tr>
<tr>
<td>The GIP DeploymentApp provides you with an interface to manage Genero</td>
<td>See <a href="#">Deploying and securing</a> applications and Web services on page 236 and <a href="#">Set default GBC client with the Genero Deployment App</a> on page 265</td>
</tr>
<tr>
<td>Archives, and Genero Browser Clients deployed on the Genero Application</td>
<td></td>
</tr>
<tr>
<td>Server (GAS). It replaces the legacy deployment portal web service.</td>
<td></td>
</tr>
<tr>
<td>• There are enhancements to the GBC deployment page to display a list of</td>
<td></td>
</tr>
<tr>
<td>static GBC found via the <code>as.xcf GBC_LOOKUP_PATH</code>.</td>
<td></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.

### GAS 3.10 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 3.10.

**Important:** This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: [GAS 3.10 upgrade guide](#) on page 325.

### Table 48: General

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Genero Web Client for JavaScript (GWC-JS) is renamed to Genero</td>
<td></td>
</tr>
<tr>
<td>Browser Client (GBC). All documentation is updated to use the new name.</td>
<td>No additional reference.</td>
</tr>
<tr>
<td>GBC is no longer embedded in the Genero Application Server (GAS). It is</td>
<td>See <a href="#">Genero Browser Client User Guide</a>.</td>
</tr>
<tr>
<td>now delivered as its own module in the Genero Enterprise bundle. It is</td>
<td></td>
</tr>
<tr>
<td>also documented in a separate manual.</td>
<td></td>
</tr>
</tbody>
</table>

### Table 49: Docker and the GAS

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The GAS provides support for creating a Docker image to run the GAS in a</td>
<td>See <a href="#">Run Genero Application Server in Docker container</a> on page 300.</td>
</tr>
<tr>
<td>Docker container.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** In `$(FGLASDIR)/docker`, follow the instructions in the Readme.md to build and run a docker container.
Table 50: Single Sign-On (SSO) and delegation

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a new method for managing environment variables defined in delegation parameters and passed in HTTP headers to the REST service. Environment variable names are converted to uppercase before being set in the DVM environment.</td>
<td>See DELEGATE on page 383 and From the REST service to the proxy on page 132</td>
</tr>
<tr>
<td>The PROMPT element provides a feature that allows a user to re-log in to an application after an auto logout event.</td>
<td>See Re-login prompt on page 54 and Authorize re-log in with OpenID Connect SSO on page 147 and Authorize re-log in with SAML SSO on page 160</td>
</tr>
<tr>
<td><strong>Note:</strong> Delegation Web services for SSO OpenID Connect and SAML are provided in the FGLGWS package. These support the PROMPT (for auto logout) on page 412 feature.</td>
<td></td>
</tr>
</tbody>
</table>

Table 51: Web Services and the GAS

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>In your Web services configuration files (.xcf) a reference to a parent configuration in the APPLICATION element’s Parent attribute is required. This reference must contain the prefix “ws.”.</td>
<td>See APPLICATION (for a service) on page 373 and Configure applications for Web service on page 116</td>
</tr>
</tbody>
</table>

Table 52: Engine and Architecture

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Genero Application Server for Java (JGAS) is an implementation of the Genero Application Server (GAS) that is written in Java. It can be used for the development, testing, and deployment of applications to a Java Enterprise Edition (Java EE) server. It is also documented in a separate manual.</td>
<td>See Genero Application Server for Java User Guide</td>
</tr>
<tr>
<td>The j2eedispatch dispatcher is being replaced by the JGAS.</td>
<td>The j2eedispatch supporting topics have been removed. Please see the Java Servlet Installation and Web Server Configuration topics in the Genero Application Server 3.00 User Guide.</td>
</tr>
<tr>
<td>A new element called REPORT_REMOTE_URL_PREFIX added to INTERFACE_TO_CONNECTOR allows you to configure the URL of the Genero Report Engine server.</td>
<td>See REPORT_REMOTE_URL_PREFIX on page 416, Configuring Genero Report Engine on GAS on page 173</td>
</tr>
<tr>
<td>The desupported proxies gdcproxy and html5proxy are no longer provided in the package. They are replaced by the universal proxy, uaproyxy (ua), which is used for all applications.</td>
<td>The supporting topics have been removed. Please see the Tools and Commands topics in the Genero Application Server 2.00 User Guide.</td>
</tr>
<tr>
<td>In an enhancement for GAS v3.10.16, dispatchers create a pid file. You can set the httpdispatch and fastcgidispatch dispatchers to write their process-ids to a file on disk with the --pid-file option of the command.</td>
<td>See • Dispatcher: httpdispatch on page 344 • Dispatcher: fastcgidispatch on page 345</td>
</tr>
</tbody>
</table>
Overview

The options of the GAS tool named gasadmin have been reorganized into commands: session, config, gar, gbc, reset-log.

- The gbc command adds functions for working with GBC clients.
- The reset-log provides options to reconfigure logs for a proxy session; such as change the output type, or output path.

The --session-cleanup option of the gasadmin tool has been enhanced to clean temporary files/directories created, that may not have been removed at the end of a session.

The --monitor option of the gasadmin tool allows you to retrieve information to monitor a session.

Specifying a GBC front-end customization has been simplified. The GBC element may contain the name of a file that specifies the customization directory. The GBC can also be specified in a URI query string parameter to override the application configuration.

The GWC_JS_LOOKUP_PATH element is renamed to GBC_LOOKUP_PATH. Both names continue to be supported.

The GWC-JS element is renamed to GBC. GWC-JS is deprecated. Both elements continue to be supported.

The environment variable FGL_VMPROXY_START_URL is set to the URL used to start the application, which is available for use by the Genero BDL application.

Table 53: Deployment

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The fglgar tool has new features for packaging and deploying Web applications. It is no longer provided in the GAS package. It is now delivered in the FGLGWS package.</td>
<td>See the fglgar topic in Genero Business Development Language User Guide.</td>
</tr>
<tr>
<td>The res.gbc.deployment resource points to $(res.appdata.path)/gbc_deployment. It contains the deployed GBC clients and is part of the default GBC_LOOKUP_PATH on page 395.</td>
<td>See GAS installation and application data directories on page 40 and the gasadmin gbc command options in gasadmin tool on page 348.</td>
</tr>
</tbody>
</table>

A deployment portal is provided to deploy GBC and Genero Archives via an HTML web page. It is accessible at URL http://host:port/ws/r/services/DeploymentService/html.

The GBC can be reset to the initial installed version delivered in the FGLGWS package with the gasadmin gbc command and the --reset option. A reset can also be carried out via the deployment portal.

See:  
- gasadmin tool on page 348  
- Reset GBC client to initial installed version on page 266  
- See the Genero deployment portal topic in Genero Application Server User Guide 3.10.
Upgrading

Overview
A GBC client deployed on the GAS can be renamed with the `gasadmin gbc` command and the `--rename` option. You can also use the deployment portal interface to rename the GBC.

Reference
See:
- `gasadmin tool` on page 348
- See the `Genero deployment portal` topic in `Genero Application Server User Guide 3.10`.

### Table 54: Miscellaneous

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The desupported <code>fglxslp</code> migration tool is no longer provided in the package.</td>
<td>All supporting topics have been removed.</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.

### GAS 3.00 new features

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

**Important:** This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: `GAS 3.00 upgrade guide` on page 328.

### Table 55: GWC

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
</table>
| Genero Web Client for JavaScript (GWC-JS) is introduced as the new Web client for developing Genero Web Client applications. In later versions, it will be renamed Genero Browser Client (GBC). | See the `Genero Browser Client User Guide`.

### Table 56: Single Sign-On

<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.</td>
<td>See <code>What is Single sign-on (SSO)</code>? on page 55.</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 <code>$FGLDIR/web_utilities/services/openidconnect</code>.</td>
<td>See <code>OpenID Connect SSO</code> on page 142</td>
</tr>
</tbody>
</table>

### Table 57: Web Services and the GAS

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>A new element called <code>REQUEST_RESULT</code> 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 <code>SERVICE_APPLICATION_TIMEOUT_COMPONENT</code> on page 422</td>
</tr>
</tbody>
</table>
Table 58: Engine and Architecture

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>gasadmin</code> command has been updated with new features to manage archives. You can list, enable, deploy and undeploy archives with options of the <code>gasadmin</code> command.</td>
<td>See <code>gasadmin</code> tool on page 348.</td>
</tr>
<tr>
<td>The GAS supports a new timeout feature called <code>AUTO_LOGOUT_COMPONENT</code> 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 <code>AUTO_LOGOUT_COMPONENT</code> on page 377</td>
</tr>
<tr>
<td>GAS 3.00 introduces a new universal proxy for applications using GDC v3 and GWC-JS interfaces. It is called <code>uaproxy</code> (ua). It replaces the <code>gdcproxy</code> and <code>html5proxy</code> proxies. It provides protocol improvements and better performance overall.</td>
<td>See <code>Proxy: uaproxy</code> on page 347</td>
</tr>
<tr>
<td>There is a new bootstrap mechanism for starting GWC-JS applications, which is used to initialize information for rendering an application.</td>
<td>See the Genero Browser Client User Guide.</td>
</tr>
<tr>
<td>A new element called <code>GWC_JS_LOOKUP_PATH</code> added to <code>INTERFACE_TO_CONNECTOR</code> allows you to configure the location of your custom GWC-JS front-end.</td>
<td>See <code>GBC_LOOKUP_PATH</code> on page 395</td>
</tr>
<tr>
<td>The new user agent protocol, the <code>uaproxy</code> for GDC, GMA, GMI and GWC-JS, introduces a new set of resource URLs. The <code>ua</code> protocol does not use snippets sets or output maps as the <code>wa</code> protocol did previously to specify output theme.</td>
<td>See <code>Proxy</code> on page 39</td>
</tr>
<tr>
<td>Genero Ghost Client is a new Java framework introduced for testing different scenarios by emulating user interaction on Genero applications.</td>
<td>Documentation about the Genero Ghost Client has been moved to the Genero Ghost Client User Guide.</td>
</tr>
<tr>
<td>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 <code>/ua/grv</code> is provided to the Genero Report Engine (GRE).</td>
<td>See <code>REPORT_VIEWER_DIRECTORY</code> on page 416</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>No additional reference.</td>
</tr>
<tr>
<td>Enhancements for the GAS (v3.00.21):</td>
<td>See <code>HTTP</code> on page 399, <code>SESSION_COOKIE</code> on page 425</td>
</tr>
<tr>
<td>• A method to add custom HTTP headers for HTTP responses is provided.</td>
<td></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>
</tbody>
</table>

Table 59: Deployment

<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>The supporting topic has been removed, as the fglgar tool is now delivered in the FGLGWS package. Please see the <code>fglgar</code> topic in the Genero Application Server 3.00 User Guide and in the Genero Business Development Language User Guide.</td>
</tr>
</tbody>
</table>
Overview

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)
- A resource where the Application Server stores files temporarily during file transfer, $(res.path.tmp)

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.

The DOCUMENT_ROOT allows multiple document root paths.

New entries in the application MANIFEST file:

- 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.
- A new TRIGGERS element defines a set of deployment parameters that can be used when deploying an application with the deployment framework.

Table 60: Miscellaneous

Overview

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.

In addition to specifying access by IP address, the ACCESS_CONTROL and MONITOR configuration elements have been updated with two access control keywords (NOBODY, ALL) which can be used with ALLOW_FROM.

Introducing a new resource, res.access.control, that specifies access control for Web services, application programs, such as demos, and MONITOR. It is defined with the keyword NOBODY by default.

A new element called ROOT_URL_PREFIX added to INTERFACE_TO_CONNECTOR 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.

Reference

See GAS installation and application data directories on page 40 and Paths to application resources on page 277

See WEB_COMPONENT_DIRECTORY on page 440

See DOCUMENT_ROOT on page 386

See MANIFEST file on page 280 and TRIGGERS (for manifest) on page 281

Note: If you have to restrict access of some applications to specific IP addresses, in that case the ACCESS_CONTROL entry must be configured with IPv6 addresses. See ACCESS_CONTROL on page 367

See ALLOW_FROM on page 370

Note: The default deployment does not allow access to demo programs and MONITOR. Access has to be configured explicitly with ALLOW_FROM. See MONITOR on page 405 and ACCESS_CONTROL on page 367

See ROOT_URL_PREFIX on page 421
GAS 2.50 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.50.

Important: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 2.50 upgrade guide on page 331.

Table 61: GWC for HTML5

<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: Cascading Style Sheets, HTML5 theme.</td>
</tr>
<tr>
<td>The HTML5 theme adds support for frozen columns, splitters (in forms, between the form and the actionPanel, between the form and the startMenu), and GridChildrenInParent.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see Html5 Theme topics in Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>The html5proxy manages Genero web client applications using the HTML5 theme.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see Html5 Theme topics in the Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>The HTML5 theme supports StartMenus and applications displayed in folder tabs.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Enable the StartMenu and applications in folder tabs, in the Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>Add language support to a snippet file (HTML5).</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Translations in the snippets, in the Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>Start an application while ignoring the application's stored settings (HTML5).</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Start an application without stored settings (GWC for HTML5), in the Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>GDCAX is deprecated in favor of the HTML5 theme. GDC HTTP is still supported.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topics, Html5 Theme, in Genero Application Server 2.50 User Guide.</td>
</tr>
<tr>
<td>Control the folder size of an application rendered by the HTML5 client.</td>
<td>The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Control the folder size, in Genero Application Server 2.50 User Guide.</td>
</tr>
</tbody>
</table>
**Overview**

UI enhancements for the GWC (HTML5 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>
<thead>
<tr>
<th>Table 62: Single Sign-On</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong></td>
</tr>
<tr>
<td>Kerberos is deprecated. Promoted SSO solutions are OpenID and SAML.</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 and is delivered in the Genero Web Services package under $FGLDIR/web_utilities/services/saml.</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 63: Web Services and the GAS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong></td>
</tr>
<tr>
<td>Support for sticky Web services.</td>
</tr>
<tr>
<td>Service invalidation provides notification when a Genero Web Service configuration is invalid. Once identified as having invalid configuration, the dispatcher will not attempt to start the gwsproxy for the service until the configuration is modified.</td>
</tr>
</tbody>
</table>
Table 64: Engine and Architecture

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better internal/temporary data organization.</td>
<td>See GAS installation and application data directories on page 40.</td>
</tr>
<tr>
<td>The gasadmin command has been updated to handle multiple dispatchers.</td>
<td>See gasadmin tool on page 348.</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 files.</td>
<td>See Log files on page 185.</td>
</tr>
<tr>
<td>The Genero Application Server on Linux®/UNIX™ will use domain sockets to</td>
<td>See SOCKET_FAMILY on page 427 and SOCKET_PATH on page 428.</td>
</tr>
<tr>
<td>communicate between the dispatcher and the proxies.</td>
<td></td>
</tr>
<tr>
<td>Enable and disable resource compression in the imt.cfg file. Compress</td>
<td>See Compression in Genero Application Server on page 169 and gasadmin tool</td>
</tr>
<tr>
<td>resources (static files) using the -z option of the gasadmin command.</td>
<td>on page 348.</td>
</tr>
</tbody>
</table>

Table 65: Deployment

<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 apps with Genero Archive on page 270.</td>
</tr>
</tbody>
</table>

Table 66: Miscellaneous

<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>

GAS 2.41 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.41.

Important: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 2.41 upgrade guide on page 332.

Table 67: General

<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,</td>
<td>The supporting topic has been removed, as the Silverlight theme is</td>
</tr>
<tr>
<td>the default behavior is to have a left and right button on the toolbar</td>
<td>deprecated. Please see the Genero Application Server 2.41 User Guide.</td>
</tr>
<tr>
<td>(horizontal scrolling).</td>
<td></td>
</tr>
</tbody>
</table>
### Overview

**GWC for HTML5** is a supported theme. It is no longer considered a “preview version”.

**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.

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.

The GWC hybrid mode takes Genero applications and delivers them as native applications for Android™ or iOS mobile platforms.

New template paths: Application hierarchy, Window hierarchy, Layout hierarchies, and Widgets hierarchies have been added.

---

### Reference

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*](#).

No additional reference.

The supporting topic has been removed, as GWC hybrid mode is deprecated. Please see the topic, *Genero Web Client hybrid mode*, in the [*Genero Application Server 2.41 User Guide*](#).

The supporting topics have been removed, as the template paths theme is deprecated. Please see the topics:

- Template Paths - Application hierarchy
- Template Paths - Window hierarchy
- Template Paths - Layout hierarchies
- Template Paths - Widgets hierarchies

in the [*Genero Application Server 2.41 User Guide*](#).

---

### GAS 2.40 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.40.

**Important**: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: [*GAS 2.40 upgrade guide*](#) on page 332.

### Table 68: General

<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 <a href="#"><em>Genero Application Server 2.40 User Guide</em></a>.</td>
</tr>
<tr>
<td>The Genero Application Server can handle submit parameters with the POST method.</td>
<td>See <a href="#"><em>Handling POST Method Submit Parameters</em></a> in the <a href="#"><em>Genero Application Server 2.40 User Guide</em></a>.</td>
</tr>
<tr>
<td>Performance improvements on the GAS have been made to provide users with a better experience using Genero Web Applications.</td>
<td>No additional reference.</td>
</tr>
<tr>
<td>• Less memory consumed for GWC rendering</td>
<td></td>
</tr>
<tr>
<td>• Less data traffic for tables and windows</td>
<td></td>
</tr>
<tr>
<td>• Better stack management in threads</td>
<td></td>
</tr>
</tbody>
</table>
Table 69: Silverlight theme

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<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>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>• localAccelerators</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>• imageCache</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>• showAcceleratorInToolTip</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, 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>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 text indentation</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 local action binding</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>• xHTML is mandatory</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>• New lines in paragraph are replaced by spaces</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 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. See What is the Type Ahead mechanism? in the Genero Application Server 2.40 User Guide.</td>
</tr>
</tbody>
</table>

Table 70: 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>
<tr>
<td><strong>Important:</strong> 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).</td>
<td></td>
</tr>
</tbody>
</table>
GAS 2.32 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.32.

**Important:** This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: None.

**Table 71: General**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Genero Application Server administration tool <code>gasadmin</code> is provided. This tool is a command to list, ping and kill applications sessions.</td>
<td>See <code>gasadmin tool</code> on page 348.</td>
</tr>
<tr>
<td><code>MAX_REQUESTS_PER_DVM</code> 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 <code>MAX_REQUESTS_PER_DVM</code> on page 405.</td>
</tr>
<tr>
<td>A new log output allows logging both in a directory and to the console.</td>
<td>See <code>LOG</code> on page 403.</td>
</tr>
</tbody>
</table>

GAS 2.30 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.30.

**Important:** This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 2.30 upgrade guide on page 334.

**Table 72: 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 Dispatcher on page 38</td>
</tr>
<tr>
<td>Java Application Server integration: The GAS server can be fully integrated into any J2EE Servlet container using the GWC dispatcher <code>j2eedispatch</code>.</td>
<td>The supporting topic has been removed, as dispatcher <code>j2eedispatch</code> 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 176.</td>
</tr>
<tr>
<td>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.</td>
<td>See LOG on page 403.</td>
</tr>
</tbody>
</table>
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 34

Table 73: GWC themes

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.30 User Guide.</td>
</tr>
<tr>
<td>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.</td>
<td>The supporting topic has been removed, as the iPhone® theme is deprecated. Please see the Genero Application Server 2.30 User Guide.</td>
</tr>
<tr>
<td>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®.</td>
<td>The supporting topic has been removed, as the AJAX theme is deprecated. Please see the Genero Application Server 2.30 User Guide.</td>
</tr>
</tbody>
</table>

Table 74: Miscellaneous

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWC for AJAX and GWC for Silverlight support drag-and-drop.</td>
<td>The supporting topics have been removed, as the AJAX and Silverlight themes are deprecated. Please see the Genero Application Server 2.30 User Guide.</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Note:</td>
<td>See URI Examples on page 48</td>
</tr>
<tr>
<td>• On Mac, you need to manually add the .gdc association for Safari® or save the .gdc and double click to launch the application</td>
<td></td>
</tr>
<tr>
<td>• Currently, any access to /da/r url is handled by DUA_GDC map, no matter which OutputMap argument is set in the url</td>
<td></td>
</tr>
<tr>
<td>The ACCESS_CONTROL element specifies which IP addresses are allowed to execute an application or web service. By default, all IP addresses are allowed.</td>
<td></td>
</tr>
<tr>
<td>The new resource configuration.filepath provides the absolute path to the GAS configuration file.</td>
<td>See ACCESS_CONTROL on page 367.</td>
</tr>
<tr>
<td>See Common predefined resources on page 359.</td>
<td></td>
</tr>
</tbody>
</table>
Overview | Reference
---|---
New topics have been introduced to the documentation about each of the following themes (snippet sets): | The supporting topics have been removed, as the AJAX, Silverlight, Basic and iPhone® themes are deprecated. Please see the Genero Application Server 2.30 User Guide.
  • 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.

GAS 2.22 new features
A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.22.

Important: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 2.22 upgrade guide on page 336.

Table 75: General

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications started by a StartMenu can be rendered in folder tabs.</td>
<td>See Enable the StartMenu and applications in folder tabs in the Genero Application Server 2.22 User Guide.</td>
</tr>
<tr>
<td>Textedit with textFormat html 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 in the Genero Application Server 2.22 User Guide.</td>
</tr>
</tbody>
</table>

GAS 2.21 new features
A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.21.

Important: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 2.21 upgrade guide on page 337.
Table 76: 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 77: Genero Web Client

<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 44</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: SpinEdit/minValue SpinEdit/maxValue in the Genero Application Server 2.21 User Guide.</td>
</tr>
<tr>
<td>Applications in configuration files support both a short description and a long description.</td>
<td>See DESCRIPTION on page 385</td>
</tr>
</tbody>
</table>
GAS 2.20 new features

A summary of new features and changes in functionality introduced with Genero Application Server (GAS) 2.20.

Important: This page covers only those new features introduced with the Genero Application Server version specified in the page title. Check prior new features pages if you migrate from an earlier version. Make sure to also read the upgrade guide corresponding to this Genero version.

Corresponding upgrade guide: GAS 2.20 upgrade guide on page 337.

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 78: Application Server

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>The architecture of the Genero Application Server has since changed. See Architecture of the Genero Application Server on page 30.</td>
</tr>
<tr>
<td>The following improvements have been made in GAS 2.20:</td>
<td>The architecture of the Genero Application Server has since changed. See Architecture of the Genero Application Server on page 30.</td>
</tr>
<tr>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td>The GAS gives access to server and applications details through the &quot;/monitor&quot; URL. The monitor page exposes details about DVMs, Web applications, Web services, GDC applications, and GAS activity.</td>
<td>See Monitoring on page 176.</td>
</tr>
</tbody>
</table>

Table 79: GDCAX management

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 80: Web application management (Genero Web Client)

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree views are fully supported by the Genero Web Client.</td>
<td>The supporting topic has been removed, as the HTML5 theme is deprecated. See the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of what the HTML5 theme supports.</td>
</tr>
</tbody>
</table>
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 front call functions (launchUrl and feinfo/screenresolution) which are fully supported by the GWC.

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.

The supporting topic has been removed, as the XUL theme is deprecated. Please see the Genero Application Server 2.20 User Guide.

The front call 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.

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. See the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of what the HTML5 theme supports.

The supporting topic has been removed, as the HTML5 theme is deprecated. See the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of what the HTML5 theme supports.

The supporting topic has been removed, as the HTML5 theme is deprecated. See the JavaScript topic in the Genero Application Server 2.20 User Guide for a complete listing of what the HTML5 theme supports.

No additional reference.

**Upgrade Guides for the GAS**

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.

**Important:** 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.

**Migrate your configuration files**

Migrating your configuration (*.cf*) files involves more than simply copying existing files to the latest directory.

Configuration files may include updates or the addition of new elements that will not be included if you simply copy your files.

