Genero Application Server for Java User Guide
## Contents

**Genero Application Server for Java User Guide** .................................................. 5

**What's New in 1.00** ................................................................................................... 5

**JGAS overview** ........................................................................................................ 6
  Install and configure for JGAS..................................................................................... 7
  JGAS package content.................................................................................................. 7
  Override the WebContent.............................................................................................. 8
  Override the GBC........................................................................................................ 8
  Deploy and run standalone.......................................................................................... 9

**Quick Start Guide** ................................................................................................... 9
  Compile applications...................................................................................................... 9
  Build Genero archive.................................................................................................. 11
  Build war file............................................................................................................. 12
  Run an application in JGAS........................................................................................ 13
  Tutorial: Run fgl demos through JGAS...................................................................... 13
  Tutorial: Run ggc demos through JGAS...................................................................... 15

**Configuring applications on JGAS** ....................................................................... 15
  Application configuration file...................................................................................... 16
  Application and service templates............................................................................. 17
  How to implement delegation..................................................................................... 17
    How delegation works.............................................................................................. 17
    Configure delegation for application or service.................................................... 20
    From the user agent to the REST service............................................................... 21
    From the REST service to the proxy...................................................................... 23
    REST service example............................................................................................ 24
    Delegation use cases............................................................................................... 26
  How to implement Single sign-on (SSO).................................................................. 27
    OpenID Connect SSO............................................................................................... 27
    SAML SSO............................................................................................................. 33

**Application Web address** ....................................................................................... 43

**Serving static files** .................................................................................................. 44

**JGAS User Interface** ............................................................................................... 44
  Home page................................................................................................................ 44
  JGAS global resources page....................................................................................... 46
  JGAS application resources page.............................................................................. 48
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
</table>

**JGAS logging**

JGAS log file.................................................................50
Override category log resource........................................50
Override log resource at command line............................52

**Auto logout**

Override auto logout resource........................................52

**Upgrading**

New Features of the JGAS..................................................54
JGAS 1.00 new features......................................................54

**JGAS Configuration Reference**

JGAS configuration file..................................................55
JGAS configuration file hierarchy........................................56
Syntax diagrams..............................................................57
Configuration file elements..............................................58
ACCESS_CONTROL..........................................................58
ALLOW_FROM........................................................................60
APPLICATION (for an application).................................60
APPLICATION (for HTTP)..................................................61
APPLICATION (for JGAS)..................................................62
APPLICATION (for a service)............................................63
AUTO_LOGOUT (for JGAS).................................................64
CACHE_CONTROL_MAX_AGE.............................................65
CATEGORIES (under LOG for JGAS).................................66
CLASS...............................................................................67
COMMAND (for auto logout)..............................................68
COMMAND (under EXECUTION for JGAS).........................69
CONFIGURATION (for JGAS).............................................69
CONSOLE (under LOG for JGAS).......................................71
DAILYFILE (under LOG for JGAS).................................72
DELEGATE.........................................................................72
DESCRIPTION.....................................................................73
DVM_Available..................................................................74
DVM.................................................................................74
ENVIRONMENT_VARIABLE.............................................75
EXECUTION (for a service)................................................76
EXECUTION (under APPLICATION for JGAS)....................77
FORMAT (under LOG for JGAS)........................................78
GBC...............................................................................80
GBC_LOOKUP_PATH..........................................................81
GDC_SHORTCUT...............................................................82
HEADER............................................................................82
HTTP...............................................................................83
INTERFACE (for JGAS)......................................................84
KEEP_ALIVE......................................................................85
LONG..............................................................................86
LOG (for JGAS)..............................................................86
MAX_AVAILABLE............................................................87
MAX_REQUESTS_PER_DVM.............................................88
Genero Application Server for Java User Guide

Manual organization at a glance.

<table>
<thead>
<tr>
<th>JGAS overview on page 6</th>
<th>JGAS Quick Start on page 9</th>
<th>JGAS User Interface on page 44</th>
<th>JGAS logging on page 49</th>
</tr>
</thead>
<tbody>
<tr>
<td>• JGAS 1.00 new features on page 5</td>
<td>• Compile applications on page 9</td>
<td>• JGAS home page on page 44</td>
<td>• JGAS log file on page 50</td>
</tr>
<tr>
<td>• JGAS package content on page 7</td>
<td>• Build Genero archive on page 11</td>
<td>• JGAS global resources page on page 46</td>
<td>• Override category log resource on page 50</td>
</tr>
<tr>
<td>• Run an application in JGAS on page 13</td>
<td></td>
<td>• JGAS application resources page on page 48</td>
<td>• Override log resource at command line on page 52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JGAS auto logout on page 52</th>
<th>JGAS Configuration Reference on page 55</th>
<th>How to implement Single sign-on (SSO) on page 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Override auto logout resource on page 52</td>
<td>• JGAS configuration file on page 55</td>
<td>• Quick start: Set up OpenID Connect in JGAS on page 27</td>
</tr>
<tr>
<td></td>
<td>• JGAS configuration file hierarchy on page 56</td>
<td>• Quick start: Set up SAML in the JGAS on page 33</td>
</tr>
<tr>
<td></td>
<td>• Syntax diagrams on page 57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Configuration file elements on page 58</td>
<td></td>
</tr>
</tbody>
</table>

JGAS 1.00 new features

A summary of new features and changes in functionality introduced with Genero Application Server for Java (JGAS) 1.00.