1. Do a `diff` between your existing configuration file and the previous version's standard configuration file. This creates a list of the additions you made to the standard configuration file.
2. Apply the differences to the latest standard configuration file.
3. Save your changes using the appropriate file name.
GAS 3.20 upgrade guide

These topics describe product changes you must be aware of when upgrading to version 3.20.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 3.20 new features on page 9.

**TCP_ADMIN_PORT**

TCP_ADMIN_PORT on page 429 is a new element in the GAS configuration file. If your GAS version is 3.20.07 or later, you need to be aware of this new element. It provides a port for GAS administration tasks. If you are using an as.xcf without this element, there is no need to add it. The default TCP admin port is set internally to 6999. If you need to set the port, you may need to add the element to the as.xcf.

It is necessary to set this port, for example, if you have several GAS running on the same machine. Then you need to ensure that you set a unique port for each GAS. Also ensure you select a different port to that used for application access. For more information on configuring multiple dispatchers for TCP admin port, see Configure multiple dispatchers on page 171.

**GBC in application path**

The means to provide a Genero Browser Client (GBC) in the application path is introduced. See Provide GBC in application path on page 262.

With 3.20, you can package a GBC in the gbc directory of the application path. This is the path specified by the PATH element in the application xcf file. The GAS first looks for the GBC in APPDIR/gbc. If this directory exists and it contains a GBC installation, this GBC is used. If the gbc directory is not set, the GAS then uses the standard GBC_LOOKUP_PATH on page 395 lookup mechanism.

What does this mean for your upgrade to 3.20? It allows you to provide a GBC to use in the application path's gbc directory independent of the GAS configuration.

**Deployment portal has been replaced with DeploymentApp**

The legacy deployment portal service has been replaced with the Genero Deployment App (DeploymentApp). If you used the deployment portal to deploy and manage your applications and GBC clients, you will need to move to using this application. For information on how to access and use it, see Deploying and securing applications and Web services on page 236.

**Changes to the deployment service under GIP**

The deployment service is now protected by Genero Identity Provider (GIP) on page 204. To access it requires a valid access token with the scope "deployment". The service URL has changed too. If you previously used the deployment service on the GAS at base URL /ws/r/services/DeploymentService to manage your Genero applications, you will need to use this new URL /ws/r/admin/GeneroDeploymentService instead. For more information on how to access and use it, see Deploying and securing applications and Web services on page 236.

**Changes to ImportOAuth tool for managing OpenID Connect identity providers (IdP)**

The ImportOAuth tool command supports getopt parsing of its command-line arguments. The use of the tool to manage IdP for your Genero applications has therefore changed. For examples of changes you need to make, see Table 81: Example change to ImportOAuth import command on page 324. For help using the tool, run the command:

```
fglrun ImportOAuth -h
```
For further details about the ImportOAuth command, see The ImportOAuth program on page 149.

**Table 81: Example change to ImportOAuth import command**

<table>
<thead>
<tr>
<th>New command</th>
<th>Previous command</th>
</tr>
</thead>
</table>

**Changes to ImportIdP tool for managing SAML identity providers**

The ImportIdP tool command supports `getopt` parsing of its command-line arguments. The use of the tool to manage IdP for your Genero applications has therefore changed. For examples of changes you need to make, see Table 82: Example change to ImportIdP import command on page 324. For help using the tool, run the command:

```bash
fglrn ImportIdP -h
```

For further details about the ImportIdP command, see The ImportIdP program on page 154.

**Table 82: Example change to ImportIdP import command**

<table>
<thead>
<tr>
<th>New command</th>
<th>Previous command</th>
</tr>
</thead>
</table>

**Changes to support for file transfer**

Starting with FGLGWS 3.20.11 and GAS 3.20.13 there is full support for file transfer with filenames set on any locale. This update allows you to specify filenames with locale-specific characters in your application file transfers.

**Important: Apache**

If you have configured mod_proxy_fcgi in your Apache server, ensure application URLs are not escaped by setting `proxy-fcgi-pathinfo=unescape`. For more details, see Apache 2.4: mod_proxy_fcgi on page 100.

If you are using any older combinations of FGLGWS and GAS, file transfer only works for filenames with ASCII characters.
Enhancement for GIP command line tools

Starting with FGLGWS version 3.20.11, GAS version 3.20.14, and GIP version 1.00.11 there are changes to how you use GetToken on page 251, DeployGar on page 253, and DeployGbc on page 256. These commands are now run from a script. For example, if previously you ran the "GetToken" command as:

```
fglrun $FGLDIR/web_utilities/services/gip/bin/gettoken/GetToken.42r
client_credentials --help
```

You can now run it directly as:

```
GetToken client_credentials --help
```

GetToken is located in $FGLDIR/web_utilities/services/gip/bin/gettoken. DeployGar and DeployGbc are located in the $FGLDIR/web_utilities/services/gip/bin/deploy directory. You need to include these paths in your PATH environment variable to run the commands from any directory.

Tip: Run the script file $FGLDIR/web_utilities/services/gip/envidp.sh to ensure the PATH is set correctly to run the commands.

SESSION_COOKIE definition

Starting with GAS version 3.20.14, the SESSION_COOKIE element includes the SameSite attribute.

If you are migrating SESSION_COOKIE elements, you may need to set the SameSite attribute. Typically, to avoid cross-site domain errors, you set attribute SameSite=None and ensure all requests are secured and in HTTPS.

For more information see SESSION_COOKIE on page 425.

GAS 3.10 upgrade guide

These topics describe product changes you must be aware of when upgrading to version 3.10.

Important: This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 3.10 new features on page 306.

Delegation and environment variables

Starting with GAS version 3.10, all environment variables defined via X-FourJs-Environment- HTTP headers are forced to uppercase. If you use delegation in your applications or Web services, make sure that you are not relying on case (for example title or camel case, "MyEnvVar") in the naming of parameters as there is no guarantee that it will be preserved.

Some Web servers convert all header names to lowercase, which is outside your control. To provide a consistency in use across platforms (Windows® and UNIX®), proxies parse the HTTP headers and convert all environment variables to uppercase.

HTTP security header: X-Content-Type-Options="nosniff"

Since GAS versions 3.00.35 and GAS 3.10 (GA version), for security reasons the default header "X-Content-Type-Options" needs to be set to the value "nosniff". This is the default use in the GAS configuration file (as.xcf):

```
<HEADER Name="X-Content-Type-Options">nosniff</HEADER>
```

Use of this header blocks browsers sniffing file types, which is guessing the correct Multipurpose Internet Mail Extensions (MIME) type by looking at the file. The header blocks sniffing of script and style type files.
Warning: On Internet Explorer 11, use of this header may cause images not to be displayed if the image files do not have extensions. To work around this, if your applications need to serve images through Genero Application Server (GAS), make sure your image files have extensions.

For more information, see HEADER on page 398.

Delegation and URLs

To avoid query decoding issues, the delegate mechanism has been changed to follow URL RFCs by passing the delegate URL as a standard QueryString.

See How to implement delegation on page 124 for more information.

OpenID Single sign-on (SSO) protocol not supported

The OpenID authentication mechanism for Single Sign-on authentication is deprecated. If you have previously used OpenID to authenticate users launching applications, you must now use an alternate method, such as OpenID Connect or SAML. Any new development requiring Single Sign-on should plan to use OpenID Connect, or SAML. See How to implement Single sign-on (SSO) on page 139. For alternative solutions, please contact your Four Js support center.

Sample SSO implementation (simplesso) located on GitHub

The sample implementation of Single Sign On (SSO) for Genero is no longer provided in the FGLGWS installation at $FGLDIR/web_utilities/services/simplesso, instead it can now be found under the Four Js Genero GitHub repository. See https://github.com/FourjsGenero/ex_simplesso.

What does this mean for your upgrade to 3.10? If you had previously used the simplesso mechanism for your SSO service, you will need to download it from GitHub. For complete implementation instructions and its use, please see the README file in the download. For more information, see Delegation use cases on page 136 and SSO custom sample (simplesso) on page 165.

Parent application for a Web Service must start with "ws."

To support Genero Archive packaging, all parent applications defined in the configuration of Web Services applications must now start with "ws." See Configure applications for Web service on page 116.

Desupported GAS configuration elements

The following elements are no longer allowed or used:

- The DVM_PINGTIMEOUT, child element of the TIMEOUT (for an application) on page 431 is not supported. The GAS no longer sends a ping event when the DVM is running with GAS. No action needs to be taken.

  Note: If you have set the gui.protocol.pingTimeout in your FGLPROFILE, this setting is not used when the DVM runs with the GAS. For applications running on Genero Desktop Client (GDC) in direct mode (without GAS), the ping event is still part of the protocol and the GDC keeps sending it as usual.

- The FILE_TRANSFER element and its child element, TIMEOUT, are not supported.

- The OUTPUT element which specified the parameters for rendering GWC for HTML5 Web applications is no longer supported. Starting with Genero 3.10, all UI applications should use the UA proxy instead, which is configured by the UA_OUTPUT on page 434 element.

Desupported proxies

The following proxies are no longer supported:

- The gdcproxy proxy used for GDC is no longer supported. The uaproyxy proxy is now used for GDC.

- The html5proxy proxy for Genero Web Client for HTML5 (GWC-HTML5) is no longer supported. This version of the Genero web client is no longer supported and has been replaced by the Genero Browser Client (GBC) instead. The uaproyxy proxy is used for GBC.
Desupported session variables

Session variables are no longer managed in the GAS when using UAProxy. If before you used session variables to maintain states between runs of an application by means of configuring the HTTP_COOKIES element in the as.xcf file, this is no longer utilized and you will need to remove those entries.

In your 4GL code session state is set through "localStorage" front calls. Starting with version 3.10, you must use the new localStorage front calls, supported by all Genero front-ends, to dynamically set and get session state from within your Genero application. For example:

```plaintext
ui.Interface.frontCall("localStorage", "setItem", [key,value], [])
```

Deprecated session front calls

The session module front calls setVar and getVar are deprecated. You will need to replace these calls with calls to "localStorage". For more information, see the Local storage front calls section in Genero Business Development Language User Guide.

Alias GAS configuration elements: GWC-JS and GBC

The latest GAS version configuration file shows the following element changes:

- **GBC** is now the preferred element for Genero Browser Client customization. **GWC-JS** is deprecated.
- **GBC_LOOKUP_PATH** on page 395 is now the preferred element for GBC customization look up. **GWC_JS_LOOKUP_PATH** is deprecated.

Whilst GWC-JS and GWC_JS_LOOKUP_PATH elements continue to be supported, we recommend you take the opportunity at this time to replace the deprecated elements (GWC-JS, GWC_JS_LOOKUP_PATH) in your application configuration files with their replacements, GBC, and GBC_LOOKUP_PATH respectively.

Apply fix for mod_proxy_fcgi URL decoding in Apache 2.4

The Apache bug 55329 preventing mod_proxy_fcgi decoding application URLs with spaces or special characters correctly has been fixed for Apache 2.4.11. This bug prevented, for example, applications with URLs like "http://server/gas/wa/r/test/220%2012", from working.

If you are using Apache 2.4.11 or later you must apply the following fix in the Apache configuration to avoid this issue:

```plaintext
SetEnvIf Request_URI . proxy-fcgi-pathinfo=unescape
```

For more details, see Apache 2.4: mod_proxy_fcgi on page 100 or Apache 2.4: Configure mod_proxy_fcgi for remote server on page 101.

If you are using a version of Apache prior to 2.4.11, as the 4GL developer you need to be aware that you can get spaces encoded in URLs and need to handle these in your program code. It is recommended that you decode URLs before using them.

SESSION_COOKIE definition

Starting with GAS version 3.10.16, the SESSION_COOKIE element includes the SameSite attribute.

If you are migrating SESSION_COOKIE elements, you may need to set the SameSite attribute. Typically, to avoid cross-site domain errors, you set attribute SameSite=None and ensure all requests are secured and in HTTPS.

For more information see SESSION_COOKIE on page 425.
GAS 3.00 upgrade guide

These topics describe product changes you must be aware of when upgrading to version 3.00.

Important: This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 3.10 new features on page 306.

Client scope (protocols)

The FGLGWS package includes both Genero Business Development Language and Genero Web Services.

Table 83: FGLGWS 3.00 and client protocols on page 328 shows the protocols and front-ends that are supported with FGLGWS 3.00.

Table 83: 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, GBC, GMA, GMI</td>
</tr>
<tr>
<td>ws</td>
<td>GWS</td>
</tr>
</tbody>
</table>

Table 84: FGLGWS 2.50 and client protocols on page 328 shows the protocols and front-ends that are supported with FGLGWS 2.50.

Table 84: 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 URI Examples on page 48.

IIS configuration no longer automatic

The IIS automatic configuration has been removed from the installation process. Manual configuration is required.

Genero Web Client

Genero Browser Client (GBC) is the default rendering for Genero Web applications, replacing Genero Web Client for HTML5 (GWC-HTML5). New development should use GBC.

Important: Genero Web Client for JavaScript (GWC-JS) applies to the web client 3.00 and lower. From version 3.10, the name GWC-JS is discontinued and replaced by Genero Browser Client (GBC).

Document your existing customization under GWC-HTML5, and determine what needs to be reimplemented with GBC.

Genero Desktop Client ActiveX (GDCAX) is desupported

For new development, use GBC.
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 139. 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](http://www.regex.com) 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 the *Genero Application Server 3.00 Administrator Guide*.

```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](http://www.regex.com) on page 414.

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](http://www.regex.com) on page 276.

Web Components

In GAS 2.50, web components are deployed under `$FGLASDIR/web/components` directory, see [WEB_COMPONENT_DIRECTORY](http://www.regex.com) on page 440.

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` 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` element. Otherwise, you will see the following error message at GAS startup:

```
Application Server startup . ...........................................
httpdispatch
"Configuration ERROR" Code:1871 Message:Element 'TEMPORARY_DIRECTORY': This element is not expected.
  Expected is ( GBC_LOOKUP_PATH ).
  [fail] httpdispatch "Main Configuration"
Cannot build main configuration
```

To find your GAS version, at the command line run:

```
gasadmin -V
```

For more information on the `gasadmin` tool, see `gasadmin tool` on page 348 topic.

**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`
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 [REPORT_VIEWER_DIRECTORY](#) on page 416.

**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. For more information, see the [Genero Browser Client User Guide](#).

**GAS 2.50 upgrade guide**

These topics describe product changes you must be aware of when upgrading to version 2.50.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: [GAS 2.50 new features](#) on page 312.

- **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 installation and application data directories](#) on page 40 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.

**Related concepts**

Log files on page 185

When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**GAS 2.41 upgrade guide**

There are no migration tasks specific for the Genero Application Server 2.41 release.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 2.41 new features on page 314.

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**

These topics describe product changes you must be aware of when upgrading to version 2.40.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 2.40 new features on page 315.

**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

```xml
<html xmlns:gwc="http://www.4js.com/GWC" xmlns="http://www.w3.org/1999/xhtml">
...>

$FGLASDIR/tpl/common/Canvas.xhtml

```xml
<svg:svg xmlns="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"
```
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:

```xml
<sr:Button
    bhv:Media.URI="['{resourceURI("images/close.png", "SetSL")}', 'SmallImage']"
    IsEnabled="{action/isActive || false}"
    IsTabStop="False" />
```

In 2.40:

```xml
<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

These topics describe product changes you must be aware of when upgrading to version 2.30.

Important: This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 2.30 new features on page 317.

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:

```
<RESOURCE Id="res.gwcproxy.param" Source="INTERNAL">--development</RESOURCE>
```

This resource res.gwcproxy.param is used as parameter when launching the GWC proxy

```
<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 the PICTURE element instead. For more information see the PICTURE topic in the Genero Application User Guide v2.30

**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 MAXAVAILABLE. 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 longer 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

These topics describe product changes you must be aware of when upgrading to version 2.22.

Important: This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete
Upgrading

the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 2.22 new features on page 319.

**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**

These topics describe product changes you must be aware of when upgrading to version 2.21.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 2.21 new features on page 319.

**Topics**

- **Configuration**

**Configuration**

- **Picture**

  For web applications, the picture configuration has moved from `APPLICATION/PICTURE` to `APPLICATION/OUTPUT/MAP/PICTURE`.

  See the *Genero Application Server User Guide 2.21* for more details.

---

**GAS 2.20 upgrade guide**

These topics describe product changes you must be aware of when upgrading to version 2.20.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Corresponding new features page: GAS 2.20 new features on page 321.

**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.

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 437 for more details.

DVM ping timeout

Since 2.20.09, the DVM ping timeout, is configurable. See the DVM_PINGTIMEOUT element topic in the Genero Application Server 2.20 User Guide.

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:

http://myWebServer/gas/fglcisapi.dll/ua/r/gwc-demo (with IIS)
http://myWebServer/gas/fglccgi.exe/ua/r/gwc-demo (with Apache)
http://myWebServer/gas/fglccgi/ua/r/gwc-demo

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.

**Important:** This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

**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">
  <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">
  
```

The new CardStep1.xcf:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  
```
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 topic in the Genero Application Server User Guide version 2.20.

**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 409 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 (such as 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 Administrator Guide.

For new development, use Genero Browser Client. See the Genero Browser Client User Guide.

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

These topics describe product changes you must be aware of when upgrading to version 2.10.

Important: This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

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

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

These topics describe product changes you must be aware of when upgrading to version 2.00.

Important: This is an incremental upgrade guide that covers only topics related to the Genero Application Server version specified in the page title. Check prior upgrade guides if you migrate from an earlier version, and complete the migration tasks for all versions between your existing version and the target version in order. Make sure to also read about the new features for this version.

Starting with version 3.10, the fglxmlp migration tool is no longer provided in the package.

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:

```
$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).

```
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:

```
fglcomp -c src1.4gl
fglcomp -c src2.4gl
fgllink -o project.42r src1.42m src2.42m
```

Files used in the example

src1.4gx:

```
01 FUNCTION useRecord (myRecord)
02     DEFINE myRecord $(record)
... 
06 END FUNCTION
```

resource.xrf:

```
01 <?xml version="1.0" ?>
02 
03 <RESOURCE_FILE>
04   <RESOURCE_LIST>
```
The output file src1.4gl:

```groovy
FUNCTION useRecord (myRecord)
    DEFINE myRecord
        RECORD
            nb_columns INTEGER
            nb_lines INTEGER,
            name CHAR (8)
        END RECORD
    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 (such as 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 Administrator Guide*.

For new development, use Genero Browser Client. See the *Genero Browser Client User Guide*.

**Reference**

Genero Application Server reference.

**Tools and Commands**

Information about the dispatchers, proxies, and command line utilities.

**Dispatchers**

GAS dispatchers refer to the connectors in charge of dispatching a GAS request to the appropriate proxy.

- httpdispatch (Standalone)
- fastcgidispatch (Web Server)
- isapidispatch (dedicated to IIS Web server on Windows® platforms)

**Proxies**

Proxies refer to binaries started by a dispatcher to serve a type of request.

- uaproxy (Genero Browser Client (GBC), Genero Desktop Client)
- gwsproxy (GWS)
**Command-line utilities**

The GAS delivers this binary:

- `gasadmin` allows administration of the GAS.

**Dispatcher: httpdispatch**

`httpdispatch` is the standalone dispatcher that starts the Genero Application Server (GAS) in command line. No web server is needed.

**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 (as.xcf) on page 360.

**Syntax**

```
httpdispatch [options]
```

**Options**

**Table 85: 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</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>-f 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, <code>$FGLASDIR/etc/as.xcf</code>, is used.</td>
</tr>
<tr>
<td>-k, --no-keep-alive</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.</td>
</tr>
<tr>
<td>-v, --version</td>
<td>Displays version information.</td>
</tr>
</tbody>
</table>

Example:

```
httpdispatch
   -E res.dvm.wa=$FGLDIR/bin/myrun
```

If in the configuration file "res.dvm.wa" has another value it is now set to myrun. The final value is the one set in the option.

Displays version information.
**Reference**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--pid-file filename</td>
<td>Write the dispatcher process-id to the specified file. The file is created. If the file exists, it is overwritten.</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:

```xml
<INTERFACE_TO_CONNECTOR>
  <TCP_BASE_PORT>6300</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
  ...
</INTERFACE_TO_CONNECTOR>
```

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 400.

**Restarting the standalone GAS**

To restart httpdispatch, use:

```
kill -9
```

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

Pressing 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 pressing CTRL-C too if started in standalone mode. But not on SIGTERM.

**Dispatcher: fastcgidispatch**

`fastcgidispatch` is the dispatcher for Apache Web servers supporting FastCgi protocol.

It can be started by the Web server or in command line. See FastCGI Installation and Web Server Configuration on page 98 for more details.

**Note:** Fastcgidispatch is supported on UNIX™-like platforms only. While fastcgidispatch is known to work on Windows®, it is not recommended and is not supported. For Windows® systems, the Genero Dispatcher: isapidispatch on page 346 dedicated to the Internet Information Services (IIS) Web server is preferred.

**Syntax**

```
fastcgidispatch [options]
```

**Options**

**Table 86: 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>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-f configuration_file</code></td>
<td>Specify which configuration file to use when starting the Genero Application Server dispatcher. 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 name=value</code></td>
<td>Overwrites the resource defined in the configuration file or creates a new one.</td>
</tr>
<tr>
<td><code>--resource-overwrite name=value</code></td>
<td></td>
</tr>
<tr>
<td><code>-s</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>--standalone</code></td>
<td></td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Displays help information.</td>
</tr>
<tr>
<td><code>--help</code></td>
<td></td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Displays version information.</td>
</tr>
<tr>
<td><code>--version</code></td>
<td></td>
</tr>
<tr>
<td><code>--pid-file filename</code></td>
<td>Write the dispatcher process-id to the specified file. The file is created. If the file exists, it is overwritten.</td>
</tr>
</tbody>
</table>

For information on restarting the fastcgidispatch, see [Restarting fastcgidispatch without losing current sessions](#) on page 108.

### Dispatcher: isapidispatch

`isapidispatch` is the dispatcher dedicated to Internet Information Services (IIS) for Web servers on Windows® platforms.

The Genero Application Server (GAS) provides an IIS dispatcher called `isapidispatch.dll` by default, which can be deployed on any Microsoft® server supporting IIS that is configured for GAS.

In the Microsoft® Web server implementation, the GAS is embedded within the ISAPI extension, and is directly loaded by an IIS worker process.

This allows for configuration of GAS ISAPI Extension configuration file and administration of the server with IIS tools. The `isapidispatch` offers the same functionality as [Dispatcher: httpdispatch](#) on page 344 and [Dispatcher: fastcgidispatch](#) on page 345.

### Related concepts

- [ISAPI Extension Installation and Web Server Configuration](#) on page 71
Follow the procedures to configure a Microsoft® Internet Information Services (IIS) Web server for the GAS using ISAPI extensions and the isapidispatch dispatcher.

**Proxy: uaproxy**

The uaproxy manages a Genero Browser Client (GBC) 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 87: 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>
<tr>
<td>--development</td>
<td>Enable the development mode that gives access to the application AUI tree and logs. See Configure the GBC development environment on page 171</td>
</tr>
</tbody>
</table>

**Related concepts**

Proxy on page 39

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

Proxy: gwsproxy on page 347

The gwsproxy manages all services for a single Genero Web application.

**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".

**Related concepts**

Configure applications for Web service on page 116

Create an application configuration file (.xcf) for a Web services application.

Dispatcher on page 38

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

DVM on page 40

The Dynamic Virtual Machine (DVM) or runtime system is the process (fglrn process) 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.

Proxy: uaproxy on page 347
The uproxy manages a Genero Browser Client (GBC) or Genero Desktop Client (GDC) application session.

**gasadmin tool**

The *gasadmin* tool is provided as an administrative command for the Genero Application Server.

Links to examples of some of the more common uses of the tool are grouped under headings: manage sessions, manage applications and clients, manage logs, GAS configuration, and display information:

- **Manage sessions**
  - List all sessions. See [List sessions](#).
  - Ping active sessions. See [Ping active sessions](#).
  - Retrieve information to monitor a session. See [Monitor session](#) on page 355.
  - Stop sessions. See [Stop sessions](#).
  - Close sessions gracefully. See [Close sessions](#).

- **Manage applications and GBC clients**
  - Manage archive files. See [Deploying and managing applications with GAR](#) on page 285.
  - Manage Genero Browser Clients deployed in the GAS. See [GBC command examples](#) on page 356.

- **Manage logs**
  - Reset the log output for a session. See [Reset logs](#) on page 357.

- **GAS configuration**
  - Validate the GAS configuration. See [Validating with the gasadmin tool](#) on page 109.
  - Explode GAS configuration resources. See [Explode configuration file](#).
  - Specify the GAS application directory. See [Specify GAS directory](#).

- **Display information**
  - Display GAS version information. See [Display GAS version information](#) on page 357.

### Syntax 1

```
gasadmin { -V | -h | command [options] }
```

### Syntax 2: with commands (session, config, gar, gbc, reset-log)

```
gasadmin { session | config | gar | gbc | reset-log | options }
```

**gasadmin commands**

The *gasadmin* tool supports five commands:

1. **gasadmin session [options]**
   
The *session* command administers GAS sessions (default). Options are described in Table 89: *gasadmin* session command options on page 349.

2. **gasadmin config [options]**
   
The *config* command handles GAS configuration. Options are described in Table 90: *gasadmin* config command options on page 350.

3. **gasadmin gar [options]**
   
The *gar* command deploys Genero archives (*gar*) files. Options are described in Table 91: *gasadmin* gar command options on page 351.
**Important:** If you start the dispatcher with the option `-E` to override the `$(res.appdata.path)` location, you must also override the resource when deploying applications with the `gasadmin gar` command, in order to deploy to the correct directory.

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 gar -E res.appdata.path=/work/tmp/gas/appdata --deploy-archive myapp.gar
  ```

4. **gasadmin gbc [options]**

   The `gbc` command deploys Genero Browser Client (GBC). Options are described in Table 92: `gasadmin gbc command options` on page 352.

5. **gasadmin reset-log [options] [session-id ...]**

   The `reset-log` command reconfigures the logs for one or more sessions. Options are described in Table 93: `gasadmin reset-log command options` on page 353.

**Options**

**Table 88: gasadmin version and help options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>Display GAS version information.</td>
</tr>
<tr>
<td>--version</td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>Displays help for the <code>gasadmin</code> command.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
</tbody>
</table>

**Table 89: gasadmin session command options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays help for the command.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>-q</td>
<td>Operates in silent mode</td>
</tr>
<tr>
<td>--quiet</td>
<td></td>
</tr>
<tr>
<td>-p directory-name</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--as-directory directory-name</td>
<td></td>
</tr>
<tr>
<td>-f filename</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>--configuration-file filename</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-E resource_name=value</td>
<td>Define or overwrite a resource.</td>
</tr>
<tr>
<td>--resource-overwrite</td>
<td></td>
</tr>
<tr>
<td>resource_name=value</td>
<td></td>
</tr>
<tr>
<td>-d dispatcher-name</td>
<td></td>
</tr>
<tr>
<td>--dispatcher dispatcher_name</td>
<td></td>
</tr>
<tr>
<td>-K</td>
<td></td>
</tr>
<tr>
<td>--kill-all-sessions</td>
<td>Stop (kill) all active sessions by requesting each proxy to stop. The user agent is notified with error messages.</td>
</tr>
<tr>
<td>-k session-id</td>
<td></td>
</tr>
<tr>
<td>--kill-session session_id</td>
<td></td>
</tr>
<tr>
<td>--close-all-sessions</td>
<td>Close all active sessions. No messages are sent to the user agent. Sessions are closed gracefully.</td>
</tr>
<tr>
<td>--close-session session_id</td>
<td></td>
</tr>
<tr>
<td>-X</td>
<td></td>
</tr>
<tr>
<td>--ping-all-sessions</td>
<td>Ping all active sessions. See Ping dispatcher sessions on page 354.</td>
</tr>
<tr>
<td>-x session-id</td>
<td></td>
</tr>
<tr>
<td>--ping-session session_id</td>
<td></td>
</tr>
<tr>
<td>-l</td>
<td></td>
</tr>
<tr>
<td>--list-sessions</td>
<td>List all known sessions. See List dispatcher sessions on page 354.</td>
</tr>
<tr>
<td>-C</td>
<td>Clear remaining Linux®/UNIX™ domain sockets and delete temporary files/directories created, that may not have been removed at the end of a session. See Cleanup session on page 355.</td>
</tr>
<tr>
<td>--session-cleanup</td>
<td>Retrieve monitor information for a session. Information is displayed in XML format on the standard output. See Monitor session on page 355.</td>
</tr>
<tr>
<td>--monitor</td>
<td></td>
</tr>
</tbody>
</table>

**Table 90: gasadmin config command options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays help for the command.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>-q</td>
<td>Operates in silent mode</td>
</tr>
<tr>
<td>--quiet</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>-p directory-name</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--as-directory directory-name</td>
<td></td>
</tr>
<tr>
<td>-f filename</td>
<td>Specify the configuration file to use. If not specified, the</td>
</tr>
<tr>
<td>--configuration-file filename</td>
<td>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>--resource-overwrite resource_name=value</td>
<td></td>
</tr>
<tr>
<td>-c</td>
<td>Checks the configuration file and exits.</td>
</tr>
<tr>
<td>--configuration-check</td>
<td></td>
</tr>
<tr>
<td>-e</td>
<td>Explode the GAS configuration into a hierarchy of</td>
</tr>
<tr>
<td>--configuration-explode</td>
<td>configuration elements and output to file in XML format,</td>
</tr>
<tr>
<td></td>
<td>one for each application.</td>
</tr>
<tr>
<td>-t config_file_name</td>
<td>Explode the given external configuration file in current</td>
</tr>
<tr>
<td>--configuration-explode-external config_file_name</td>
<td>directory. See Explode configuration file into an XML file</td>
</tr>
<tr>
<td></td>
<td>on page 356</td>
</tr>
<tr>
<td>-r</td>
<td>Expand resources and replace with real values.</td>
</tr>
<tr>
<td>--configuration-expand-resources</td>
<td>Used with --configuration-explode or --configuration-</td>
</tr>
<tr>
<td></td>
<td>explode-external. See Explode configuration file into XML</td>
</tr>
<tr>
<td></td>
<td>files on page 356</td>
</tr>
<tr>
<td>-z paths, [...], --compress-resources paths, [...]</td>
<td>Compress the resources located in specified paths. The</td>
</tr>
<tr>
<td></td>
<td>path separator is a comma (.). See Compress resources on</td>
</tr>
<tr>
<td></td>
<td>page 356</td>
</tr>
<tr>
<td>--list</td>
<td>Lists all applications and services (not just the deployed</td>
</tr>
<tr>
<td></td>
<td>ones) found in the GAS.</td>
</tr>
<tr>
<td>--xml-output</td>
<td>Output result in XML format (for --list option only).</td>
</tr>
</tbody>
</table>

Table 91: `gasadmin gar` command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays help for the <code>gar</code> command.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>-p directory-name</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--as-directory directory_name</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>-f configuration_file_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>--configuration-file configuration_file_name</td>
<td></td>
</tr>
<tr>
<td>-E resource_name=value</td>
<td>Define or overwrite a resource with a value.</td>
</tr>
<tr>
<td>--resource-overwrite resource_name=value</td>
<td></td>
</tr>
<tr>
<td>--deploy-archive archive_file</td>
<td>Unpack the given archive content into the deployment directory.</td>
</tr>
<tr>
<td>--undeploy-archive archive_file</td>
<td>Undeploy the given archive.</td>
</tr>
<tr>
<td>--enable-archive archive_file</td>
<td>Expose all services and applications contained in the given 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>Output result of command in XML format. Only compatible with archive options.</td>
</tr>
<tr>
<td>-y</td>
<td>Do not prompt for confirmation.</td>
</tr>
<tr>
<td>--yes</td>
<td></td>
</tr>
</tbody>
</table>

Table 92: gasadmin gbc command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays help for the gbc command.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>-p directory_name</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>--as-directory directory_name</td>
<td></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>--configuration-file configuration_file_name</td>
<td></td>
</tr>
<tr>
<td>-E resource_name=value</td>
<td>Define or overwrite a resource.</td>
</tr>
<tr>
<td>--resource-overwrite resource_name=value</td>
<td></td>
</tr>
<tr>
<td>--deploy gbc_content</td>
<td>Unpack given GBC content into the deployment directory defined by the res.gbc.deployment resource. See Deploy GBC on page 356.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>--undeploy gbc_content</td>
<td>Remove the given GBC content. If the undeployed GBC is the current default, the new default will be the one embedded in the FGLGWS package.</td>
</tr>
<tr>
<td>--default gbc_client</td>
<td>Set the specified GBC as default client. See Set default GBC on page 356</td>
</tr>
<tr>
<td>--list</td>
<td>List all static GBC (those configured in the as.xcf) and deployed clients on the Genero Application Server.</td>
</tr>
<tr>
<td>--reset</td>
<td>Reset to initial delivered GBC in the FGLGWS package.</td>
</tr>
</tbody>
</table>
| --rename old_gbc_name=new_gbc_name | Rename the given GBC.  
**Note:** The GBC client set as default, can not be renamed as it may be in use. |
| --xml-output | Output result of command in XML format. |

Table 93: gasadmin reset-log command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h --help</td>
<td>Displays help for the reset-log command.</td>
</tr>
<tr>
<td>-q --quiet</td>
<td>Operates in silent mode</td>
</tr>
<tr>
<td>-p directory_name --as-directory directory_name</td>
<td>Specify the Genero Application Server directory.</td>
</tr>
<tr>
<td>-f configuration-file --configuration-file configuration_file_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>-E resource_name=value --resource-overwrite resource_name=value</td>
<td>Define or overwrite a resource.</td>
</tr>
<tr>
<td>-d dispatcher-name --dispatcher dispatcher_name</td>
<td>Target the dispatcher - used by session-related options to select the target dispatcher.</td>
</tr>
<tr>
<td>--output-type { CONSOLE</td>
<td>DAILYFILE }</td>
</tr>
<tr>
<td>--output-path output_dir</td>
<td>Log output directory for DAILYFILE, default is the current directory.</td>
</tr>
<tr>
<td>--raw-data-max-length max</td>
<td>Data max length, default is -1.</td>
</tr>
<tr>
<td>--format column-headings</td>
<td>Log format, columns to output, the default is: time event-type event-params.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--categories category-list</td>
<td>Log categories to enable, the default is: GAS ACCESS PROCESS ERROR</td>
</tr>
</tbody>
</table>

**Session command examples**

These examples show how you use the `gasadmin session` command to work with sessions.

**Ping dispatcher sessions**

This example shows how to use `-X -f -d` options to ping all sessions of a specified GAS configuration file. This effects sessions running on the specified dispatcher.

```bash
gasadmin session -X -f as1.xcf -d httpdispatch
```

The GAS outputs the reply from all current sessions. The output displays the session id, which is a long string, for example, "96c9ce0ded72135ddf43ad421a2d87b9" with the name of the service or application in parenthesis. If the ping is successful, the reply from the dispatcher is OK.

Checking all sessions: (httpdispatch)
Ping session 96c9ce0ded72135ddf43ad421a2d87b9 (demo/RestBook): OK
Ping session dfd29c347ecf2d572aef95a13c6d4a04 (_default/gwc-demo): OK

**List dispatcher sessions**

This example shows you how to use the `-l` option to list all sessions. The `-d` option specifies the dispatcher.

```bash
gasadmin session -l -d httpdispatch
```

The output displays a list of the sessions. It includes the following details:

- **session identifier.** This represents the GAS session for the application or Web service. In the example, "96c9ce0ded72135ddf43ad421a2d87b9" is the session for the demo/RestBook Web service.
- **Port:** this is the port number the dispatcher is listening on
- **Type:** identifies the type of session (service or application)
- **Pid:** this is the process id number
- **GSID:** this is the Genero session id used by Web applications. (for example, 1a5569ed45193a6abd7a2e8e67199300)

**Note:** The GSID is used by the browser to keep track of the session of a Web application. The value is stored in a cookie. GSID is not used by Web services.

**Stop dispatcher sessions**

If you need to stop a session, you use the `-k` option of the `gasadmin session` command for this purpose.
This action effects the specified session id, "d98290172c8f7c0d861db329f1ce6597" in the example. With the option \(-f\) you can specify the configuration file. The \(-d\) option you specify the dispatcher where the session is running.

```
gasadmin session -k d98290172c8f7c0d861db329f1ce6597 -f as1.xcf -d isapidispatch
```

**Tip:** You can use the \(-K\) option to terminate (kill) all active sessions.

**Close dispatcher sessions**

If you need to stop a session gracefully and therefore not send messages to the user agent, use the close session option instead of the \(-k\) (kill) option. The \(--\text{close-session}\) option runs the `gasadmin` command on the `TCP_ADMIN_PORT` on page 429 port for this purpose.

In the example the close session action effects the specified session id, "d98290172c8f7c0d861db329f1ce6597". With the option \(-f\) you can specify the configuration file. The \(-d\) option specifies the dispatcher where the session is running.

```
gasadmin session --close-session d98290172c8f7c0d861db329f1ce6597 -f as1.xcf -d isapidispatch
```

**Tip:** You can use the \(--\text{close-all-sessions}\) option to close all active sessions on the dispatcher.

**Cleanup session**

This example shows you how to perform a cleanup on the GAS to remove temporary files or directories that may have been used during a session.

```
gasadmin session --cleanup-session -d <dispatcher>
```

The \(-d\) or \(--\text{dispatcher}\) option is required to specify the dispatcher. The dispatcher is specified as either `httpdispatch`, `isapidispatch`, or `fastcgidispatch`.

**Note:** This cleanup is performed automatically at dispatcher start up.

If your GAS version is prior to 3.10, you need to use the following command:

```
gasadmin --session-cleanup -d <dispatcher>
```

**Monitor session**

This example shows how to use the \(--\text{monitor}\) 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 session --dispatcher <dispatcher> --monitor <session-id>
```

The \(-d\) or \(--\text{dispatcher}\) option is required to specify the dispatcher.

If your GAS version is prior to 3.10, you need to use the following command:

```
gasadmin --dispatcher <dispatcher> --monitor <session-id>
```

**Config command examples**

These examples show how you use the `gasadmin config` command to work with configuration files.
### Explode configuration file into an XML file

This example shows how you use the `-t` option of the `gasadmin config` command to explode the specified application configuration file and expand its resources and its parent's resources into an XML file.

```
  gasadmin config -t demo/Card
```

### Explode configuration file into XML files

This example shows how you use the `-r -t` options to explode the specified application configuration file. This causes its resources and its parent resources to be replaced with real values. The result is output in separate XML files.

```
  gasadmin config -r -t demo/Card
```

### Compress resources

This example shows how you compress the resources located in the specified paths.

```
  gasadmin config -z $FGLASDIR/app,$FGLASDIR/services,$FGLASDIR/web,$FGLASDIR/tpl
```

### GBC command examples

These examples show how you use the `gasadmin gbc` command to manage GBC clients.

#### Deploy GBC

This example shows how you can use the `gasadmin deploy` command to deploy a GBC client on the GAS.

```
  gasadmin gbc --deploy c:\fjs\gbc-projects\gbc-1.00.53\archive\custA.zip
```

#### Set default GBC

These examples show how you can use `gasadmin deploy` command to list the deployed GBC clients and set a default client on the GAS.

```
  gasadmin gbc --list
```

```
  gasadmin gbc --default custB
```
Figure 71: List deployed GBC and set default

**Reset-log command examples**

This example shows how you can use the `gasadmin reset-log` command to reconfigure the log output.

**Reset logs**

```
C:\fjs\gst\3.20.04\gst\bin>gasadmin gbc --list
Command succeeded.
Deployed GBC:
  - custA (default)
  - custB
  - sample
3 GBC(s) found

Static GBC available on C:\fjs\fgl\3.20.05\web_utilities\gbc:
  - gbc (default)
1 GBC(s) found

C:\fjs\gst\3.20.04\gst\bin>gasadmin gbc --default custB
Command succeeded.

CustB set as default GBC
```

**Display GAS version information**

```
gasadmin -V
```

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.
Predefined resources

Predefined resources for the Genero Application Server can be general or front-end specific.

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 cannot 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:

```<WEB_COMPONENT_DIRECTORY>${application.path}/webcomponents</WEB_COMPONENT_DIRECTORY>```
Related reference

Common predefined resources on page 359
These Genero Application Server predefined resources are available for any front-end.

Common predefined resources

These Genero Application Server predefined resources are available for any front-end.

These resources include:

Table 94: 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>&lt;WEB_APPLICATION_PICTURE_COMPONENT&lt;br&gt; Id=&quot;cpn.gwc.picture&quot;&lt;br&gt; ...&lt;br&gt; &lt;PATH Id=&quot;Image&quot; Type=&quot;APPSERVER&quot;&gt;&lt;br&gt; $(res.path.pic);$(application.path)&lt;br&gt; &lt;/PATH&gt;&lt;br&gt; ...&lt;br&gt; &lt;/WEB_APPLICATION_PICTURE_COMPONENT&gt;</td>
</tr>
<tr>
<td>connector.uri</td>
<td>This is replaced by the URL path to the connector. For example, if the URL is:</td>
</tr>
<tr>
<td></td>
<td><a href="http://localhost/gas/ua/r/demo">http://localhost/gas/ua/r/demo</a>&lt;br&gt; The value of connector.uri is /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>

Related concepts

URI Examples on page 48
Several URI examples with ways to help you launch applications.

GAS Configuration Reference

These topics provide reference details for Genero Application Server configuration.
**GAS configuration file (as.xcf)**

The Genero Application Server is configured through a configuration file. The default configuration file is `as.xcf`, located in the `$FGLASDIR/etc` directory.

The configuration file serves as a global configuration for the GAS and the applications that run on it. It is an XML file that contains a list of configuration elements.

- The root element is the [CONFIGURATION](#) element. This element has a single child element, the [APPLICATION_SERVER](#) element, which defines the GAS.
- The [RESOURCE_LIST](#) element provides a platform-specific resource configuration for Windows® and UNIX® systems.
- The [COMPONENT_LIST](#) element provides support for component configuration in the GAS. Where there is a `using` attribute in, for example the [APPLICATION](#) element, a component element can be referenced to provide default configurations for applications and services.
- The [APPLICATION_LIST](#) element provides support for abstract application definition. It contains a list of default groups and abstract application configurations that define details that are common to many applications. Where there is a `Parent` attribute in, for example, an [APPLICATION](#) element, it inherits the configuration of the parent application.
- The [SERVICE_LIST](#) provides support for abstract service application definitions.

See [GAS configuration file hierarchy](#) for a full list of valid elements and their place in the file hierarchy.

To create a new Genero Application Server configuration file, create a copy of the default and make modifications in the copy. Use the dispatcher `-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.

**Important:** When you add an application to the GAS configuration file, you must restart the GAS for the application to be recognized.

After the restart, depending on the dispatcher, you may or may not have the applications (for example, proxies) stopped; some of the dispatchers can recover the application and continue where the application left off.

**Related concepts**

- [Dispatcher: httpdispatch](#) on page 344
  `httpdispatch` is the standalone dispatcher that starts the Genero Application Server (GAS) in command line. No web server is needed.

- [Dispatcher: fastcgidispatch](#) on page 345
  `fastcgidispatch` is the dispatcher for Apache Web servers supporting FastCgi protocol.

- [Dispatcher: isapidispatch](#) on page 346
  `isapidispatch` is the dispatcher dedicated to Internet Information Services (IIS) for Web servers on Windows® platforms.

- [Dispatcher](#) on page 38
  Understand what the dispatcher does and identify which dispatcher to use with a specific Web server.

- [Configuration for Web and service applications](#) on page 360
  Configure Genero Web applications and services using a subset of the elements provided for the Genero Application Server configuration.

**Configuration for Web and service applications**

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 has its own configuration file, written in XML.
Elements in an application's configuration

The elements allowed, and how they are used, depend on whether the application is a Web application or a Web service (GWS). For a full list of valid elements and their place in the file hierarchy, see the appropriate topic:

- GBC configuration file hierarchy on page 364
- GWS configuration file hierarchy on page 365

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.

Related concepts
Configuring applications on GAS on page 111
Understand the options available for configuring and deploying applications on the GAS.

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

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
    - UNIX
      - RESOURCE
  - COMPONENT_LIST
    - WEB_APPLICATION_EXECUTION_COMPONENT
      - ENVIRONMENT_VARIABLE
      - PATH
      - LIST bullet 5
      - WEB_COMPONENT_DIRECTORY on page 440
      - LIST bullet 5
      - MODULE
      - LIST bullet 5
      - PARAMETERS
      - LIST bullet 6
      - PARAMETER on page 408
    - DELEGATE
      - LIST bullet 6
      - ACCESS_CONTROL
      - LIST bullet 5
• SERVICE_APPLICATION_EXECUTION_COMPONENT
  • ENVIRONMENT_VARIABLE
  • PATH
  • DVM
  • MODULE
  • PARAMETERS
  • ACCESS_CONTROL
  • DELEGATE
  • POOL
  • START
  • MIN_AVAILABLE
  • MAX_AVAILABLE
  • MAX_REQUESTS_PER_DVM
• WEB_APPLICATION_TIMEOUT_COMPONENT
  • USER_AGENT
  • REQUEST_RESULT (for an application) on page 417
• AUTO_LOGOUT_COMPONENT on page 377
  • TIMELIMIT (for auto logout) on page 433
• UA_OUTPUT_COMPONENT on page 435
  • PROXY (for an application) on page 413
  • PUBLIC_IMAGEPATH on page 414
  • TIMEOUT (for an application) on page 431
  • GBC on page 394
  • GDC_SHORTCUT on page 396
• SERVICE_APPLICATION_TIMEOUT_COMPONENT
  • DVM_AVAILABLE
  • KEEP_ALIVE
  • REQUEST_RESULT (for a service) on page 418

• INTERFACE_TO_CONNECTOR
  • ROOT_URL_PREFIX on page 421
  • LISTEN on page 402
  • ADDRESS (LISTEN) on page 369
  • TCP_ADMIN_PORT on page 429
  • TCP_BASE_PORT
  • TCP_PORT_OFFSET
  • DOCUMENT_ROOT
  • GBC_LOOKUP_PATH on page 395
  • TEMPORARY_DIRECTORY
  • SESSION_DIRECTORY
  • REPORT_VIEWER_DIRECTORY on page 416
  • REPORT_REMOTE_URL_PREFIX on page 416
  • SOCKET_FAMILY
• SOCKET_PATH
  • HTTP on page 399
    • SESSION_COOKIE on page 425
    • CACHE_CONTROL_MAX_AGE on page 378
    • APPLICATION (for HTTP) on page 372
    • SERVICE (for HTTP) on page 423
      • INTERFACE_TO_DVM
      • ADDRESS
      • LOG
        • OUTPUT
        • FORMAT
        • CATEGORIES_FILTER
        • RAW_DATA
      • MONITOR
        • ALLOW_FROM
      • APPLICATION
        • APPLICATION_LIST
          • GROUP
          • APPLICATION
            • DESCRIPTION
              • SHORT
                List bullet 5
              • LONG
                List bullet 5
            • RESOURCE
            • EXECUTION
              • ENVIRONMENT_VARIABLE
                List bullet 5
              • PATH
                List bullet 5
              • MODULE
                List bullet 5
              • PARAMETERS
                List bullet 5
                • PARAMETER
                  • ACCESS_CONTROL
                    • ALLOW_FROM
                      List bullet 6
                    • DELEGATE
                      List bullet 5
                    • WEB_COMPONENT_DIRECTORY on page 440
                  • DELEGATE
                    • AUTO_LOGOUT on page 375
                • PROXY
                  • PUBLIC_IMAGEPATH
                  • TIMEOUT
                    • USER_AGENT
                      List bullet 6
                    • REQUEST_RESULT
                      List bullet 6
                    • DVM_AVAILABLE
                      List bullet 6
                • GBC on page 394
                • GDC_SHORTCUT on page 396
Unordered List bullet 5

- ACCESS_CONTROL
- DELEGATE
- POOL
  - MIN_AVAILABLE
  - MAX_AVAILABLE
  - MAX_REQUESTS_PER_DVM

- TIMEOUT (for a service) on page 432
  - DVM_AVAILABLE
  - KEEP_ALIVE
  - REQUEST_RESULT (for a service) on page 418

**GBC configuration file hierarchy**

A listing of the elements valid for a Genero Browser Client (GBC) 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
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
    - PARAMETERS
      - PARAMETER on page 408
  - ACCESS_CONTROL
    - ALLOW_FROM
    - DELEGATE
    - POOL
• **START**
• **MIN_AVAILABLE**
• **MAX_AVAILABLE**
• **MAX_REQUESTS_PER_DVM**
• **TIMEOUT** (for a service) on page 432
• **DVM_AVAILABLE**
• **KEEP_ALIVE**
• **REQUEST_RESULT** (for a service) on page 418

**Syntax diagrams**

Syntax diagrams indicate the rules and requirements for the XML-based GAS configuration file elements. These elements may be found in a GAS configuration file (default \$FGLASDIR/etc/as.xcf) or in an external application configuration file.

**Example: APPLICATION Element Syntax Usage.**

This syntax diagram describes the APPLICATION configuration element, its attributes, and its child elements:

```xml
<APPLICATION [Id=app-name]
    Parent=parent-app-name
    xmlns:xsi=uri
    xsi:noNamespaceSchemaLocation=schema
    [Abstract=["TRUE"]|"FALSE"]
    [mode=mode-name]>
    [<DESCRIPTION>...</DESCRIPTION>]  
    [<RESOURCE>...</RESOURCE>] [....]
    [<EXECUTION>...</EXECUTION>]
    [<AUTO_LOGOUT>...</AUTO_LOGOUT>]
    [<UA_OUTPUT>...</UA_OUTPUT>]
    [<END_URL>...</END_URL>]<APPLICATION>
```

• Configuration elements are written in uppercase within an opening tag (<ELEMENT_NAME>) and a closing tag (<ELEMENT_NAME>)

• If a configuration element has attributes, each attribute consists of an attribute name, followed by an equals sign (=), followed by the attribute value.

• An attribute name is a string with no spaces. Attribute names are presented in code format.

• Attribute values are in double quotes. Attribute values presented in code format are exact, and should be used as-is. For example, the attribute `Abstract` takes a value of either "TRUE" or "FALSE". Attribute values presented as variables expect a string to replace the variable name. For example, the attribute `mode` expects you to replace "mode_name" with a valid mode name.

• Attributes appear separated by spaces within the opening tag of the configuration element.

• Wildcard characters in syntax definitions are usually marked with an underscore. They are used to indicate elements that can either repeat, be mandatory, or be optional.

**Table 95: Wildcard characters**

<table>
<thead>
<tr>
<th>Wildcards</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ e ]</td>
<td>Square braces indicate an optional element in the syntax.</td>
</tr>
<tr>
<td>...</td>
<td>A sign with three dots (ellipsis) indicates that a number of items may appear in a list. If used between an element’s opening and closing tags, it indicates one or more child elements may appear.</td>
</tr>
</tbody>
</table>
### Wildcards

<table>
<thead>
<tr>
<th>Wildcards</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`[ e1</td>
<td>e2 ... ]`</td>
</tr>
<tr>
<td>`{ e1</td>
<td>e2 ... }`</td>
</tr>
<tr>
<td><code>[...]</code></td>
<td>A sign made of square braces with three dots indicate that the previous element can appear more than once.</td>
</tr>
<tr>
<td><code>, ...</code></td>
<td>A sign made of square braces with a comma followed by three dots indicate that the previous element can appear more than once, and must be separated by a comma.</td>
</tr>
</tbody>
</table>

### Configuration file elements

An alphabetical listing of the elements used across the Genero Application Server configuration files.

#### ACCESS_CONTROL

The `ACCESS_CONTROL` element specifies access from a list of IP allowed to access applications or services.

**Syntax**

```xml
<ACCESS_CONTROL>
   [ALLOW_FROM]...
</ACCESS_CONTROL>
```

**Child elements**

- Zero or more `ALLOW_FROM` elements.

**Usage**

You use this element to specify what users have access to applications or services. By default, an application or a service is not accessible by anyone. It needs to be explicitly configured with `ALLOW_FROM` elements.

```xml
<ACCESS_CONTROL>
   <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
   <ALLOW_FROM>10.0.0.0</ALLOW_FROM>
   <ALLOW_FROM>192.168.0.0/16</ALLOW_FROM>
   <ALLOW_FROM>fdbd:2768:c176:1::323a</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." or "10.", or the IPv6 address "fdbd:2768:c176:1::323a".

**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.

**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).

To allow access from the localhost, in the GAS (default `$FGLASDIR/etc/as.xcf`) configuration file you need to change the application element for `gwc-demo` from:

```xml
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
```

To:

```xml
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
```

```xml
</ACCESS_CONTROL>
```
Example configuring access control for Genero Identity Provider

Access to the Genero Identity Provider (GIP) on page 204 applications is specified by the resource res.access.control. This is set with the value NOBODY by default.

To allow users to have access, you need to change the GAS (default $FGLASDIR/etc/as.xcf) configuration file for res.access.control to ALL:

```
<RESOURCE Id="res.access.control" Source="INTERNAL">ALL</RESOURCE>
```

Parent elements

This element is a child of one of the following elements:

- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 421
- WEB_APPLICATION_EXECUTION_COMPONENT on page 437
- EXECUTION (for an application) on page 390
- EXECUTION (for a service) on page 391

ADDRESS

The ADDRESS element specifies the machine where the Dynamic Virtual Machine (DVM) runs.

Syntax

```
<ADDRESS>address</ADDRESS>
```

1. *address* is the name or IP address of the machine where the Dynamic Virtual Machine (DVM) runs. This address is used by the DVM to set (build) the FGLSERVER environment variable.

Usage

You use this element to specify the address of the machine where the DVM runs. Examples are shown of how to specify this using the server name and IP address.

Usage example by server name

```
<INTERFACE_TO_DVM>
<ADDRESS>app_server</ADDRESS>  
</INTERFACE_TO_DVM>

**Usage example by IP address**

<INTERFACE_TO_DVM>  
  <ADDRESS>192.127.45.17</ADDRESS>  
</INTERFACE_TO_DVM>

**Warning**: Avoid using localhost or 127.0.0.1. The license server requires the real address of the machine to check the licenses. If your machine is not well configured, such settings return a bad address to the license server, which will then refuse to start a new DVM.

**Parent elements**

This element is a child of the `INTERFACE_TO_DVM` on page 401 element.

**ADDRESS (LISTEN)**

The `ADDRESS` element specifies the IP address of the machine where the dispatcher listens.

**Syntax**

```
<ADDRESS>ip-address</ADDRESS>
```

1. `ip-address` is any valid IP address of the machine.

   This is an optional element.

**Child elements**

There are no child elements.

**Usage**

This element allows you to configure one machine to communicate with the dispatcher. For security reasons it is recommended to configure the dispatcher to only accept connections coming from the web server (Apache for instance) on one IP address.

For example, you can restrict access to the loopback address (127.0.0.1) of the machine where the dispatcher is running, or you can set it to any valid IP address of the machine. Examples are shown of how to specify this using the loopback and IP address.

**Usage example with loopback**

```
<Listen>  
  <ADDRESS>127.0.0.1</ADDRESS>  
</Listen>
```

**Usage example with actual IP address**

```
<Listen>  
  <ADDRESS>192.127.45.17</ADDRESS>  
</Listen>
```

**Parent elements**

This element is a child of the `LISTEN` on page 402 element.
ALLOW_FROM
The ALLOW_FROM element specifies rules for access to GAS applications or services from specific hosts.

Syntax

```
<ALLOW_FROM>"NOBODY" | "ALL" | ip-address</ALLOW_FROM>
```

1. 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). Access can also either be globally denied or allowed by the following keywords:
   - NOBODY
   - ALL

Usage

You use this element to specify the specific hosts who have access to your GAS applications or services. An example is shown using the default resource res.access.control. For more examples see Access Control.

Example default usage

```
<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 405 references this resource.

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 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 the following:

- ACCESS_CONTROL on page 367
- MONITOR on page 405

APPLICATION (for an application)

This APPLICATION element defines an application within the Genero Application Server configuration file or in an external application configuration file.

Syntax

```
<APPLICATION [Id=app-name]
    Parent=parent-app-name
    xmlns:xsi=uri
    xsi:noNamespaceSchemaLocation=schema
    [Abstract="TRUE" | "FALSE"]
    [mode=mode-name]>
    [...]
    [</DESCRIPTION>]
    [</RESOURCE>]
```
1. The `app-name` identifies the application. The `Id` attribute is required for applications defined within the Genero Application Server configuration file. For external configuration files, the `Id` attribute is ignored. The `Id` specified is compared to the application name in the request.

2. The `parent-app-name` identifies the parent application, or the application from which this application will inherit its default configuration settings.

3. The `uri` defines the URL of the XML schema. The default value is "http://www.w3.org/2001/XMLSchema-instance".

4. The `schema` defines the XML schema instance. When defining external application files, the valid value for a Web applications is "http://www.4js.com/ns/gas/version/cfextwa.xsd"

5. The `Abstract` attribute defines whether this application configuration element is for an abstract application or a real application. Valid values are "TRUE" and "FALSE". An abstract application acts as a template and can not instantiate Virtual Machines.

6. The `mode-name` when set to "sticky", defines a Web service as a sticky Web service. See Configure sticky Web services on page 269.

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 (for an application) on page 390` elements.
4. Zero or one `AUTO_LOGOUT on page 375` elements.
5. Zero or one `UA_OUTPUT on page 434` elements. This element is used for all UI applications.
6. Zero or one `END_URL on page 388` element.

Usage

You use this element to configure Web applications you wish to make accessible through the GAS.

Example external Web application configuration file

```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/3.20/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>
  <AUTO_LOGOUT Using="cpn.wa.autologout"/>
  <UA_OUTPUT>
    <PROXY>$(res.uaprox.cmd)</PROXY>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <GBC>gwc-dev</GBC>
    <TIMEOUT> Using="cpn.wa.timeout"</TIMEOUT>
  </UA_OUTPUT>
  <END_URL>http://www.4js.com</END_URL>
</APPLICATION>
```
For more information, see Configuring applications on GAS on page 111.

**Parent elements**

When used in an application configuration file (.xcf), it is the top-most element.

When used in the GAS configuration file, this element is a child of the APPLICATION_LIST on page 374 element.

**Related concepts**

APPLICATION (for a service) on page 373
This APPLICATION element defines a service within the Genero Application Server configuration file or in an external application configuration file.

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.

**Syntax**

```
<HTTP>
  ...
  <APPLICATION>
    <!--HEADER>...</HEADER> -->
  </APPLICATION>
  ...
</HTTP>
```

**Child elements**

The APPLICATION element may contain zero to many HEADER on page 398 elements.

**Usage**

You use this element to specify a list of HEADER elements defining the communication carried on the HTTP protocol between Web application 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.

**Example usage**

```
<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>
  </HTTP>
</INTERFACE_TO_CONNECTOR>
```

**Parent elements**

This element is a child of the HTTP on page 399 element.
APPLICATION (for a service)

This APPLICATION element defines a service within the Genero Application Server configuration file or in an external application configuration file.

Note: Since Genero 2.0, Web services are named applications as they host several Web services in one DVM.

Syntax

```
<APPLICATION 
    Id="app-name"
    Parent="parent-app-name"
    xmlns:xsi="uri"
    xsi:noNamespaceSchemaLocation="schema"
    [Abstract="[TRUE][FALSE]"
    [mode="mode-name"]>

    [<DESCRIPTION>...</DESCRIPTION>]
    [<RESOURCE>...</RESOURCE>]
    [<PROXY>...</PROXY>]
    [<EXECUTION>...</EXECUTION>]
    [<TIMEOUT>...</TIMEOUT>]

</APPLICATION>
```

1. The `app-name` identifies the application. The `Id` attribute is required for applications defined within the Genero Application Server configuration file. For external configuration files, the `Id` attribute is ignored. The `Id` specified is compared to the application name in the request.
2. The `parent-app-name` identifies the parent application, or the application from which this application will inherit its default configuration settings.
3. The `uri` defines the URL of the XML schema. The default value is "http://www.w3.org/2001/XMLSchema-instance".
4. The `schema` defines the XML schema instance. When defining external application files, the valid value for a Web service is "http://www.4js.com/ns/gas/version/cfextws.xsd"
5. The `Abstract` attribute defines whether this application configuration element is for an abstract application or a real application. Valid values are "TRUE" and "FALSE". An abstract application acts as a template and can not instantiate Virtual Machines.
6. The `mode-name` when set to "sticky", defines a Web service as a sticky Web service. See Configure sticky Web services on page 269.

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 PROXY (for a service) element
4. Zero or one EXECUTION element.
5. Zero or more TIMEOUT elements.

Usage

You use this element to configure Web services you wish to make accessible through the GAS.

Example usage

```
<APPLICATION Id="webapp" Parent="ws.abswebapp">

    <EXECUTION>
        <PATH>$(res.path.fgldir.demo)</PATH>
        <MODULE>webapp.42r</MODULE>
    </EXECUTION>
    <TIMEOUT> </TIMEOUT>
</APPLICATION>
```
Parent elements

When used in an application configuration file (.xcf), it is the top-most element. When used in the GAS configuration file, this element is a child of the SERVICE_LIST on page 424 element.

Related concepts

APPLICATION (for an application) on page 370
This APPLICATION element defines an application within the Genero Application Server configuration file or in an external application configuration file.

APPLICATION_LIST
The APPLICATION_LIST element provides a list of default groups and abstract application configurations.

Syntax

```xml
<APPLICATION_LIST>
  [...]
  [...]
  [...]
</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.

Usage

This element is defined within the Genero Application Server (GAS) configuration file (as.xcf). For each application to be served by the GAS, you must provide the details for that application in either the GAS configuration file or in an external application configuration file (xcf).

Example usage in as.xcf

```xml
<APPLICATION_LIST>
  <GROUP Id="appgroup">/home/appgroup</GROUP>
  <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 375 element.

Related concepts

Configuring applications on GAS on page 111
Understand the options available for configuring and deploying applications on the GAS.

**APPLICATION_SERVER**
This element acts as a parent container for all Genero Application Server configuration elements.

**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> [...]]
  [<APPLICATION_LIST>...</APPLICATION_LIST>]
  [<SERVICE_LIST>...</SERVICE_LIST>]
</APPLICATION_SERVER>
```

1. Each element is discussed in its own section of this manual.

**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.
- 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 one `APPLICATION_LIST` element, containing a list of applications.
- Zero or one `SERVICE_LIST` element, containing a list of Web services.

**Usage**

This element forms a parent container for all Genero Application Server configuration elements found in the `as.xcf`.

**Example usage**

```
<CONFIGURATION
  <APPLICATION_SERVER>
    ...
  </APPLICATION_SERVER>
</CONFIGURATION>
```

**Parent elements**

This element is a child of the `CONFIGURATION` on page 382 element.

**AUTO_LOGOUT**

The `AUTO_LOGOUT` element defines the auto-logout mechanism to be used for an application.

**Syntax**

```
<AUTO_LOGOUT Using="component-id">```

1. The component-id references an AUTO_LOGOUT_COMPONENT on page 377 to inherit its parameters.

Child elements

The AUTO_LOGOUT element may contain the following child elements:

1. Zero or one TIMEOUT (for auto logout) on page 433 element.
2. Zero or one COMMAND (for auto logout) on page 381 element.
3. Zero or one PROMPT (for auto logout) on page 412 element.

Usage

You use this element to define the auto logout mechanism for an application. You can reference the default auto-logout component or set the element directly in your application configuration file. Examples of both methods are shown.

When the AUTO_LOGOUT is set, the DVM detects when an application has no user activity. It then waits the number of seconds specified by the TIMEOUT element before triggering an auto logout event and sending the application a log out message.

- For Genero Browser Client (GBC) applications, you get a log out page.
- For Genero Desktop Client (GDC) applications, you 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.

Usage example - reference 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 example 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>
```

The default AUTO_LOGOUT_COMPONENT, cpn.wa.autologout, is shown in this example. The default timeout duration set to 0 seconds means the auto logout is ignored and applications keep running.

```xml
<COMPONENT_LIST>
  ...
  <AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
    <TIMEOUT>0</TIMEOUT>
  </AUTO_LOGOUT_COMPONENT>
  ...
</COMPONENT_LIST>
```
### Usage example - use auto logout directly

If you want to set auto logout directly in your application configuration file (`xcf`), you can add an `AUTO_LOGOUT` element. In this example, the timeout duration is set to 30 seconds, which means the DVM waits 30 seconds from when it detects an application has no user activity before the application gets a log out message.

```xml
<APPLICATION Parent="defaultwa">
  <EXECUTION>
    <PATH>my_app_dir</PATH>
    <MODULE>my_module</MODULE>
  </EXECUTION>
  <AUTO_LOGOUT>
    <TIMEOUT>30</TIMEOUT>
  </AUTO_LOGOUT>
  ...
</APPLICATION>
```

### Auto logout and child applications

When an application has launched child applications, the auto logout is not triggered as long as the parent or any of its child applications have user activity.

**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.

### Parent elements

The `AUTO_LOGOUT` element is a child of the `APPLICATION` element.

### AUTO_LOGOUT_COMPONENT

The `AUTO_LOGOUT_COMPONENT` element creates a component, which defines a mechanism for triggering and handling auto-logout events.

#### Syntax

```
<AUTO_LOGOUT_COMPONENT Id="component-id">
  [...]<TIMEOUT>...</TIMEOUT>]
  [...]<COMMAND>...</COMMAND>]
  [...]<PROMPT>...</PROMPT>]
</AUTO_LOGOUT_COMPONENT>
```

1. The `component-id` specifies the unique identifier for the auto-logout definition. It is this unique identifier that is referenced by an application, providing it with a base set of auto-logout values.

### Child elements

The `AUTO_LOGOUT_COMPONENT` element may contain the following child elements:

1. Zero or one `TIMEOUT` element.
2. Zero or one `COMMAND` element.
3. Zero or one `PROMPT` element.
Usage

You use this element to define a component for handling auto-logout events. There is a default auto logout component in the GAS configuration file, cpn.wa.autologout, which is shown in the first example. If you need to configure other conditions for auto logout, you can set the COMMAND element to run a script as shown in second example.

Usage example default AUTO_LOGOUT_COMPONENT

```
<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 375 element.

Usage example - auto logout with COMMAND

```
<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.

Parent elements

This element is a child of the following element: COMPONENT_LIST on page 381.

CACHE_CONTROL_MAX_AGE

The CACHE_CONTROL_MAX_AGE element specifies a setting for a HTTP Cache-Control header when a file is cacheable.

Syntax

```
<CACHE_CONTROL_MAX_AGE> seconds </CACHE_CONTROL_MAX_AGE>
```

1. seconds specifies in seconds the time allowed for holding files in the cache.

Child elements

There are no child elements.

Usage

You use the CACHE_CONTROL_MAX_AGE element to define the duration files sent by the Genero Application Server are held in front-end cache. The value of 300 seconds is the default, as configured in the CACHE_CONTROL_MAX_AGE element of the GAS configuration file. When the GAS sends a file to the front-end, if the file is cacheable, the header Cache-Control is added in the message response, for example:

```
Cache-Control: max-age=300
```
The standalone dispatcher (httpdispatch), as a development tool, always ignores this configuration entry and will force the maximum duration for caching to one second:

```
Cache-Control: max-age=1
```

**Usage example**

```xml
<INTERFACE_TO_CONNECTOR>
  ...
  <HTTP>
    <SESSION_COOKIE Secure="FALSE" />
    <CACHE_CONTROL_MAX_AGE>300</CACHE_CONTROL_MAX_AGE>
    <!-- Applications custom headers -->
    ...
  </HTTP>
</INTERFACE_TO_CONNECTOR>
```

**Parent elements**

This element is a child of the [HTTP](on page 399) element.

**CATEGORIES_FILTER**

The CATEGORIES_FILTER element specifies which categories of log messages will be written to the log.

**Syntax**

```
<CATEGORIES_FILTER> name [ ... ]</CATEGORIES_FILTER>
```

1. The *name* is the filter name of a category of messages to write to the log. You can include multiple categories of messages by listing multiple filter names, separated by spaces. [Table 96: Category filters](on page 379) lists the valid category filter names.

**Table 96: Category filters**

<table>
<thead>
<tr>
<th>Filter Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Log internal information for debugging.</td>
</tr>
<tr>
<td></td>
<td>Important: Do not set unless requested by your support center.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Log warning messages.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Log error information.</td>
</tr>
<tr>
<td>MUTEX</td>
<td>Log multi-threading information.</td>
</tr>
<tr>
<td>SESSION</td>
<td>Log session management information.</td>
</tr>
<tr>
<td>GAS</td>
<td>Log information about the Genero Application Server version, build and</td>
</tr>
<tr>
<td></td>
<td>package.</td>
</tr>
<tr>
<td>LOG</td>
<td>Display log parameters.</td>
</tr>
<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</td>
</tr>
<tr>
<td></td>
<td>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>Filter Names</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</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 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>

**Child elements**

There are no child elements.

**Usage**

You use this element to specify which categories of log messages to write to the log. It is an optional element of the LOG element. If it is omitted, the default messages for LOG and ERROR categories are logged.

With this example, all error and warning messages are written to the logs.

```xml
<CATEGORIES_FILTER>ERROR WARNING</CATEGORIES_FILTER>
```

In production, it is recommended that you use these categories: (GAS ACCESS PROCESS DEPRECATED ERROR WARNING).

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.

Your support center will tell you when you should set ALL for your categories filter.

**Identify the category of a message**

It is recommended to add the category entry to the FORMAT on page 392 element, as it allows you to easily identify the category of each message written to the log file. Each line in the log will show the category of the message. You can use this information to:

- Scan the logs for messages of a certain category
- Identify which messages are of use, and update your <CATEGORIES_FILTER> element accordingly.

**Generate a detailed daily log file**

If you encounter an issue, you can generate a detailed daily log file to send to your support center. To create the detailed daily log file, use the ALL category filter name.

```xml
<LOG>
  <OUTPUT Type="DAILYFILE">/work/tmp/gas</OUTPUT>
  <FORMAT>date time relative-time process-id thread-id contexts event-type category 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 403 element.

Related concepts
Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

COMPONENT_LIST
The COMPONENT_LIST element provides a set of preset configurations such as runtime environment, time outs, auto logout, etc., that can be used by applications that share common configurations.

Syntax

```xml
<COMPONENT_LIST>
    [SERVICE_APPLICATION_EXECUTION_COMPONENT] [.....]
    [WEB_APPLICATION_EXECUTION_COMPONENT] [.....]
    [WEB_APPLICATION_TIMEOUT_COMPONENT] [.....]
    [AUTO_LOGOUT_COMPONENT] [.....]
    [SERVICE_APPLICATION_TIMEOUT_COMPONENT] [.....]
</COMPONENT_LIST>
```

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 SERVICE_APPLICATION_EXECUTION_COMPONENT elements.
- Zero to many WEB_APPLICATION_EXECUTION_COMPONENT elements.
- Zero to many WEB_APPLICATION_TIMEOUT_COMPONENT elements.
- Zero to many AUTO_LOGOUT_COMPONENT on page 377 elements.
- Zero to many UA_OUTPUT_COMPONENT on page 435 elements.
- Zero to many SERVICE_APPLICATION_TIMEOUT_COMPONENT elements.

Usage
You use this element to add components that define configurations such as runtime environment, time outs, auto logout, etc., for use by applications that share common configurations. When defining an application, you can then reference the component by its unique Id value.

Parent elements
This element is a child of the APPLICATION_SERVER on page 375 element.

COMMAND (for auto logout)
This is an element of AUTO_LOGOUT that provides a mechanism for the Genero Application Server to override an application's auto logout.

Syntax

```xml
<COMMAND Timeout=seconds>command-script</COMMAND>
```

1. The Timeout attribute specifies a timeout value. In the seconds parameter you specify a number of seconds as the time allowed for the command process to complete. If the system does not respond within this time, the GAS terminates the command process.
2. *command-script* is the name of the script or command to be run. You specify in this a check for conditions that would override the auto logout. An exit code value of zero is returned, if the auto logout is to be overridden, otherwise a non-zero value is returned.

**Child elements**

There are no child elements.

**Usage**

Use the `COMMAND` option to override the auto logout process. The command could be used, for example, to only allow auto logout between the hours of 21.00 pm and 08.00 am. This means that there would be no auto logout during work hours.

**Usage example**

```xml
<COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND>
```

In the example, the command runs a shell script that determines whether the auto logout takes place. The timeout for the command is set to 20 seconds. Table 97: How the command affects auto logout on page 382 shows how the command and the timeout can affect the auto logout.

**Table 97: How the command affects auto logout**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Exit code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The script runs. Condition to override auto logout is valid.</td>
<td>A zero exit code is returned.</td>
<td>The auto logout does not occur. The application keeps running.</td>
</tr>
<tr>
<td>The script runs. Condition to override auto logout is not valid.</td>
<td>A non-zero exit code is returned</td>
<td>The auto logout occurs as usual and stops the application.</td>
</tr>
<tr>
<td>The script may or may not run. The GAS times out the command process.</td>
<td>A non-zero exit code is returned</td>
<td>The auto logout occurs as usual and stops the application.</td>
</tr>
</tbody>
</table>

**Parent elements**

This element is a child of the following:

- `AUTO_LOGOUT` on page 375
- `AUTO_LOGOUT_COMPONENT` on page 377

**Related concepts**

TIMEOUT (for auto logout) on page 433

This TIMEOUT element sets a timeout to handle auto logout.

**CONFIGURATION**

The `CONFIGURATION` element is the starting point for the Genero Application Server configuration.

**Syntax**

```xml
<CONFIGURATION xmlns:xsi=uri 
   xsi:noNamespaceSchemaLocation=schema>
  <APPLICATION_SERVER>...</APPLICATION_SERVER>
</CONFIGURATION>
```

1. The `uri` defines the URL of the XML schema. The default value is "http://www.w3.org/2001/XMLSchema-instance".
Child elements
The CONFIGURATION element contains a single child element:

- One APPLICATION_SERVER element.

Usage
This element forms the root for all Genero Application Server configuration elements found in the as.xcf.

Example

```xml
<CONFIGURATION
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/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.

DELEGATE
The DELEGATE element specifies the Genero REST service in charge of handling requests for access to applications.

Syntax

```xml
<DELEGATE service="name">
  ...
  [/<anyparameter1>any-value</anyparameter1>
  [/<anyparameter2>any-value</anyparameter2>]
</DELEGATE>
```

1. name specifies the group and name of the Genero REST service.
2. All other child attributes are optional and, if present, are passed as parameters to the REST service using HTTP headers.

Important: Some Web servers convert all HTTP header names to lowercase. For example, a parameter called AnyParameter in the configuration, in the header may become:

```xml
x-fourjs-environment-parameter-anyparameter
```

Therefore, it is not recommended to rely on case in the naming of parameters as there is no guarantee that it will be preserved. When working with HTTP headers in your Genero program code, make sure you allow for this by converting them to either upper or lower case.

Child elements
When working with a Single Sign-on (SSO) solution, child elements of the DELEGATE element are 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.

Usage
You use this element 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.
You may also use the element to configure the SSO log out options at the close of the application with, for example, redirection to a logout URL. However, this will depend on how fully your IdP follows the standard logout protocol, which varies depending on the IdP. For configuration examples see Configure OpenID Connect SSO log out on page 147 and Configure SAML SSO log out on page 161.

The Genero Application Server dispatcher passes the request to the REST service identified as the delegation service if the type of request is defined by the /ua/r or /ws/r path segments in the application URI. See How to implement delegation on page 124.

Parent elements

This element is a child of one of the following elements:

- WEB_APPLICATION_EXECUTION_COMPONENT on page 437
- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 421
- EXECUTION (for an application) on page 390
- EXECUTION (for a service) on page 391

DELEGATE_OPTIONS (for a service)

The DELEGATE_OPTIONS element specifies options for a service using delegation.

Syntax

```xml
<DELEGATE_OPTIONS verb="OPTIONS"/>
```

1. The attribute verb can only have the value OPTIONS. Its purpose is to have the dispatcher transform the original HTTP request (POST, GET, PUT, DELETE) to an HTTP OPTIONS request. This allows for the optimization of the delegation mechanism because the original request body is not sent to the delegate service, only the headers are sent.

Child elements

There are no child elements.

Usage

You use this element to configure options about how a delegation service has to work.

Note: If the service is not configured with a delegation service, the element is ignored.

A typical use of the DELEGATE_OPTIONS feature would be to have the delegate service check for specific headers (for instance the HTTP authorization header) to see whether you want to forward the original request to the final proxy, without reading the entire original request body.

If the delegate service agrees to transmit it to the final proxy by returning the HTTP code 307 response, the dispatcher will then forward the original request plus the message body to the final proxy.

In the case that the delegate service returns something other than HTTP code 307, the original request body is discarded and the original request will get the delegate response (plus body) as response (standard delegation mechanism).

Usage example

```xml
<?xml version='1.0' encoding='windows-1252'?>
<APPLICATION Parent="ws.default" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.20/cfextws.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="FGLWSDEBUG">0</ENVIRONMENT_VARIABLE>
    <PATH>$(res.path.qa)/services/jira/gas-3501</PATH>
  </EXECUTION>
</APPLICATION>
```
In this example xcf file, the dispatcher transforms the request to an HTTP OPTIONS request: For an example of the output, see Optimize delegation with HTTP options on page 128.

### Parent elements
This element is a child of the following elements:
- EXECUTION (for a service) on page 391
- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 421

### Related concepts
Configure delegation for application or service on page 127
To delegate the start of an application or service to the Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

How to implement delegation on page 124
The Genero Application Server 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.

### DESCRIPTION
The DESCRIPTION element allows a short and a long description to be associated with an application definition.

#### Syntax
```
<DESCRIPTION>
  [<SHORT>...</SHORT>]
  [<LONG>...</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.

#### Usage
You use this element to provide descriptions of your application.

#### Example
```
<DESCRIPTION>
  <SHORT>A short description</SHORT>
  <LONG>A long description gives more detail</LONG>
</DESCRIPTION>
```
**DOCUMENT_ROOT**

The **DOCUMENT_ROOT** element specifies root directories that determine file system paths for serving files to applications.

**Syntax**

```
<DOCUMENT_ROOT>[path-list;...]</DOCUMENT_ROOT>
```

1. *path-list*, 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**

You can use this element to specify root directories for serving files to applications. The default document root directory (res.path.docroot) is $FGLASDIR/web.

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).

**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.

**Usage example**

```
<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).

**Parent elements**

This element is a child of the **INTERFACE_TO_CONNECTOR** on page 400

**DVM_AVAILABLE**

The **DVM_AVAILABLE** element specifies a timeout (in seconds) that the Genero Application Server allows for the DVM to start.

**Syntax**

```
<DVM_AVAILABLE>seconds</DVM_AVAILABLE>
```

1. *seconds* specifies in seconds the time allowed for the DVM to start. If not specified, the default time out is 10 seconds.

**Child elements**

There are no child elements.
Usage
You use this element to configure a timeout mechanism to handle the failure of the DVM to start. If the DVM has not started by the time the DVM_AVAILABLE timeout expires, the GAS sends an error message to the front-end and logs the message: DVM_AVAILABLE timeout expired

\[ <\text{DVMAVAILABLE}>10</\text{DVMAVAILABLE}> \]

In this usage example, the DVM available timeout is set to 10 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 a sub process in interactive mode (IN LINE MODE). If you run the sub process in the background (IN FORM MODE), the DVM_AVAILABLE timeout is not applicable.

Web services
For a Web service, as there is no concept of a front-end an error message is therefore not sent. In addition, a Web service is not expected to perform RUN commands with new DVMs connecting by themselves.

Parent elements
This element is a child of one of the following elements:

- WEB_APPLICATION_TIMEOUT_COMPONENT on page 439
- SERVICE_APPLICATION_TIMEOUT_COMPONENT on page 422
- TIMEOUT (for a service) on page 432
- TIMEOUT (for an application) on page 431

DVM
This element specifies the name of the Dynamic Virtual Machine.

Syntax

\[ <\text{DVM}>\text{name}</\text{DVM}> \]

1. \( \text{name} \) defines the name of the DVM. This value is typically \texttt{fglrun} for Linux®/UNIX™ Systems (UNIX) and \texttt{fglrun.exe} for Windows® Systems (WNT).

Child elements
There are no child elements.

Usage
You use this element to specify the DVM you want to use to start and run the application.

Usage example

\[ <\text{DVM}>\$(\text{res.dvm.wa})</\text{DVM}> \]

Parent elements
This element is a child of one of the following elements:
**END_URL**
This element specifies a URL that the user agent redirects to at end of the Web application.

**Syntax**

```xml
<END_URL>
  url
</END_URL>
```

1. `url` specifies the URL that the application is redirected to when it ends.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify a URL that the user agent redirects to at end of the Web application. The URL must start with "http://" or "https://". If not, the uaproxy will not start and you will get an error in the ua logs as follows:

```
18:39:01.599000 0.007 [uaproxy.c:530] 8640 0 - "SECURITY ERROR"
END_URL=www.4js.com must start with http:// or https://
```

**Usage example**

```xml
<END_URL>http://www.4js.com</END_URL>
```

**Parent elements**

When used in an application configuration file, it is the last child of the following element `APPLICATION (for an application)` on page 370.

**Related tasks**

- Configure OpenID Connect SSO log out on page 147
- Configure log out from the OpenID Connect Single Sign on (SSO) authentication server after an application ends.
- Configure SAML SSO log out on page 161
  Configure log out from the SAML SSO authentication server after an application ends.

**ENVIRONMENT_VARIABLE**
The `ENVIRONMENT_VARIABLE` element provides the value to be set for an environment variable.

**Syntax**

```xml
<ENVIRONMENT_VARIABLE
  Id=name
  Concat=
    "APPEND"
  "PREPEND"
>
  env
</ENVIRONMENT_VARIABLE>
```

1. `name` specifies the name of the environment variable.
2. `Concat` specifies a concatenation operator. It defines how an existing setting from a parent configuration is inherited. It is optional. The default is to omit the attribute and to ignore inherited values.
3. `env` sets the value of the environment variable.
**Child elements**

There are no child elements.

**Usage**

You use this type of element to set the environment for the application. Prior to the start of the application, the environment variable is set using this information.

```xml
<ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
```

In this example, the environment variable FGLGUI is set to "1", the default graphic user interface (GUI) mode. (The application starts in text user interface (TUI) mode if FGLGUI is set to "0".)

```xml
<ENVIRONMENT_VARIABLE Id="FGLWSDEBUG">3</ENVIRONMENT_VARIABLE>
```

In this example, the environment variable FGLWSDEBUG is set to turn on debug information display in Web services. (To enable Web Services debug, set FGLWSDEBUG to 1, 2, or 3 depending on the level of debug information required. If FGLWSDEBUG is set to "0", debug is turned off.)

**Examples with Concat attribute for FGLPROFILE**

You would use this configuration if your application or service is configured for its own FGLPROFILE in an external xcf file, but you also wanted to take settings from the default GAS as.xcf configuration as the parent configuration.

In this case, you would set the Concat attribute to "APPEND" or "PREPEND", depending on the order you want to handle the inherited settings. A separator is applied internally via the $(res.path.separator) resource.

**APPEND**

In this example, the Concat attribute is set to "APPEND" in order to take the FGLPROFILE environment variable setting coming from the parent configuration after the current one.

```xml
<ENVIRONMENT_VARIABLE Id="FGLPROFILE" Concat="APPEND">/myapp/myfglprofile</ENVIRONMENT_VARIABLE>
```

**PREPEND**

In this example, the Concat attribute is set to "PREPEND" in order to place the FGLPROFILE environment variable setting coming from the parent configuration before the current one.

```xml
<ENVIRONMENT_VARIABLE Id="FGLPROFILE" Concat="PREPEND">/opt/myfglprofile</ENVIRONMENT_VARIABLE>
```

**Default**

In this example, the parent configuration is discarded and only the current one is used. This is the default behavior. The Concat attribute is not used.

```xml
<ENVIRONMENT_VARIABLE Id="FGLPROFILE"/>/myapp/myfglprofile</ENVIRONMENT_VARIABLE>
```

See also [Application environment](#) on page 44
Parent elements

This element is a child of one of the following elements:

- **WEB_APPLICATION_EXECUTION_COMPONENT** on page 437
- **SERVICE_APPLICATION_EXECUTION_COMPONENT** on page 421
- **EXECUTION (for an application)** on page 390
- **EXECUTION (for a service)** on page 391

EXECUTION (for an application)

This EXECUTION element sets the runtime environment for an application by specifying parameters for executing it.

Syntax

```xml
<EXECUTION _ Using=component-id _
  _ AllowUrlParameters={"TRUE"|"FALSE"} _ _
  AllowUnsafeSession={"TRUE"|"FALSE"} _ _>

  [_<ENVIRONMENT_VARIABLE>... </ENVIRONMENT_VARIABLE> ] [...]
  [_<PATH>... </PATH> ]
  [_<DVM>... </DVM> ]
  [_<MODULE>... </MODULE> ]
  [_<PARAMETERS>... </PARAMETERS> ]
  [_<ACCESS_CONTROL>... </ACCESS_CONTROL> ]
  [_<DELEGATE>... </DELEGATE> ]
  [_<WEB_COMPONENT_DIRECTORY>... </WEB_COMPONENT_DIRECTORY> ]
</EXECUTION>
```

1. The **Using** attribute references an execution component, *component-id*, to inherit its parameters.
2. The **AllowUrlParameters** attribute 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. The **AllowUnsafeSession** attribute 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 124.

When working with a Single Sign-on (SSO) solution, child elements of the **DELEGATE** element are 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** on page 440 element.

For more information on defining execution components, see Service Application Execution Component.
Usage

You use this element to set the runtime environment for an application by specifying parameters for executing it. Execution settings may be defined by referencing a `WEB_APPLICATION_EXECUTION_COMPONENT` on page 437 element in its `Using` attribute, and/or by setting individual execution elements specific to the application.

When settings are inherited, settings you define locally within the `EXECUTION` element override the component settings.

Usage examples

```xml
<EXECUTION Using="cpn.wa.execution.local" />
```

```xml
<EXECUTION Using="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
</EXECUTION>
```

```xml
<EXECUTION AllowUrlParameters="TRUE" >
  <PATH>/home/examples/BuiltIn/Arguments</PATH>
</EXECUTION>
```

For examples using URL parameters, see:
- Application with arguments set in URL on page 49
- Set GBC customization in URL parameter on page 50

Parent elements

This element is a child of the `APPLICATION` (for an application) on page 370 element.

Related concepts

**EXECUTION (for a service)** on page 391
This `EXECUTION` element sets the runtime environment for a Web service application by specifying parameters for its execution.

**EXECUTION (for a service)**
This `EXECUTION` element sets the runtime environment for a Web service application by specifying parameters for its execution.

Syntax

```xml
<EXECUTION [ Using=service-component-id ]>
  [ <ENVIRONMENT_VARIABLE> ... </ENVIRONMENT_VARIABLE> ]
  [ <PATH> ... </PATH> ]
  [ <DVM> ... </DVM> ]
  [ <MODULE> ... </MODULE> ]
  [ <PARAMETERS> ... </PARAMETERS> ]
  [ <ACCESS_CONTROL> ... </ACCESS_CONTROL> ]
  [ <DELEGATE> ... </DELEGATE> ]
  [ <POOL> ... </POOL> ]
  [ <DELEGATE_OPTIONS> ... </DELEGATE_OPTIONS> ]
</EXECUTION>
```

1. The `Using` attribute references an execution component, `service-component-id`, to inherit its parameters.

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 POOL on page 411 element.
9. Zero or one DELEGATE_OPTIONS (for a service) on page 384 element.

Usage

You use this element to set the runtime environment for a Web service application by specifying parameters for executing it. Execution settings may be defined by referencing a SERVICE_APPLICATION_EXECUTION_COMPONENT element in its Using attribute, and/or by setting individual execution elements specific to the application.

When settings are inherited, settings you define locally within the EXECUTION element override the component settings.

Usage examples referencing components

```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 APPLICATION (for a service) on page 373.

Related concepts

EXECUTION (for an application) on page 390

This EXECUTION element sets the runtime environment for an application by specifying parameters for executing it.

FORMAT

The FORMAT element specifies the line output of a log message.

Syntax

```xml
<FORMAT Type="TEXT" > field-id[ ...]</FORMAT>
```

1. Type is an optional attribute whose value can only be "TEXT". When set, log messages are formatted in a single line of text, without line breaks. New line characters are escaped.
2. The field-id specifies a field identifier. You can include multiple field identifiers by listing multiple field identifiers, separated by spaces. The order of the field identifiers determines the order of the fields in each line of the log output. Table 98: Log message field identifiers on page 392 lists the valid field identifier names.

Table 98: Log message field identifiers

<table>
<thead>
<tr>
<th>Field Identifier Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Name of the category of the messages.</td>
</tr>
<tr>
<td></td>
<td>Tip: By including the category, you enable the person reading the logs to scan and filter messages based on their category type.</td>
</tr>
<tr>
<td>Field Identifier Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>component</td>
<td>Name of the component that generates 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 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>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.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify the format and content of the line output of a log message.

**Examples**

```xml
<FORMAT Type="TEXT">time event-type event-params</FORMAT>
```

```xml
<FORMAT>date time relative-time process-id thread-id contexts event-type event-params</FORMAT>
```

**Parent elements**

This element is a child of the LOG on page 403 element.

**Related concepts**

Log files on page 185
When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**GBC**

The GBC element specifies a customization directory for the Genero Browser Client (GBC) front-end.

**Note:** In versions of the GAS before 3.10, the GBC element was known as GWC-JS. Both names continue to be supported.

**Syntax**

```xml
<GBC>dir-file-name</GBC>
```

1. The *dir-file-name* can be either the name of a directory or a file.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify a customization directory for the GBC. This directory contains a collection of HTML, CSS, JavaScript files, and all the resources required for rendering Web applications.

The Genero Application Server (GAS) searches for the *dir-file-name* in `GBC_LOOKUP_PATH` on page 395 paths. If found, it is processed as determined by these conditions:

- If it is a directory, it is expected to be a GBC installation.
- If it is a file, then the file is expected to contain one text line, giving the name of a directory located in the same directory as the file. This directory is expected to be a GBC installation.

**Usage example: default application**

The bootstrap directories are defined in the $(res.gbc) resource. The GBC element in the defaultwa abstract application configuration references this resource:

```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"/>
  <UA_OUTPUT>
    <PROXY>$(res.uaproxy.cmd)</PROXY>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <TIMEOUT Using="cpn.wa.timeout"/>
    <GBC>$(res.gbc)</GBC>
  </UA_OUTPUT>
</APPLICATION>
```

The value of the $(res.gbc) is set by $(res.gwc-js) in the as.xcf file to reference the "_default" file. This file is located in the $(res.fgldir)/web_utilities/gbc directory, which refers to the $FGLDIR/web_utilities/gbc/gbc directory.

```xml
<RESOURCE Id="res.gwc-js" Source="INTERNAL">_default</RESOURCE>
<RESOURCE Id="res.gbc" Source="INTERNAL">$(res.gwc-js)</RESOURCE>
```

**Note:** If the _default file content is changed, the next application started referencing the $(res.gwc-js) resource uses this new value.
Usage example: customization project directory

\[
\text{<GBC}\text{gbc-custom}</GBC>
\]

In this example, you create a new set of files to customize the look and feel of a Web application. You place these files in a separate directory to the default bootstrap. In this example, that file is $(res.path.as)/gbc-custom. The GAS searches for the GBC directory at paths defined by the GBC_LOOKUP_PATH.

Usage example: customization file

\[
\text{<GBC}\text{customization_file</GBC>}
\]

A text file (customization_file in the example) is used to specify a customization directory. Paths defined by the GBC_LOOKUP_PATH element, are searched to locate the customization_file.

The first line of the file must contain the name of a directory located in the same directory as the file, where a GBC customization installation is found.

URI gbc query string parameter

The customization directory can also be provided in the application URL using the gbc query string parameter. If the URI contains a gbc query string and value, this takes precedence over the configuration for the GBC element in the application configuration file (xcf).

Note: In the Genero web client prior to version 3.00, the parameter gwc-js was used. From version 3.10, it is renamed gbc. Use of gwc-js is deprecated but remains for compatibility.

See Set GBC customization in URL parameter on page 50.

Parent elements

This element is a child of UA_OUTPUT on page 434

GBC_LOOKUP_PATH

The GBC_LOOKUP_PATH element specifies the location of installed Genero Browser Client (GBC) front-ends.

Note: In versions of the GAS before 3.10, the GBC_LOOKUP_PATH element was known as GWC_JS_LOOKUP_PATH. Both names continue to be supported.

Syntax

\[
\text{<GBC_LOOKUP_PATH>path-list[(; ...)]</GBC_LOOKUP_PATH>}
\]

1. The path-list allows for multiple directory paths to be specified. The separator between resource paths is a semi-colon (;).

Child elements

There are no child elements.

Usage

You use this element to specify the path to the GBC on page 394. Examples are shown of how to specify this using the GBC_LOOKUP_PATH and with dispatcher overrides.

Example using GBC_LOOKUP_PATH

\[
\text{<GBC_LOOKUP_PATH>${res.path.gbc.user};${res.gbc.deployment};${res.fgldir}/web_utilities/gbc</GBC_LOOKUP_PATH>}
\]
The dispatcher searches for the GBC files in the GBC_LOOKUP_PATH paths. If the directory is found, the requested file is sent. If the file does not exist, no additional searching is performed on that path. This is to avoid the risk of mixing up files in other GBC versions that may be located in various directories in these paths. The following paths are defined for the location of the GBC:

- **res.path.gbc.user** is not defined in the GAS configuration file by default, however it can be set as required at runtime. See Set GBC_LOOKUP_PATH with dispatcher override on page 396.
- **res.gbc.deployment** resource points to the $(res.appdata.path)/gbc_deployment directory. It contains your deployed GBC clients.
- If the directory specified by the GBC is not found, the default GBC directory at $(res.fgldir)/web_utilities/gbc path, which is the last item of the GBC_LOOKUP_PATH list, is used.

You can add the path to your GBC directory to the list:

**Tip:** The dispatcher searches the paths specified in order, therefore it is recommended to place the path to your GBC directory first in the path list.

**Set GBC_LOOKUP_PATH with dispatcher override**

httpdispatch -E res.path.gbc.user=<directory-of-your-choice>

This example sets the location of your GBC files with the dispatcher switch (-E) by creating or overwriting the res.path.gbc.user resource with the path to the directory holding the folder.

**Parent elements**

This element is a child of INTERFACE_TO_CONNECTOR on page 400.

**Related concepts**

Configuring GBC client for applications on page 263

There are different methods for specifying the GBC client your applications use depending on whether you want to use the same client for all applications, or a given client for a specific application.

**Related tasks**

Provide GBC in application path on page 262

Describes a method for providing a GBC client in your application path.

**GDC_SHORTCUT**

The GDC_SHORTCUT element specifies a path with a shortcut to the Genero Desktop Client (GDC).

**Syntax**

```
<GDC_SHORTCUT>path</GDC_SHORTCUT>
```

1. The **path** specifies the path to shortcuts of type gdc.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify a path to shortcuts of type gdc that you can use to launch Web applications locally to your GDC.

**Usage example**

```
<GDC_SHORTCUT>${(res.deployment.path)}/tpl/shortcut/gdc-http.gdc</GDC_SHORTCUT>
```
Parent elements
This element is a child of UA_OUTPUT on page 434.

GROUP (for an application)
This GROUP element allows you to specify a directory where external application configuration files are located.

Syntax

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

Note:
1. `name` is a string that uniquely identifies the group.
2. `path` is the path to the directory where the external application configuration files are to be placed.

Child elements
There are no child elements.

Usage
You use this element to declare a group and specify a directory where external application configuration files belonging to this group 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 locates and uses that file without having to be restarted.

Usage example

```xml
<GROUP id="tut-demo">$(res.path.as.demo)/tut/app</GROUP>
<GROUP id="mygroup">/home/myuser/config</GROUP>
```

URL with group
You use the alias for the group in the application URL to specify where the application configuration file is found.

```
http://server/gas/ua/r/mygroup/appl
```

Parent elements
This element is a child of the APPLICATION_LIST on page 374 element.

Related concepts
Create an application group on page 112
Use groups to define aliases for directories where application configuration files can be stored.

GROUP (for a service)
This GROUP element allows you to specify a directory where external Web service application configuration files are located.

GROUP (for a service)
This GROUP element allows you to specify a directory where external Web service application configuration files are located.

Syntax

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

1. `name` is a string that uniquely identifies the group.
2. *path* is the path to the directory where the external Web service application configuration files are to be placed.

**Child elements**

There are no child elements.

**Usage**

You use this element to declare a group and specify a directory where external application configuration files belonging to this group 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 locates and uses that file without having to be restarted. You use the alias for the group in the application URL to specify where the application configuration file is found.

**Usage example**

```
<GROUP id="mygroup">/home/myuser/config</GROUP>
```

**URL with group**

You use the alias for the group in the application URL to specify where the service application configuration file is found.

```
http://webserver/gas/ws/r/mygroup/app1
```

**Parent elements**

This element is a child of the SERVICE_LIST on page 424 element.

**Related concepts**

- **Create an application group** on page 112
- Use groups to define aliases for directories where application configuration files can be stored.
- **GROUP (for an application)** on page 397
- This GROUP element allows you to specify a directory where external application configuration files are located.

**HEADER**

The HEADER element defines the request and response type communication carried on the HTTP protocol between Web applications, Web services and the client.

**Syntax**

```
<HEADER Name=id> value-list[;...]</HEADER>
```

1. *id* defines the unique identifier for the HTTP header.
2. *value-list* is a list of values separated by semi-colons.

**Child elements**

There are no child elements.

**Usage**

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.

You use this element to define HTTP headers. HTTP headers can be configured for applications using the APPLICATION element and for Web services using the SERVICE element.
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.

  Warning: On Internet Explorer 11, use of this header may cause images not to be displayed if the image files do not have extensions. To work around this, if your applications need to serve images through Genero Application Server (GAS), make sure your image files have extensions.
- 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/>
  </HTTP>
</INTERFACE_TO_CONNECTOR>
```

Parent elements

This element is a child of the following elements:

- APPLICATION (for HTTP) on page 372
- SERVICE (for HTTP) on page 423

HTTP

The `HTTP` element contains elements defining the communication carried on the HTTP protocol between Web applications and the client.

Syntax

```xml
<HTTP>
  <SESSION_COOKIE>...</SESSION_COOKIE>
  <CACHE_CONTROL_MAX_AGE>...</CACHE_CONTROL_MAX_AGE>
  <APPLICATION>...</APPLICATION>
  <SERVICE>...</SERVICE>
</HTTP>
```

Child elements

The `HTTP` element may contain the following child elements:

- One `SESSION_COOKIE` on page 425 element
- One `CACHE_CONTROL_MAX_AGE` on page 378 element
- One `APPLICATION` (for HTTP) on page 372 element
- One `SERVICE` (for HTTP) on page 423 element
Usage

You use the HTTP element to define a secure flag for cookies, a cache control header, and the HTTP headers for applications and/or services. It therefore can contain APPLICATION and SERVICE elements.

Example

```xml
<INTERFACE_TO_CONNECTOR>
  ...
  <HTTP>
    <SESSION_COOKIE Secure="TRUE"></SESSION_COOKIE>
    <CACHE_CONTROL_MAX_AGE>300</CACHE_CONTROL_MAX_AGE>
    <APPLICATION>
      <HEADER/>
    </APPLICATION>
    <SERVICE>
      <HEADER/>
    </SERVICE>
  </HTTP>
</INTERFACE_TO_CONNECTOR>
```

Parent elements

This element is a child of INTERFACE_TO_CONNECTOR on page 400.

INTERFACE_TO_CONNECTOR

The INTERFACE_TO_CONNECTOR element specifies the connection between the GAS and the GAS Connector located with the Web server.

Syntax

```xml
<INTERFACE_TO_CONNECTOR>
  [...]
  [...]
  [...]
  [...]
  [...]
  [...]
  [...]
</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 421 element.
2. Zero or one LISTEN on page 402 element.
3. Zero or one TCP_ADMIN_PORT on page 429 element.
4. One TCP_BASE_PORT on page 429 element.
5. One TCP_PORT_OFFSET on page 430 element.
6. One DOCUMENT_ROOT on page 386 element.
7. One GBC_LOOKUP_PATH on page 395 element.
8. One `TEMPORARY_DIRECTORY` on page 431.
10. One `REPORT_VIEWER_DIRECTORY` on page 416 element.
11. Zero or one `REPORT_REMOTE_URL_PREFIX` on page 416 element.
12. One `SOCKET_FAMILY` on page 427 element.
13. One `SOCKET_PATH` on page 428 element.
14. Zero or one `HTTP` on page 399 element.

**Usage**

You use the `INTERFACE_TO_CONNECTOR` element to define various configurations for the Genero Application Server, such as the port it listens to for incoming requests. This configuration applies for all GAS connectors including the standalone GAS (httpdispatch).

You can have several instances of the GAS running concurrently on the same host, but you need to make sure that each has a separate GAS configuration file with different connection configurations including port offsets. Once started, each application server listens at the offset specified.

**Important:** If you create multiple GAS configuration files, ensure that the port values are unique for each application server started. If two GAS 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.

**Usage example**

```xml
<INTERFACE_TO_CONNECTOR>
    <ROOT_URL_PREFIX></ROOT_URL_PREFIX>
    <LISTEN></LISTEN>
    <TCP_BASE_PORT>6300</TCP_BASE_PORT>
    <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
    <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
    <GBC_LOOKUP_PATH>$(res.path.gwcjs.user);$(res.path.docroot)</GBC_LOOKUP_PATH>
    <TEMPORARY_DIRECTORY>$(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)`.

**Parent elements**

This element is a child of the `APPLICATION_SERVER` on page 375 element.

**INTERFACE_TO_DVM**

The `INTERFACE_TO_DVM` element specifies the address of the host where the GAS dispatcher runs.

**Syntax**

```xml
<INTERFACE_TO_DVM>
    <ADDRESS>...</ADDRESS>
</INTERFACE_TO_DVM>
```
**Child elements**
The `INTERFACE_TO_DVM` element contains a child element:

1. One `ADDRESS` element.

**Usage**
You use this element to specify 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.

**usage**

```xml
<INTERFACE_TO_DVM>
  <ADDRESS>app_server</ADDRESS>
</INTERFACE_TO_DVM>
```

**Parent elements**
This element is a child of one of the following elements: [APPLICATION_SERVER](on page 375)

**KEEP_ALIVE**
The `KEEP_ALIVE` element specifies how long (in seconds) the proxy waits before shutting down when no longer serving requests.

**Syntax**

```xml
<KEEP_ALIVE>seconds</KEEP_ALIVE>
```

1. `seconds` specifies how long the proxy waits before shutting down when no longer serving requests.

**Child elements**
There are no child elements.

**Usage**
You use this element to specify a time out for the proxy before it shuts down due to inactivity. If the `KEEP_ALIVE` element is not specified, the proxy will never shutdown. If it is specified, it prevents the DVMs from running indefinitely in the unlikely event that the dispatcher or the Web server crashes.

**Usage example**

```xml
<KEEP_ALIVE>10</KEEP_ALIVE>
```

**Parent elements**
This element is a child of the following:

- [SERVICE_APPLICATION_TIMEOUT_COMPONENT](on page 422)
- [TIMEOUT (for a service)](on page 432)

**LISTEN**
The `LISTEN` element specifies the IP address of the machine the dispatcher listens on.

**Syntax**

```xml
<LISTEN>
```
LISTEN is optional.

Child elements
1. Zero or one ADDRESS (LISTEN) on page 369 elements.

Usage
You use this element to specify the IP address of the machine the dispatcher uses to listen for incoming requests. If not defined, the GAS will accept connections on any interface. For security reasons it is recommended to configure the dispatcher to only accept connections coming from the web server (Apache for instance) on one IP address.

Parent elements
This element is a child of INTERFACE_TO_CONNECTOR on page 400

LOG
The LOG element specifies details of the log files the Genero Application Server creates.

Important: Sensitive and personal data may be written to the output. Make sure that the log output is written to files that can only be read by administrators, and review the management strategy for log files.

Syntax

```<LOG>
  <OUTPUT>...</OUTPUT>
  <FORMAT>...</FORMAT>
  <CATEGORIES_FILTER>...</CATEGORIES_FILTER>
  [<RAW_DATA>...</RAW_DATA>]
</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.

Usage
You use the LOG element to specify where log files are created, the format of the log messages, the type of information logged, and the maximum size of a single log message. The GAS creates logs for each of its dispatchers, proxies, and DVMs.

It is possible to specify multiple LOG elements. For example, you may need to have separate log files capturing different categories of messages.

Usage example

```<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>
```
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 375 element.

**Related concepts**
- [Log files](#) on page 185
- When you run an application, the Genero Application Server (GAS) creates log files which may be viewed for troubleshooting.

**LONG**
The LONG element contains the long description to be associated with an application definition.

**Syntax**

```xml
<LONG>description</LONG>
```

1. *description* specifies the long description.

**Child elements**
There are no child elements.

**Usage**
You use this element to provide a detailed description of your application.

**Usage example**

```xml
<DESCRIPTION>
  <SHORT></SHORT>
  <LONG>A long description describes the application in more detail</LONG>
</DESCRIPTION>
```

**Parent elements**
This element is a child of the [DESCRIPTION](#) on page 385 element.

**Related concepts**
- [SHORT](#) on page 426
  - The SHORT element contains the short description to be associated with an application definition.

**MAX_AVAILABLE**
The MAX_AVAILABLE element specifies the maximum number of available DVMs to be attached to a Web service.

**Syntax**

```xml
<MAX_AVAILABLE> max </MAX_AVAILABLE>
```

1. *max* specifies the maximum number of DVMs to be attached to the Web service.

**Child elements**
There are no child elements.
Usage

You use `MAXAVAILABLE` to control the number of DVMs - and in effect the number of licenses - used by each application. You can control your license usage using `MAXAVAILABLE`. By using a different FGLPROFILE file for each application, you can specify X licenses for application 1, Y licenses for application 2.

Constraints

`START` on page 428 and `MINAVAILABLE` on page 406 can not exceed `MAXAVAILABLE` DVMs.

```markdown
START <= MAXAVAILABLE
MINAVAILABLE <= MAXAVAILABLE
```

Parent elements

This element is a child of the `POOL` on page 411 element.

Related concepts

Licensing examples on page 65

Some scenarios and examples are illustrated where applications and GAS configurations may help you use licenses more efficiently.

`MAXREQUESTS_PER_DVM`

The `MAXREQUESTS_PER_DVM` element specifies the maximum number of requests a DVM can handle before being stopped by the pool manager.

Syntax

```xml
<MAXREQUESTS_PER_DVM> max </MAXREQUESTS_PER_DVM>
```

1. `max` specifies the maximum number of requests allowed per DVM.

Child elements

There are no child elements.

Usage

You use this element to set the number of requests a Web service DVM is allowed to process. In some circumstances a DVM must be stopped after it has processed one request. The value must be equal to or greater than 1.

Parent elements

This element is a child of the `POOL` on page 411 element.

`MONITOR`

The `MONITOR` element specifies the hosts allowed to access the monitor page of the GAS.

Syntax

```xml
<MONITOR>
  [<ALLOW_FROM>...</ALLOW_FROM>] [...]
</MONITOR>
```

Child elements

- Zero or more `ALLOW_FROM` on page 370 elements.
**Usage**

You use this element to specify the IP address of the machine from where 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.

**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:

```
<MONITOR>
  <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
</MONITOR>
```

To (for example):

```
<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.".

**Parent elements**

This element is a child of the [APPLICATION_SERVER](#) element.

**MIN_AVAILABLE**

The MIN_AVAILABLE element specifies the minimum number of available DVMs to be attached to a Web Service.

**Syntax**

```
<MIN_AVAILABLE> min </MIN_AVAILABLE>
```

1. `min` specifies the minimum number of DVMs to be attached to the Web service.

**Child elements**

There are no child elements.

**Usage**

You use MIN_AVAILABLE to specify the minimum number of DVMs to be available for a Web Service.
Constraint

MIN_AVAILABLE can be zero but it can not exceed MAX_AVAILABLE on page 404.

\[ 0 \leq MIN\_AVAILABLE \leq MAX\_AVAILABLE \]

MIN_AVAILABLE can be either less than or greater than the value specified by START on page 428.

If START is greater than MIN_AVAILABLE, the number of DVMs can decrease to reach MIN_AVAILABLE.

Parent elements

This element is a child of the POOL on page 411 element.

MODULE

The MODULE element specifies the application module name.

Syntax

\[ <\text{MODULE}>\text{name}</\text{MODULE}> \]

1. name specifies the name of the application module to run.

Child elements

There are no child elements.

Usage

You use this element to specify the application module name (the name of the .42r module) you want to run.

While this element can be specified as part of an execution component, it is typically defined at the application level.

If the MODULE element is omitted, the Genero Application Server uses the name of the requested application.

<MODULE>Edit</MODULE>

Tip: If you receive an error that states "Configuration" FGL_VMPROXY_COMMAND_LINE is missing", the MODULE has not been set. Verify that you have provided the MODULE element in your application or service configuration (or in the parent configuration).

Parent elements

This element is a child of one of the following elements:

- WEB_APPLICATION_EXECUTION_COMPONENT on page 437
- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 421
- EXECUTION (for an application) on page 390
- EXECUTION (for a service) on page 391

OUTPUT (under LOG)

The OUTPUT element specifies where the log messages are sent or written.

Syntax

\[ <\text{OUTPUT} \text{Type=} \{ "DAILYFILE" | "CONSOLE" | "CONSOLE,DAILYFILE" \} >[\text{path}]</\text{OUTPUT}> \]

1. The Type attribute specifies whether log message are output to the console or written to a log file, or both. See Table 99: Valid values for the Type attribute on page 408.
2. *path* specifies the path to the log file. If the path value is not specified, the default of `$FGLASDIR/log` is used.

   **Note:** The default Genero Application Server configuration file (`as.xcf`) 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 99: Valid values for the Type attribute

<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 daily and are stored in a directory named by date in the format <code>YYYYMMDD</code>. If <code>path</code> is specified, the daily directory is created and stored under its <code>log</code> directory. If <code>path</code> 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>

**Usage**

You use this element to specify where log messages are displayed or written.

### Output configuration examples

- `<OUTPUT Type="DAILYFILE"/>`
  
  In this example, the daily log file is written to the default logging directory.

- `<OUTPUT Type="DAILYFILE">$(res.as.dir)/logdirs</OUTPUT>`
  
  In this example, the daily directory is created in `$(res.as.dir)/logdirs/log`.

- `<OUTPUT Type="CONSOLE"/>`
  
  In this example, log messages are sent to standard output.

- `<OUTPUT Type="CONSOLE,DAILYFILE"/>`
  
  In this 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 the `LOG` on page 403 element.

**PARAMETER**

The `PARAMETER` element specifies a parameter to provide on the DVM command line.

### Syntax

```xml
<PARAMETER>value</PARAMETER>
```

1. *value* specifies the parameter value.
**Child elements**

There are no child elements.

**Usage**

You use this element to provide parameters for the start of the DVM on the command line. Parameters are specified within the `PARAMETERS` element.

**Usage examples**

The following example provides two parameters:

- Hello world!
- Again

```xml
<PARAMETERS>
  <PARAMETER>Hello world!</PARAMETER>
  <PARAMETER>Again</PARAMETER>
</PARAMETERS>
```

**Parent elements**

This element is a child of `PARAMETERS` on page 409.

**PARAMETERS**

The `PARAMETERS` element acts as a parent container for parameters provided on the DVM command line.

**Syntax**

```xml
<PARAMETERS>
  ...</PARAMETER>
</PARAMETERS>
```

**Child elements**

- Zero or more `PARAMETER` on page 408 elements.

**Usage**

You use this element to provide parameters for the start of the DVM on the command line.

**Usage examples**

The following example provides two parameters:

- Hello world!
- Again

```xml
<PARAMETERS>
  <PARAMETER>Hello world!</PARAMETER>
  <PARAMETER>Again</PARAMETER>
</PARAMETERS>
```

**Parameters with URL arguments**

If the `AllowUrlParameters` attribute in the `EXECUTION (for an application)` on page 390 is set to `TRUE`, parameters can be passed as arguments in the application URL using the format “Arg=value”. For
example, the "myApp" application is launched with two arguments by the "myWebServer" Web server:

```
http://myWebServer/gas/ua/r/myApp?
Arg=Val1&Arg=Val2
```

If URL parameters are allowed, these parameters are listed after the ones defined in the PARAMETERS element of the configuration file.

**Note:** For Web services applications, the AllowUrlParameters attribute is not supported.

**Parent elements**

This element is a child of one of the following elements:

- WEB_APPLICATION_EXECUTION_COMPONENT on page 437
- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 421
- EXECUTION (for an application) on page 390
- EXECUTION (for a service) on page 391

**PATH (under EXECUTION)**

The PATH element specifies the current working directory for the application module.

**Syntax**

```
<PATH>path</PATH>
```

1. *path* specifies the path to the application module.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify the path to the application MODULE on page 407.

```
<PATH>/home/appdir/sales/</PATH>
```

**Tip:** If you receive an error that states "Configuration" FGL_VMPROXY_COMMAND_DIR is missing", the PATH has not been set. Verify that you have provided the PATH element in your application or service configuration (or in the parent configuration).

**Parent elements**

This element is a child of one of the following elements:

- WEB_APPLICATION_EXECUTION_COMPONENT on page 437
- SERVICE_APPLICATION_EXECUTION_COMPONENT on page 421
- EXECUTION (for an application) on page 390
- EXECUTION (for a service) on page 391
**PLATFORM_INDEPENDENT**

The **PLATFORM_INDEPENDENT** element contains a list of platform-independent resources, available for both Linux®/UNIX™ and Windows® platforms.

**Syntax**

```xml
<PLATFORM_INDEPENDENT>
  [...]
</PLATFORM_INDEPENDENT>
```

**Child elements**

A **PLATFORM_INDEPENDENT** element main contain the following child elements:

- Zero to many **RESOURCE** elements.

**Usage**

This element contains a list of platform-independent resources, available for both Linux®/UNIX™ and Windows® platforms.

```xml
<PLATFORM_INDEPENDENT>
  <RESOURCE Id="res.fglgui" Source="INTERNAL">1</RESOURCE>
  ...
</PLATFORM_INDEPENDENT>
```

**Parent elements**

This element is a child of the **RESOURCE_LIST** on page 420 element.

---

**POOL**

The **POOL** element sets the limitations regarding the number of Virtual Machines (DVMs) that are attached to a Web service.

**Syntax**

```xml
<POOL>
  [...]
</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.

**Usage**

You use this element to configure how the Web service is managed. 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.

**Pool example**

```xml
<POOL>
  <START>5</START>
  <MIN_AVAILABLE>3</MIN_AVAILABLE>
  <MAX_AVAILABLE>10</MAX_AVAILABLE>
  <MAX_REQUESTS_PER_DVM>1</MAX_REQUESTS_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 Pools topic.

**Parent elements**

This element is a child of one of the following elements:

- **SERVICE_APPLICATION_EXECUTION_COMPONENT** on page 421
- **EXECUTION (for a service)** on page 391

**PROMPT (for auto logout)**

The **PROMPT** element provides a feature that allows a user to resume an application after an auto logout event.

**Note:** The re-login prompt feature is supported on the GBC (starting from version GBC 1.00.38). It is not supported on Genero Mobile front-ends (GMA, GMI), and the Genero Desktop Client (GDC). If used, an HTTP 410 error page is displayed.

**Syntax**

```xml
<PROMPT Timeout=seconds Type={"URL" | "DELEGATE"}>url</PROMPT>
```

1. **seconds** is the number of seconds as the allowed time for a user to re-log in.
2. The **Type** attribute has a value of either **URL** or **DELEGATE**. This attribute is mandatory. See Security considerations on page 413.
3. **url** is either the application URL or the delegation service URL, depending on the value set by **Type**.

**Usage**

You use this element to configure a re-log-in prompt, which allows a user to resume activity after an auto logout event is detected. A message is displayed for a set period of time notifying the user that a re-log in is needed to continue. If the user resumes the application during the prompt's timeout period, the application resumes, otherwise the application stops.

**Usage example: Type URL**

In this example a timeout duration is set to 200 seconds, which means that a page is displayed for that period of time notifying the user that a re-log in is needed to continue. If the user clicks on the re log button, the user agent is redirected to the specified URL.

```xml
<AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
  <TIMEOUT>0</TIMEOUT>
  <PROMPT Timeout="200" Type="URL">http://localhost:6394/prompt.html</PROMPT>
</AUTO_LOGOUT_COMPONENT>
```
Usage example: Type Delegate

In this example, the user agent is redirected to the specified delegation service.

```xml
<AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
  <TIMEOUT>0</TIMEOUT>
  <PROMPT Timeout="200" Type="DELEGATE">
    services/OpenIDConnectServiceProvider
  </PROMPT>
</AUTO_LOGOUT_COMPONENT>
```

Security considerations

When using PROMPT of Type URL, no security check is done to ensure that the request to resume the application comes from a valid user. Therefore you must use prompt of type URL only for prototyping purposes.

We recommend you use the DELEGATE mechanism for security reasons, because when the /ua/resume request is forwarded to the delegate service, you can check additional tokens to ensure the user has been properly identified before resuming the application.

Parent elements

This element is a child of the following elements:

- [AUTO_LOGOUT_COMPONENT on page 377](#)
- [AUTO_LOGOUT on page 375](#)

Related concepts

- [How autologout prompt is implemented on SSO on page 165](#)
  - The prompt feature can authenticate the user and resume the application after an auto-logout event.

Related tasks

- [Authorize re-log in with OpenID Connect SSO on page 147](#)
  - Use this procedure to configure re-log in to a Genero application authenticated by OpenID Connect SSO after an auto logout event.

PROXY (for an application)

This PROXY element specifies the proxy you want to use for an application or a set of applications.

Syntax

```xml
<PROXY>name</PROXY>
```

1. `name` is the name of the proxy executable to use.

Child elements

There are no child elements.

Usage

You use this element to specify the proxy you want to use for an application or a set of applications.

```xml
<PROXY>$(res.uaproxy.cmd)</PROXY>
```

The `$(res.uaproxy.cmd)` resource references the proxy command `uaproxy` required for Genero Desktop Client (GDC) and Genero Browser Client (GBC) applications.

Parent elements

This element is a child of the [UA_OUTPUT on page 434 element](#).
Related concepts

**PROXY (for a service)** on page 414
This PROXY element specifies the proxy to use for Web services.

**PROXY (for a service)**
This PROXY element specifies the proxy to use for Web services.

**Syntax**

```
<PROXY>name</PROXY>
```

1. `name` is the name of the proxy executable to use.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify the proxy you want to use for a Web service.

```
<PROXY>$(res.gwsproxy)</PROXY>
```

The `$(res.gwsproxy)` resource references the default proxy gwsproxy required for Web service applications.

**Parent elements**

This element is a child of the **APPLICATION (for a service)** on page 373 element.

**Related concepts**

**Proxy** on page 39
Proxies deliver applications on the GAS. Being aware of the different types of proxies there are, assists you in developing and deploying applications accordingly.

**PUBLIC_IMAGEPATH**

The PUBLIC_IMAGEPATH element defines the public resources directory used by applications.

**Syntax**

```
<PUBLIC_IMAGEPATH>path</PUBLIC_IMAGEPATH>
```

1. Where `path` is a path to your application resources.

**Child elements**

There are no child elements.

**Usage**

Use the PUBLIC_IMAGEPATH element to define a path to your public resources directory used by the DVM or fglrun to look for resources used by applications. This path is relative to the root path appdata/public. The "appdata" resource, as it is commonly known, is set by the resource `$(res.appdata.path)`.

**Usage example with resource**

```
... <UA_OUTPUT>
    <PROXY></PROXY>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
```
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.

```xml
<RESOURCE Id="res.public.resources" Source="INTERNAL">common</RESOURCE>
```

Usage example with path

```xml
... 
<UA_OUTPUT>
  <PROXY/>
  <PUBLIC_IMAGEPATH>myapp/newpictures</PUBLIC_IMAGEPATH>
  <GBC>gwc-js</GBC>
  <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.

Parent elements

This element is a child of the following element: UA_OUTPUT on page 434

**RAW_DATA**
The RAW_DATA element limits the size of a single log message.

Syntax

```xml
<RAW_DATA MaxLength=length></RAW_DATA>
```

1. Where *length* is the number of characters after which the log message is truncated. The value must be a non-negative integer.

Child elements

There are no child elements.

Usage

You use this element to specify the length of log messages. Log messages can include the complete HTML response, and can therefore be very large (an entire HTML page). This element is optional but if specified the maximum number of characters in any single log message is set.

If the RAW_DATA element is omitted, data is logged in its entirety.

Example

```xml
<RAW_DATA MaxLength="100" />
```

Parent elements

This element is a child of the LOG on page 403 element.
REPORT_REMOTE_URL_PREFIX
Specifies the URL prefix of the server where the Genero Report Engine (GRE) is running.

Syntax

```
<REPORT_REMOTE_URL_PREFIX>url</REPORT_REMOTE_URL_PREFIX>
```

1. `url` specifies the URL prefix of the server where the Genero Report Engine (GRE) is running.

The REPORT_REMOTE_URL_PREFIX element is optional and it does not support any attributes.

Child elements
There are no child elements.

Usage
Use the REPORT_REMOTE_URL_PREFIX to set the URL of the GRE server if operating on a different server to the DVM.

Usage example

```
<REPORT_REMOTE_URL_PREFIX>http://remotehost:12345</REPORT_REMOTE_URL_PREFIX>
```

Parent elements
This element is a child of INTERFACE_TO_CONNECTOR on page 400.

Related concepts
Configure GRE in distributed mode on remote machine on page 174
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.

REPORT_VIEWER_DIRECTORY
The REPORT_VIEWER_DIRECTORY element specifies the directory used by the Genero Web Report Viewer.

Syntax

```
<REPORT_VIEWER_DIRECTORY>path</REPORT_VIEWER_DIRECTORY>
```

1. `path` specifies the path to the report viewer directory.

The REPORT_VIEWER_DIRECTORY element does not support any attributes or have any child elements.

Child elements
There are no child elements.

Usage
Use the REPORT_VIEWER_DIRECTORY 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 Configuring Genero Report Engine on GAS on page 173.
Usage example

```xml
<REPORT_VIEWER_DIRECTORY>${res.gredir}/viewer</REPORT_VIEWER_DIRECTORY>
...
```

Parent elements

This element is a child of **INTERFACE_TO_CONNECTOR** on page 400.

REQUEST_RESULT (for an application)

This **REQUEST_RESULT** element specifies the number of seconds the GAS waits for the DVM to respond to pending transactions.

Syntax

```xml
<REQUEST_RESULT>seconds</REQUEST_RESULT>
```

1. **seconds** specifies the number of seconds to wait for a response from a transaction request.

Child elements

There are no child elements.

Usage

You use the **REQUEST_RESULT** timeout to provide information to the user when a transaction is taking longer than expected. After the timeout expires, the GAS 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 keep-alive 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 sends a GET request to the GAS immediately after a response. Meanwhile, the GAS stores data sent by the DVM for the application in its buffer, waiting for a GET request from the front-end. When the GET request is received by the GAS, these conditions determine the response:

- 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 GAS waits and, if the DVM is still processing the request after the specified **REQUEST_RESULT** timeout expires, it sends the keep-alive response to the front-end and resets the **REQUEST_RESULT** timer.

**Important:** The **REQUEST_RESULT** timeout has an initial setting of 60 seconds. When configuring this setting, you must ensure it is less than the Common Gateway Interface (CGI) timeout of the web server you are using. See [FastCGI Installation and Web Server Configuration](#) on page 98.

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 the following:

- **WEB_APPLICATION_TIMEOUT_COMPONENT** on page 439
- **TIMEOUT (for an application)** on page 431
Related tasks
Apache 2.4: mod_proxy_fcg on page 100
With Apache 2.4, mod_proxy_fcg is used instead of mod_fastcgi.

REQUEST_RESULT (for a service)
This REQUEST_RESULT element specifies the number of seconds the GAS waits for the DVM to respond to pending transactions.

Syntax

```xml
<REQUEST_RESULT>seconds</REQUEST_RESULT>
```

1. `seconds` specifies the number of seconds to wait for a response from a transaction request.

Child elements

There are no child elements.

Usage

You use the REQUEST_RESULT timeout to provide information to the user when a Web service transaction is taking longer than expected. It specifies the number of seconds to wait for the DVM to respond to the GAS, after which the GAS sends an HTTP 400 error page to the front-end to inform the user that the request has taken too long to fulfill.

The front-end cannot recover from a HTTP 400 error page, and any Web service client application must send a new request.

Important: The REQUEST_RESULT timeout has an initial setting of 60 seconds. When configuring this setting, you must ensure it is less than the Common Gateway Interface (CGI) timeout of the web server you are using. See FastCGI Installation and Web Server Configuration on page 98.

Usage example for Web service

```xml
<REQUEST_RESULT>60</REQUEST_RESULT>
```

In this usage example, the REQUEST_RESULT timeout is set to 60 seconds. It is referenced by the GWS proxy to release the 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 the following:

- SERVICE_APPLICATION_TIMEOUT_COMPONENT on page 422
- TIMEOUT (for a service) on page 432

Related tasks
Apache 2.4: mod_proxy_fcg on page 100
With Apache 2.4, mod_proxy_fcg is used instead of mod_fastcgi.

RESOURCE
This RESOURCE element defines general GAS settings that can be used in the definition of applications and components, and in external application configuration files.

Syntax

```xml
<RESOURCE Id=res-id [Source="INTERNAL"|"ENVIROM"]>res-data</RESOURCE>
```
1. *res-id* is the unique identifier of the resource.
2. The *Source* attribute specifies the location of the resource. This attribute is optional.

   The GAS locates the resource based on the value of the *Source* attribute, if specified.
   
   • If *Source* is *INTERNAL*, the resource is resolved to the location provided.
   • If *Source* is *ENVIRON*, the resource is the name of an environment variable, which represents the location of the resource.

   **Note:** If *Source* is not specified, the resource defaults to *INTERNAL*.
3. *res-data* is the value of the resource.

### Child elements

There are no child elements.

### Usage

You use this element to specify resources such as location of proxy program, DVM, etc., within individual application or service configurations. Resources are defined in the *RESOURCE_LIST* element of the GAS configuration file.

#### Source INTERNAL usage example

A resource defined with a reference to the internal location of the DVM or runtime program.

```
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">$(res.fgldir)/bin/fglrun.exe</RESOURCE>
```

#### Source ENVIRON usage example

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 411
- **UNIX** on page 436
- **WNT** on page 440

### Related concepts

- **RESOURCE (for an application)** on page 420
  This *RESOURCE* element defines a resource for an application.
- **RESOURCE (for a service)** on page 419
  This *RESOURCE* element defines a resource for a Web services application.

**RESOURCE (for a service)**

This *RESOURCE* element defines a resource for a Web services application.

For more information, see Resource.

### Parent elements

This element is a child of the **APPLICATION (for a service)** on page 373 element.

### Related concepts

- **RESOURCE (for an application)** on page 420
This RESOURCE element defines a resource for an application.

**RESOURCE (for an application)**
This RESOURCE element defines a resource for an application.

For more information on defining resources, see RESOURCE on page 418.

**Parent elements**
This element is a child of the **APPLICATION (for an application)** on page 370 element.

**Related concepts**
**RESOURCE (for a service)** on page 419
This RESOURCE element defines a resource for a Web services application.

**RESOURCE_LIST**
This element contains all RESOURCE elements, organized by operating system.

**Syntax**

```xml
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT>...</PLATFORM_INDEPENDENT>
  <WNT>...</WNT>
  <UNIX>...</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.

**Usage**
You use the RESOURCE_LIST element in the Genero Application Server configuration file to define RESOURCE elements, which can then be referenced in your application configuration files.

A resource is defined as **PLATFORM_INDEPENDENT** or **PLATFORM-DEPENDENT**, depending on the section (parent element) in which the resource is defined.

**Example usage**

```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 418.

**Parent elements**

This element is a child of the APPLICATION_SERVER on page 375 element.

**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.

**Syntax**

```xml
<ROOT_URL_PREFIX>url</ROOT_URL_PREFIX>
```

1. *url* specifies the URL to access the Web server behind the proxy.

**Child elements**

There are no child elements.

**Usage**

You use the ROOT_URL_PREFIX to override the URLs generated by the Web server and 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.

**Example usage**

```xml
<ROOT_URL_PREFIX>http://serverA:8080/gas</ROOT_URL_PREFIX>
```

- Where a reverse proxy server (for example, server A) is forwarding requests to the GAS on serverB.
- Where "gas" specifies the connector.uri part of the URI; typically this is the same as 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 INTERFACE_TO_CONNECTOR on page 400.

**Related concepts**

Setting up production environment on page 32

There are different options for configuring a GAS installation depending on your network. A Web server is required and security needs to be considered.

**SERVICE_APPLICATION_EXECUTION_COMPONENT**

The SERVICE_APPLICATION_EXECUTION_COMPONENT sets the runtime environment for Web services, defining execution rules and setting the execution environment.

**Syntax**

```xml
<SERVICE_APPLICATION_EXECUTION_COMPONENT Id=component-id>
  [...]
  <ENVIRONMENT_VARIABLE>...</ENVIRONMENT_VARIABLE> [...]
  <PATH>...</PATH> [...]
  <DVM>...</DVM> [...]
  <MODULE>...</MODULE> [...]
  <PARAMETERS>...</PARAMETERS> [...]
</SERVICE_APPLICATION_EXECUTION_COMPONENT>
```
1. *component-id* is the unique identifier for this set of execution definitions.

**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 `POOL` element.

**Usage**

You use this element to provide your Web service with a base set of execution parameters.

**Example usage**

```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 the `COMPONENT_LIST` on page 381 element.

**Related concepts**

EXECUTION (for a service) on page 391

This EXECUTION element sets the runtime environment for a Web service application by specifying parameters for its execution.

**SERVICE_APPLICATION_TIMEOUT_COMPONENT**

The `SERVICE_APPLICATION_TIMEOUT_COMPONENT` element creates a set of timeout values to be used when configuring a Web service.

**Syntax**

```xml
<SERVICE_APPLICATION_TIMEOUT_COMPONENT Id=component-id>
    ...<DVM_AVAILABLE>...<KEEP_ALIVE>...<REQUEST_RESULT>...</REQUEST_RESULT>
</SERVICE_APPLICATION_TIMEOUT_COMPONENT>
```

1. *component-id* is the unique identifier for this set of timeout definitions.
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` (for a service) on page 418 element.

Usage

You use this element to provide your Web service with a base set of time-out parameters. Timeouts provide a mechanism to the Genero Application Server (GAS) or Web service to react to time-based events; such as user inactivity, or the time it takes to start a new application. Each timeout element specifies the number of seconds to wait prior to having the GAS perform the task related to the timeout event.

You can use this element within the `COMPONENT_LIST` element of the GAS configuration file to define timeout components that can be referenced by `Id` when configuring an application; providing a set of timeout values for the application.

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. The settings defined by that component are inherited. The timeout elements defined within the application's configuration, override the inherited ones.

Parent elements

This element is a child of the `COMPONENT_LIST` on page 381 element.

Related concepts

Web application timeouts on page 54
Why are Web application timeouts necessary?

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.

Syntax

```xml
<SERVICE>
  [...]<HEADER>...</HEADER> [...]
</SERVICE>
```

Child elements

The `SERVICE` element may contain zero to many `HEADER` on page 398 elements.

Usage

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.

You use this element to specify a list of `HEADER` elements defining the communication carried on the HTTP protocol between Web services and the client.
Example usage

```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](#) element.

**SERVICE_LIST**
The `SERVICE_LIST` element provides a list of groups and Web services applications defined within the Genero Application Server configuration file.

Syntax

```xml
<SERVICE_LIST>
   [ <GROUP>... </GROUP> ]
   [ <APPLICATION>... </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.

Usage
This element is defined within the Genero Application Server (GAS) configuration file (`as.xcf`). For each Web service to be served by the GAS, you must provide the details for that application in either the GAS configuration file or in an external application configuration file (`xcf`).

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 the [APPLICATION_SERVER](#) element.
SESSION_DIRECTORY
The SESSION_DIRECTORY element specifies where to store the session files of applications and services started by the GAS.

Syntax

```xml
<SESSION_DIRECTORY>dir</SESSION_DIRECTORY>
```

1. `dir` specifies the session directory path.

Child elements
There are no child elements.

Usage
You use this element to specify where to store the session files of applications and services started by the GAS.

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 400.

SESSION_COOKIE
The SESSION_COOKIE element specifies that cookies the GAS uses to validate the session are created with the secure flag.

Syntax

```xml
<SESSION_COOKIE Secure=""FALSE"" | SameSite=""STRICT|LAX|NONE""/>
```

The SESSION_COOKIE element takes a mandatory Secure attribute and an optional SameSite attribute.

1. `Secure` valid values are TRUE or FALSE. When set to TRUE, the cookie is restricted to secured connections (HTTPS) only. The default is FALSE.
2. The SameSite attribute determines whether the cookie is restricted to a same-site context. Valid values are:
   * **Strict**: The cookie is sent only if the site for the cookie matches the site currently shown in the browser. The cookie is not sent if the user follows a link from another site.
   * **Lax**: The cookie can be sent with top-level navigations and with GET requests initiated by third-party websites.
   * **None**: The cookie is sent in all contexts. The Secure attribute must be set to TRUE and the request must be in HTTPS format.

   The default value is Lax.

Child elements
There are no child elements.
Usage

The main goal of cookies is to keep a state, using session variables, between two runs of an application by the same user. You use the `SESSION_COOKIE` element to specify that Genero session identifier cookies, including cookies created by Sticky Web services, are created with the secure flag and the `SameSite` attribute set to strict.

You may need to set the `Secure` and `SameSite` attributes. To avoid cross-site domain errors, you set attribute `SameSite=None` and ensure all requests are secured and in HTTPS.

Example secure session cookie

```
<INTERFACE_TO_CONNECTOR>
  ...
  <HTTP>
    <SESSION_COOKIE SameSite="None" Secure="TRUE" />
  ...
</HTTP>
</INTERFACE_TO_CONNECTOR>
```

Secure session cookie in web-xml

Note: If using a Java Enterprise Edition (J2EE) server, the secure flag needs to be configured in the `web.xml` file as shown in the example:

```
<web-app>
  ...
  <session-config>
    <cookie-config>
      <secure>true</secure>
    </cookie-config>
  </session-config>
</web-app>
```

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.

Syntax

```
<SHORT>description</SHORT>
```

1. `description` specifies the short description.

Child elements

There are no child elements.

Usage

You use this element to provide a brief description of your application.

Usage example

```
<DESCRIPTION>
  <SHORT>A short description of the app</SHORT>
  <LONG/>
</DESCRIPTION>
```
**Parent elements**

This element is a child of the **DESCRIPTION** on page 385 element.

**Related concepts**

**LONG** on page 404

The **LONG** element contains the long description to be associated with an application definition.

**SOCKET_FAMILY**

The **SOCKET_FAMILY** element specifies whether domain sockets or TCP sockets are to be used on Linux®/UNIX™ systems.

**Syntax**

```
<SOCKET_FAMILY> type </SOCKET_FAMILY>
```

1. **type** specifies the type of socket to use, TCP or UNIX.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify whether domain sockets or TCP sockets are to be used on Linux®/UNIX™ systems. By default, the Genero Application Server uses Linux®/UNIX™ domain sockets to communicate between the dispatcher and the proxies on Linux®/UNIX™ systems. On Windows®, the GAS uses TCP sockets to communicate between the dispatcher and proxies.

To use TCP sockets on Linux®/UNIX™, change the **SOCKET_FAMILY** element value to TCP.

**Warning**: Forcing TCP sockets on UNIX™ should only be done for debugging purposes, at the request of support.

On Windows®, the **SOCKET_FAMILY** element 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 the `INTERFACE_TO_CONNECTOR` on page 400 element.

**SOCKET_PATH**
The `SOCKET_PATH` element defines the directory where Linux®/UNIX™ domain sockets will be created and stored.

**Syntax**

```xml
<SOCKET_PATH> path </SOCKET_PATH>
```

1. *path* specifies the socket path.

**Child elements**
There are no child elements.

**Usage**
You use this element to define the directory where Linux®/UNIX™ domain sockets will be created and stored. By default, `SOCKET_PATH` 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>
  <UNIX>
    <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">UNIX</RESOURCE>
    <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">/tmp</RESOURCE>
  </UNIX>
  ...
</RESOURCE_LIST>
```

```xml
<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>
```

Parent elements
This element is a child of `INTERFACE_TO_CONNECTOR` on page 400.

**START**
The `START` element specifies the number of DVMs to start for Web services.

**Syntax**

```xml
<START> num </START>
```

1. *num* specifies the number of DVMs to be attached to the Web service at start.
**Child elements**

There are no child elements.

**Usage**

You use **START** to specify the number of DVMs to start for the Web service.

**Constraint**

**START** must be less than or equal to the number **MAXAVAILABLE** on page 404.

```
START <= MAX_AVAILABLE
```

**Parent elements**

This element is a child of the **POOL** on page 411 element.

**TCP_ADMIN_PORT**

The **TCP_ADMIN_PORT** element specifies the port the Genero Application Server uses for administration tasks.

This is an optional element of the **INTERFACE_TO_CONNECTOR** element.

**Syntax**

```
<TCP_ADMIN_PORT> port </TCP_ADMIN_PORT>
```

1. **port** specifies the port number for GAS administration.

**Child elements**

There are no child elements.

**Usage**

gasadmin commands such as listing or closing sessions are run on this port.

If the port is not set, the default is set at 6999. If you need to set the port, use this element to specify the value of the port the Genero Application Server is listening on.

**Usage example**

```
<TCP_ADMIN_PORT>6999</TCP_ADMIN_PORT>
```

**Parent elements**

This element is a child of **INTERFACE_TO_CONNECTOR** on page 400.

**Related concepts**

Configure multiple dispatchers on page 171

If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

**TCP_BASE_PORT**

The **TCP_BASE_PORT** element specifies the base value of the port the Genero Application Server is listening.

**Syntax**

```
<TCP_BASE_PORT> base </TCP_BASE_PORT>
```
1. `base` specifies the base port number for the GAS.

   The true port that the Genero Application Server is listening to is the port specified by `TCP_BASE_PORT + TCP_PORT_OFFSET`.

Child elements

There are no child elements.

Usage

You use this element to specify the base value of the port the Genero Application Server is listening.

Usage example

```
<TCP_BASE_PORT>6420</TCP_BASE_PORT>
```

Parent elements

This element is a child of `INTERFACE_TO_CONNECTOR` on page 400.

Related concepts

- [Configure multiple dispatchers](#) on page 171

  If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

**TCP_PORT_OFFSET**

The `TCP_PORT_OFFSET` element specifies the offset value of the port the GAS is listening.

Syntax

```
<TCP_PORT_OFFSET> offset </TCP_PORT_OFFSET>
```

1. `offset` specifies the offset from the `TCP_BASE_PORT` on page 429 number.

Child elements

There are no child elements.

Usage

You set this element to specify the offset from the base port.

The true port that the Genero Application Server is listening to is the port specified by `TCP_BASE_PORT + TCP_PORT_OFFSET`.

Usage example

```
<TCP_PORT_OFFSET>75</TCP_PORT_OFFSET>
```

Parent elements

This element is a child of `INTERFACE_TO_CONNECTOR` on page 400.

Related concepts

- [Configure multiple dispatchers](#) on page 171
If you need to configure multiple dispatchers, you must configure different ports, and directories for each dispatcher to ensure that dispatcher information does not get mixed up.

**TEMPORARY_DIRECTORY**
The `TEMPORARY_DIRECTORY` element specifies where to store files transferred between the front-end and the application server host.

**Syntax**

```
<TEMPORARY_DIRECTORY> dir </TEMPORARY_DIRECTORY>
```

1. `dir` specifies the temporary directory path.

**Child elements**

There are no child elements.

**Usage**

You use this element to specify where files transferred between the front-end and the application server host are stored.

**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 400.

**TIMEOUT (for an application)**
The `TIMEOUT` element set timeouts for process requests and the start of the DVM.

**Syntax**

```
<TIMEOUT [Using=component-id ]>
  [.../<USER_AGENT>...]</USER_AGENT>
  [.../<REQUEST_RESULT>...]</REQUEST_RESULT>
  [.../<DVM_AVAILABLE>...]</DVM_AVAILABLE>
</TIMEOUT>
```

1. The `component-id` references a component with a set of timeout definitions that are used.

**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 417 element.
3. Zero or one `DVM_AVAILABLE` element.

**Usage**

You use this element to set limits on the amount of time a DVM takes to start, and the amount of time it takes to process requests between the GAS and the client.

Timeouts are important for performance, so default values are set, and values can be configured.
Timeout settings may be defined by referencing in its Using attribute a predefined WEB_APPLICATION_TIMEOUT_COMPONENT to inherit timeout settings, and/or by setting individual timeout settings specific to the application.

When settings are inherited from a WEB_APPLICATION_TIMEOUT_COMPONENT, settings you define locally within the TIMEOUT element override the component settings.

**Usage example referencing a timeout component**

```
<TIMEOUT Using="cpn.wa.timeout.set1" />
```

**Usage example using local elements**

```
<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 the UA_OUTPUT on page 434 element.

**Related concepts**

- AUTO_LOGOUT_COMPONENT on page 377
  - The AUTO_LOGOUT_COMPONENT element creates a component, which defines a mechanism for triggering and handling auto-logout events.

- Web application timeouts on page 54
  - Why are Web application timeouts necessary?

- TIMEOUT (for a service) on page 432
  - This element sets timeouts for process requests and the start of the DVM.

**TIMEOUT (for a service)**

This element sets timeouts for process requests and the start of the DVM.

**Syntax**

```
<TIMEOUT [Using=component-id]>
  [ <DVM_AVAILABLE>...</DVM_AVAILABLE> ]
  [ <KEEP_ALIVE>...</KEEP_ALIVE> ]
  [ <REQUEST_RESULT>...</REQUEST_RESULT> ]
</TIMEOUT>
```

1. The component-id references a component with a set of timeout definitions that are used.

**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 418 element.

**Usage**

You use this element to set limits on the amount of time a DVM takes to start, and the amount of time it takes to process requests between the GAS and the client.

Timeouts are important for performance, so default values are set, and values can be configured.
Timeout settings may be defined by referencing in its Using attribute a predefined SERVICEAPPLICATION_TIMEOUT_COMPONENT to inherit timeout settings, and/or by setting individual timeout settings specific to the Web service application.

When settings are inherited from a SERVICEAPPLICATION_TIMEOUT_COMPONENT, settings you define locally within the TIMEOUT element override the component settings.

**Usage example referencing component**

```xml
<TIMEOUT Using="cpn.ws.timeout.set1" /> 
```

**Usage example using local elements**

```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 the APPLICATION (for a service) on page 373 element.

**Related concepts**

TIMEOUT (for an application) on page 431
This element set timeouts for process requests and the start of the DVM.

Web application timeouts on page 54
Why are Web application timeouts necessary?

TIMEOUT (for auto logout)
This TIMEOUT element sets a timeout to handle auto logout.

**Syntax**

```xml
<TIMEOUT>seconds</TIMEOUT>
```

1. **seconds** specifies the number of seconds the DVM waits before it triggers an auto-logout event.

**Child elements**

There are no child elements.

**Usage**

You use this element to configure a timeout for auto logout. When the DVM detects an application has no user activity, it waits for the specified length before an auto logout event is triggered. The user receives a log out message after the period of time has elapsed.

**Usage example**

```xml
<TIMEOUT>0</TIMEOUT>
```

A timeout duration set to zero seconds means the auto logout is ignored and the application keeps running. A correct configuration requires that the TIMEOUT is set.

**Parent elements**

This element is a child of the following elements:
• AUTO_LOGOUT on page 375
• AUTO_LOGOUT_COMPONENT on page 377

Related concepts
COMMAND (for auto logout) on page 381
This is an element of AUTO_LOGOUT that provides a mechanism for the Genero Application Server to override an application's auto logout.

UA_OUTPUT
The UA_OUTPUT element specifies the configuration parameters for an application delivered by the UA proxy.

Syntax

```xml
<UA_OUTPUT [Using=component-id ]>
  <PROXY>...</PROXY>
  <PUBLIC_IMAGEPATH>...</PUBLIC_IMAGEPATH>
  [<GBC>...</GBC>]
  [<GDC_SHORTCUT>...</GDC_SHORTCUT>]
  [<TIMEOUT>...</TIMEOUT>]
</UA_OUTPUT>
```

1. The `component-id` references a ua output component to inherit its parameters.

Child elements
The UA_OUTPUT element may contain the following child elements:

1. One PROXY (for an application) on page 413 element.
2. One PUBLIC_IMAGEPATH on page 414 element.
3. Zero or one GBC on page 394 element.
4. Zero or one GDC_SHORTCUT on page 396 element.
5. Zero or one TIMEOUT (for an application) on page 431 element.

Usage
You use this element to specify 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 Browser Client (GBC), Genero Mobile for Android™ and Genero Mobile for iOS. These applications are configured in part by the UA_OUTPUT element.

Usage example

```xml
<UA_OUTPUT>
  <PROXY>$(res.uaproxy.cmd)</PROXY>
  <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
  <GBC>gbc</GBC>
  <TIMEOUT Using="cpn.wa.timeout"/>
</UA_OUTPUT>
```

Parent elements
This element is a child of the APPLICATION (for an application) on page 370 element.
**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>]
   [<GBC>...</GBC>]
   [<GDC_SHORTCUT>...</GDC_SHORTCUT>]
</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 413 element.
2. One *PUBLIC_IMAGEPATH* on page 414 element.
3. Zero or one *TIMEOUT* (for an application) on page 431 element.
4. Zero or one *GBC* on page 394 element.
5. Zero or one *GDC_SHORTCUT* on page 396 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 GBC, timeouts, or path to public images, and so on.

This element is defined within the COMPONENENT_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"/>
   <GBC>${res.gbc}</GBC>
   <GDC_SHORTCUT>${res.path.as}/tpl/shortcut/gdc-http.gdc</GDC_SHORTCUT>
</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 `<GBC>mygbc</GBC>`. This specifies a specific GBC 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/3.20/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>
    <GBC>mygbc</GBC>
  </UA_OUTPUT>
</APPLICATION>
```

**Parent elements**

This element is a child of the [COMPONENT_LIST](#) element on page 381.

**UNIX**

This element contains a collection of UNIX-specific RESOURCE elements.

**Syntax**

```xml
<UNIX>
  [...]<RESOURCE>...</RESOURCE> [...]
</UNIX>
```

**Child elements**

A UNIX element may contain the following child elements:

- Zero to many RESOURCE elements.

**Usage**

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.

```xml
<UNIX>
  <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">UNIX</RESOURCE>
  ... 
</UNIX>
```
**Parent elements**
This element is a child of one of the following elements: RESOURCE_LIST on page 420

**USER_AGENT**
The USER_AGENT element sets a timeout to handle the user agent connection.

**Syntax**

```
<USER_AGENT>seconds</USER_AGENT>
```

1. `seconds` specifies the number of seconds to wait for a client request.

**Child elements**
There are no child elements.

**Usage**
You set this timeout to specify a timeout in number of seconds that the GAS is to wait for a client request before assuming that the front-end has died or that there has been a network failure.

Under normal operation, the front-end sends a GET request to the GAS immediately after receiving a response. The client-side front-end (CSF) will also send keep-alive requests (/ua/ka) to keep the application alive in the case of user inactivity.

If the client has not sent a request to the GAS before the USER_AGENT timeout expires, it is assumed that the front-end client has died and an instruction is sent to the application's DVM to 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 when a user closes the browser instead of properly exiting the application; the front-end client has no mechanism to indicate to the GAS that the user has closed the browser. With the USER_AGENT timeout, the GAS closes the socket to the DVM, which causes the DVM to shut down.

**Usage example**

```
<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 439
- TIMEOUT (for an application) on page 431

**WEB_APPLICATION_EXECUTION_COMPONENT**
The WEB_APPLICATION_EXECUTION_COMPONENT element sets the runtime environment for applications, defining execution rules and setting the execution environment.

**Syntax**

```
<WEB_APPLICATION_EXECUTION_COMPONENT Id=component-id
  AllowUrlParameters=TRUE|FALSE>
  [...]<ENVIRONMENT_VARIABLE>...</ENVIRONMENT_VARIABLE>[...]
  [...]<PATH>...</PATH>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```
Attributes for this element include:

1. `component-id` is the unique identifier for this set of execution definitions.
2. The `AllowUrlParameters` attribute defines whether parameters set in the application URL at the command line should be ignored ("FALSE", default value) or provided to the DVM ("TRUE").

**Child elements**

The `WEB_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` on page 440 element.

**Usage**

You use this element to provide your application with a base set of execution parameters.

A `WEB_APPLICATION_EXECUTION_COMPONENT` is referenced in an application by its unique identifier, set by the `Id` attribute.

**Example usage**

```xml
<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>
    <ENVIRONMENT_VARIABLE Id="DVM">$(res.dvm.wa)</ENVIRONMENT_VARIABLE>
    <WEB_COMPONENT_DIRECTORY>${connector.uri}/ua/i/$(application.path)/webcomponents</WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

**Parent elements**

This element is a child of the `COMPONENT_LIST` on page 381 element.
WEB_APPLICATION_TIMEOUT_COMPONENT
The WEB_APPLICATION_TIMEOUT_COMPONENT element defines a set of timeout values to be used when configuring a Web application.

Syntax

```xml
<WEB_APPLICATION_TIMEOUT_COMPONENT Id=component-id>
    [...]<USER_AGENT>...</USER_AGENT>
    [...]<REQUEST_RESULT>...</REQUEST_RESULT>
    [...]<DVM_AVAILABLE>...</DVM_AVAILABLE>
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

1. `component-id` is the unique identifier for this set of timeout definitions.

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.

Usage

You use this element to provide your Web application with a base set of timeout parameters. Timeout components provide instruction on how the GAS reacts to time-based events; such as user inactivity, or the time it takes to start a new application. Each timeout element specifies the number of seconds to wait prior to having the GAS perform the task related to the timeout event.

You can use this element within the COMPONENT_LIST element of the GAS configuration file, to define timeout components that can be referenced by Id when configuring applications; providing a set of timeout values for your applications.

Example usage

```xml
<WEB_APPLICATION_TIMEOUT_COMPONENT Id="cpn.wa.timeout.set1">
    <USERAGENT>300</USERAGENT>
    <REQUEST_RESULT>60</REQUEST_RESULT>
    <DVM_AVAILABLE>10</DVM_AVAILABLE>
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

In this example, the `Id` value - "cpn.wa.timeout.set1" - can be referenced when defining an application. The settings defined by that component are inherited. The timeout elements defined within the application's configuration, override the inherited ones.

Parent elements

This element is a child of the COMPONENT_LIST on page 381 element.

Related concepts

Web application timeouts on page 54
Why are Web application timeouts necessary?
AUTO_LOGOUT_COMPONENT on page 377
The `AUTO_LOGOUT_COMPONENT` element creates a component, which defines a mechanism for triggering and handling auto-logout events.

**WEB_COMPONENT_DIRECTORY**
The `WEB_COMPONENT_DIRECTORY` element specifies the path where Web components for an application are located.

Syntax

```xml
<WEB_COMPONENT_DIRECTORY>path-list [;...]</WEB_COMPONENT_DIRECTORY>
```

1. `path-list` specifies the paths to web component directories.
   
   It allows for multiple paths to be specified, the separator used between resource paths is a semi-colon, ";;".

**Child elements**

There are no child elements.

**Usage**

You use this element to define paths from where Web components are served. This element value added to the URL, builds the path used to find a Web component.

The `WEB_COMPONENT_DIRECTORY` configuration only applies to Genero Desktop Client (GDC) applications delivered via GAS, and Genero Browser Client (GBC).

**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.

**Parent elements**

This element is a child of one of the following elements:

- `EXECUTION` (for an application) on page 390
- `WEB_APPLICATION_EXECUTION_COMPONENT` on page 437

**Related concepts**

- `PUBLIC_IMAGEPATH` on page 414
  The `PUBLIC_IMAGEPATH` element defines the public resources directory used by applications.

**WNT**

This element contains a collection of Windows®-specific `RESOURCE` elements.

**Syntax**

```xml
<WNT>
    <![CDATA[<RESOURCE>...</RESOURCE>]]>
</WNT>
```
**Child elements**

A `WNT` element may contain the following child elements:

- Zero to many `RESOURCE` elements.

**Usage**

The `WNT` element contains a list of Windows® resources: those resources are only available on the Windows® operating systems.

```xml
<WNT>
  <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">TCP</RESOURCE>
  ...
</WNT>
```

**Parent elements**

This element is a child of the following: `RESOURCE_LIST` on page 420

---

### 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

---

### Copyrights and Trademarks

The Copyright and trademark page is shared amongst all Four Js documentation. Appearance in this page does not guarantee a mention within the current document.

**Copyright notice**

© Copyright Four Js Development Tools, Inc. Use, duplication or disclosure restricted.

**Trademarks**

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated.

IBM and WebSphere are trademarks or registered trademarks of International Business Machines Corp., in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information.

Linux is a registered trademark of Linus Torvalds.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
DEBIAN is a registered trademark owned by Software in the Public Interest, Inc. Debian trademark is a registered United States trademark of Software in the Public Interest, Inc., managed by the Debian project. Genero is not affiliated with Debian. Debian is a registered trademark owned by Software in the Public Interest, Inc.

Red Hat is a registered trademark of Red Hat, Inc. in the United States and other countries.

Red Hat Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Index

A
access application
  URI syntax 48
  web address URI 48
administration
gasadmin 348
appdata
directory prerequisites 44
res.appdata.path 42
application configuration
  abstract application 111
  configuration file 113
  configure environment 118
  creating application group 112
  default application group 112
  GDC application 117
  overview 111
  Web application 114
  Web service 116
application environment
  environment variable
    FGL_VMPROXY_START_URL 44
archive lifecycle 285

B
basics
  auto logout
    re-login prompt 54
  DVM 40
  proxy 39
  Single sign-on (SSO) 55
  Web application timeouts 54
  XML schema 109
BDL DeployGar program options 253
BDL DeployGbc program options 256
BDL gettoken program options 251

C
clean up undeployed archives 294
command reference
  fastCGI configuration
    wrapper.fcgi 98
  fglgar 23
  fglrung -d 189
  fglrung AccessProgram 159
  fglrung ImportIdP 154
  fglrung ImportOAuth 149
gasadmin 23
httpdispatch -E 120
netsh 63
reset Winsock 63
configuration check
  specifying XML schema definition 109
  validating configuration 108
  XML schema definition 109
configuration element
  ACCESS_CONTROL 367
  ADDRESS 368, 369
  ALLOW_FROM 370
  APPLICATION (for a service) 373
  APPLICATION (for app) 370
  APPLICATION (for HTTP) 372
  APPLICATION_LIST 374
  APPLICATION_SERVER 375
  AUTHCONTEXT 158
  AUTO_LOGOUT 375
  AUTO_LOGOUT_COMPONENT 377
  CACHE_CONTROL_MAX_AGE 378
  CATEGORIES_FILTER 379
  COMMAND auto logout 381
  COMPONENT_LIST 381
  CONFIGURATION 382
  DELEGATE 383
  DELEGATE_OPTIONS 384
  DESCRIPTION 385
  DOCUMENT_ROOT 386
  DVM 387
  DVM_AVAILABLE 386
  END_URL 388
  ENVIRONMENT_VARIABLE 388
  EXECUTION for app 390
  EXECUTION for service 391
  FORMAT 392
  GBC 394
  GBC_LOOKUP_PATH 395
  GDC_SHORTCUT 396
  GROUP for application 397
  GROUP for service 397
  HEADER 398
  HTTP 399
  IDFORMAT 157
  INTERFACE_TO_CONNECTOR 400
  INTERFACE_TO_DVM 401
  KEEP_ALIVE 402
  LISTEN 402
  LOG 403
  LONG 404
  MAX_AVAILABLE 405
  MAX_REQUESTS_PER_DVM 405
  MIN_AVAILABLE 406
  MODULE 407
  MONITOR 405
  OUTPUT 407
  PARAMETER 408
  PARAMETERS 409
  PATH 410
  PLATFORM_INDEPENDENT 411
  POOL 411
  PROMPT 412
  PROXY for app 413
  PROXY for service 414
  PUBLIC_IMAGEPATH 414
  RAW_DATA 415
delegate
configuring 127
configuration override 120
configuring multiple dispatchers 171
dispatcher
run GAS in container 300
Deployment App
WHAT IS THE Genero Deployment App 236
deployment directory 289
disable Genero archive 292
docker
commands 303
disable archive
with deployment app 291
with gasadmin 291
environment variable
set using shell script 119
examples
application and resource deployment 276
application configuration
abstract Web application 111
D

demonstrations
access from GBC 29
Genero BDL demos 29
Genero Studio demos 29
deploy
app 237, 238
Genero Deployment App 237, 239
Web service 239, 241
deploy apps
remote server 298
deploying
with deployment app 286
with gasadmin 288
deployment directory 289
disable Genero archive
with deployment app 291
with gasadmin 291
disable archive
E

disable archive
with deployment app 291
with gasadmin 291
disable archive
with deployment app 291
with gasadmin 291
environment variable
set using shell script 119
examples
application and resource deployment 276
application configuration
abstract Web application 111
access control for demos 367
configuring group directory 112
default application group 112
GBC configuration file 114
GDC configuration 117
monitor access 405
secure cookies 425
simple configuration file 21
sticky Web services 269
Web service 116
Web services POOL 35
with OpenID Connect SSO 144
with SAML SSO 155
archive with deployment triggers 296
archive with public resources 296
delegation
  examples of use 136
  optimization example 128
  REST service 130, 134
environment configuration
calling shell script 119
file serving URI 51
internationalization
  setting DVM locale in .xcf file 59
  XML encoding for ISO-8858-6 (Arabic) 58
  XML encoding for UTF-8 58
ISAPI dispatcher
  isapidispatch.ini configuration 96
  production environment setup 32
run application 296
Single sign-on (SSO)
  OpenID Connect
    authorization program 145
    retrieve user identifier 145
  SAML
    authorization program 159
    retrieve identity attributes 158
    retrieve user identifier 157
  simplesso 165
Single sign-on(SSO)
  SAML
    execute an application 160
URI examples
  accessing Web service, /ws/r 267
  getting WSDL 267

FastCGI
  configuration
    Nginx 103
    restarting dispatcher 108
    troubleshooting 105
  feature changes
    version list 322
FGL debugger
  run in Linux 189
  run in Windows 189
file format
  Apache error_log 106
cfas.xsd 109
cfextwa.xsd 109
cfextws.xsd 109
  ISAPI extension configuration file 95
  isapidispatch.ini
    as-directory property 96
    configuration-file property 96
  mod_fastcgi
    fastcgi.conf 98
  Nginx fastCGI configuration
    default 103
    fastcgi_params 103
  OpenID Connect
    FGLPROFILE file 148
    OIDC.log 149
  SAML.log 164
  Sun Web Server 7
    fastCGI configuration
      magnus.conf 104
      obj.conf 104
    wrapper.fcgi 105
XML schema definition file 109
XSD 109
xterm.sh 189
front-end
  Genero Browser Client (GBC) 16
  Genero Desktop Client (GDC) 16
  Genero Web Services (GWS) 16

G

GAS
  as.xcf 360
  configuration file 360
  reliability 31
  set up a production environment 32
  standalone web server 15
GAS architecture
  dispatcher 30
  DVM 30
  proxy 30
GAS components 38
GAS compression
  compress files with gasadmin 169
  configure in imt.cfg 169
GAS configuration
  error message 10038 63
  expanding resource configuration 109
  user access rights 61
  validate with GBC 61

F

fastCGI
  configuration
    Apache 2.4 with mod_proxy_fcgi 100
    Apache for remote GAS(mod_fastcgi) 99
    Apache with mod_fastcgi 98
    Lighttpd 103
    mod_proxy_fcgi
      RewriteRule 101
    Sun Web Server 7 104
    windows 105
  dispatcher 98
  mod_deflate compression 99
validate with GDC 62
classifying with gasadmin 109

GAS directories
application data directory 40
installation directory 40

GAS monitoring
accessing monitor page 176
classifying for invalid Web services 180
dispatcher logs 180
HTTP error code 400 180
HTTP requests 184
monitor proxies and DVMs 182
monitor proxy 181
viewing logs and statistics 177

GAS, what is it? 11

GBC 259
Genero archive
deploy 285
deploy and secure 287
Genero Browser Client
application path 262
configuring development environment 171
default client 263
deploy with deploymentApp 261
deploy with gasadmin 260
deploying with deployment portal 259
GBC_LOOKUP_PATH configuration 263
reset original client 266
setting default client 265
specify client 263
specify default customization 265
Genero Console App
applications 232
groups 231
services 234
tokens 236
users 230
Web services 233
what is the Genero Console App 230
Genero Identity Provider (Genero Identity Provider (GIP))
configure for distributed GAS 218
configure for external IdP 220
install Genero Identity Provider (GIP) 215
quick start 224
Genero Identity Provider (GIP)
deploy app from scripts 242
overview 204
set scopes from scripts 243
GIP access error
401 249
GRE configuration
distributed mode 174
local mode 173

I
IIS Web server
configuration on GAS 71
internationalization
best practices 56
charset configuration
DVM locale 59
charsets configuration
HTML5 57
encoding architecture 56
environment variable
LANG 59
LC_ALL 59
OS locale 58
system character encoding 58
XML encoding in xcf file 58
ISAPI See IIS Web server
ISAPI dispatcher
activate IIS on Windows Server 2012 71
application root directory 96
configuring IIS 10.x 71
configuring IIS 6.x 94
configuring IIS 7.x 84
configuring IIS 8.x 71
configuring IIS after installation 95
installing ISAPI 71
ISAPI directory 96
ISAPI extension 71
ISAPI extension configuration 96
restarting 97
troubleshooting installation 97

L
license
checking licenses in use
fglWrt -u 67
freeing up licenses 67
licensing tips 69
limiting Web service users 70
migrating from GDC to GBC 70
released by timeout 69
troubleshooting 69
your requirements
GBC 67
GDC 67
licensing examples
base example 65
multi user 66
summary case 67
using run command 66
using run without waiting 66
list deployed archives 289
load balancing
configuration examples 197
configure Apache 2.2.x 203
configure Apache mod_proxy 202
configure IIS 7.x 200
configure ISAPI_Rewrite filter 199
multiple instances of GAS 191
session-bound requests 191
session-bound, cookie-based requests 193
session-bound, path-based requests 193
sessionless requests 192
localization, See internationalization
log files
accessing application logs 185
GIP logs 247
managing logs 185
sensitive data 187

P

packaging applications
  create archive directory 273
  create archive with MANIFEST 275
  create configuration files 274
  create manifest file 275
  organize resources 273
  resource deployment
    public and private images 276
    Web components for GBC 279
performance tuning
  HTTP Keep-Alive 190
  SPDY 190
proxy
  gwsproxy 347
  types and function 39
  uaproxy 347

Q

quick start
  archiving an application 23
  deploying a Web service 27
  deploying an application 23
  docker 301
  enabling an application 23
  exploring resources 16
  external application configuration 21
  GAS directories 16
  run calculator client app 25
  run demo calculator service 25
  running a client application 26
  running a Web service 26
  running application on GBC 22
  running application on GDC 22
  running demo application on GBC 17
  running demo application on GDC 17
  Web service configuration 26
quick start SSO
  OpenID Connect, set up on GAS 142
  SAML, set up on GAS 152

R

reference
  configuration elements
    used in GAS 361
    used in GBC application 364
    used in GWS application 365
    used in web applications 360
  Genero identity provider 251
  GIP 251
resource paths
  $(res.appdata.path)/public 277
  $(res.path.tmp) 277
  $(res.public.resources) 277
runtime
  configuration override
    appdata 120

GBC 120

S

secure
  application 237, 238
  Genero Deployment App 237, 239
  Web service 239, 241
SharedFileApp
  what is the SharedFile App 245
Single sign-on (SSO)
  autologout prompt 165
  database connections 166
  how to implement 165
  Identity Providers (IdP) 139
  ImportOAuth program 149
OpenID Connect
  Add to Web application 144
  authorize re-log in 147
  authorize user 145
  configure for GAS 143
  configure Google identity 143
  FGLPROFILE file 148
  IDP parameter
    CLIENT_PUBLIC_ID parameter 144
    CLIENT_SECRET_ID parameter 144
  IDP redirect with HTML form 145
  OIDC log file 149
  Retrieve user identifier 145
  set up on GAS 142
SAML
  Add SSO to Web application 155
  authenticate user 158
  authorize user 159
  authorize user re-log 160
  configure GAS 154
  database 163
  define the ID format 157
  digital signatures 154
  federation of identities 157
  FGLPROFILE file 164
  FGLPROFILE file entries 161
  IDP parameter 156
  importIdP program 154
  log file 164
  retrieve identity attributes 158
  retrieve user identifier 157
  Set up on GAS 152
  specify Identity Provider 156
  support for OAuth2 149
  workflow 141
Single sign-on(SSO)
  facebook SSO 151
  instagram SSO 151
OpenID Connect
  SSO logout 147
SAML
  authenticate user 160
  SSO logout 161
standalone GAS
  httpdispatch 32
T

tools and commands
command-line utilities 343
dispatchers 343
fgldb 189
proxies 343
troubleshooting
application does not start 63, 122
application issues 121
application rendering issues 123
application session id 123
FastCGI
  all applications down 106
  application timeout on Apache
  mod_reqtimeout 105
  SYSTEM ERROR 105
  invalid installation directory 106
  mod_proxy_fcgi URL decode 106
  permission denied 105
  SIGUSR1 106
  using multiple dispatchers 107
GAS configuration
  cannot find localhost 63
  Windows proxy errors 63
GIP access errors 249
GIP errors in logs 247
GIP HTTPS errors 250
GIP issues 246
GIP oidc errors 249
HTTP errors in logs 121
httpdispatch
  address already in use 344
  restarting httpdispatch 344
IDP errors 248
ISAPI dispatcher installation 97
license
  machine real address 69
  session ownership 64
Web services
  HTTP error 503 268
  invalid configuration 268

U

undeploy archives
  with deployment App 293
  with gasadmin 293
upgrade a Web service
  manually 296
upgrade an archive
  upgrade an application 295
upgrade guide
  Migrating templates and snippets 343
    Version 2.00 342
    Version 2.20 337
    Version 2.11 337
    Version 2.22 336
    Version 2.30 334
    Version 2.40 332
    Version 2.41 332
    Version 2.50 331

X

xcf file 113