Table 1: Engine and Architecture, Version 1.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>A GAS written in Java (JGAS) is provided. It can be packaged in a Java Web Archive (war) file with applications for development environment testing as standalone without the need for a separate GAS installation or a Java Enterprise Edition server and you can access your applications and services via a browser. TheCACHE_CONTROL_MAX_AGE element allows you to specify the duration files sent by the JGAS are held in front-end cache.</td>
<td>No additional reference.</td>
</tr>
<tr>
<td>The cache.control.max.age resource defines the setting for the front-end cache in CACHE_CONTROL_MAX_AGE.</td>
<td>See CACHE_CONTROL_MAX_AGE on page 65</td>
</tr>
</tbody>
</table>


Table 2: Single Sign-On (SSO) and delegation

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenID Connect and SAML are Single sign-on (SSO) protocols supported by</td>
<td>See How to implement Single sign-on (SSO) on</td>
</tr>
<tr>
<td>the JGAS. They are based on Genero delegation REST services delivered</td>
<td>page 27.</td>
</tr>
<tr>
<td>in the FGLGWS package under $FGLDIR/web_utilities/services/.</td>
<td></td>
</tr>
<tr>
<td>The JGAS is able to delegate the start of a Genero Web service (/ws/r),</td>
<td>See How to implement delegation on page 17.</td>
</tr>
<tr>
<td>or GBC/GDC (/ua/r) application to another REST service in order to</td>
<td></td>
</tr>
<tr>
<td>perform some controls (such as authentication, authorization, or</td>
<td></td>
</tr>
<tr>
<td>monitoring) in a single and centralized Genero program.</td>
<td></td>
</tr>
</tbody>
</table>

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

**JGAS overview**

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.

This implementation performs the same functions as the GAS. Where it differs noticeably from the standard GAS is in its architecture and in terms of how it implements configuration and logging. In configuration it offers more flexibility with element order and schema validation, and it enhances the logging mechanism. See JGAS configuration file on page 55.

**Using the JGAS**

The JGAS is packaged with the FGLGWS product, and the `fglgar` tool, included in the FGLGWS installation, provides you with access to the JGAS and a means to package and deploy applications.

**Note:** For more information on using `fglgar`, see the `fglgar` topic in the Genero Business Development Language User Guide.

The `fglgar war` command provides the ability to package a Genero archive (gar) file with your applications, and to include the JGAS implementation, the Genero Browser Client, and any additional files in a Java Web Archive (war) file. The war can be deployed in any existing Java Enterprise Edition container such as Apache Tomcat®, Jetty, or Glassfish and the applications can be run in a browser.

**JGAS Architecture**

The JGAS uses Java and is designed to run on the J2EE servlet. It differs from the standard GAS in the way it uses Web server resources such as sockets and memory in the handling of HTTP communication between the DVM and the front-end.

In the standard GAS when an application is started (for example by launching it from the browser or from the Genero Desktop Client), the communication passes through the following three stages of processing:

1. The Web server receives an HTTP request that is forwarded to the GAS dispatcher.
2. The GAS dispatcher forwards it to the proxy using sockets.
3. The proxy sends the data to the DVM and handles any child VMs that may be started.

In the JGAS architecture those processes are similar except that proxies are not used between the DVM and the dispatcher. HTTP requests are processed internally and one socket per DVM is all that is required to maintain the communication.
JGAS offers the following advantages:

- Less resources are consumed overall as less system processing is required.
- There are no proxy processes running, and therefore no sockets are required between the proxy and the dispatcher.
- Asynchronous management of requests allows the J2EE container to recycle worker threads instead of keeping them locked waiting for a response.

**JGAS as standalone**

The JGAS can not run standalone but a standalone feature similar to that of the standard GAS standalone dispatcher httpdispatcher, is provided by the jgas-standalone.jar included in the war. This jar provides the function of a J2EE server when the war file is run with the fglgar run command, allowing you to deploy and run applications without having to install and administer a complete J2EE server.

This feature offers an enhancement for testing applications that allows you to access the JGAS configuration and deployed applications via a user interface. See Deploy and run standalone on page 9.

**Install and configure for JGAS**

The JGAS is embedded in the FGLGWS package. No additional installation is required.

Its installation path is FGLDIR/web_utilities/jgas. The contents of the installation are described in JGAS package content on page 7.

You need a Java Runtime Environment (JRE) on production sites, or if you want to test a WAR file using fglgar run.

**JGAS package content**

When working with JGAS, it is helpful to know the files contained in the installation, and their function.

**JGAS installation directory structure**

Your JGAS installation directory, FGLDIR/web_utilities/jgas, contains a directory structure with files as shown:

```plaintext
### PRODUCTINFO
### VERSION
### WebContent
#   ### index.jsp
#   ### META-INF
#   #   ### MANIFEST.MF
#   #   ### services
#   #   #   ### javax.servlet.ServletContainerInitializer
#   ### WEB-INF
#   #   ### as.xcf
#   #   ### lib
#   #   #   ### jgas.jar
#   #   ### mimetypes.properties
#   #   ### web.xml
### jgas-standalone.jar
```

The files listed in Table 3: Main JGAS installation files and directories on page 8 are the most important ones.

**Important:** All installation files are subject to change when an upgrade occurs. It is therefore not recommended to modify any files in the installation directory to avoid having to reapply your modifications each time the product is upgraded.
Table 3: Main JGAS installation files and directories

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCTINFO</td>
<td>This file contains JGAS version and copyright information.</td>
</tr>
<tr>
<td>VERSION</td>
<td>This is the JGAS version number. For example, 1.00.00-1479199935. <strong>Note:</strong> The JGAS version differs from the FGLGWS package version.</td>
</tr>
<tr>
<td>WebContent</td>
<td>The WebContent directory is provided as the template to generate a war file.</td>
</tr>
<tr>
<td>as.xcf</td>
<td>This is the main JGAS configuration file, it provides defaults for embedded applications. See JGAS configuration file on page 55</td>
</tr>
<tr>
<td>mimetypes.properties</td>
<td>This file contains a list of the MIME types that are mapped to file extensions.</td>
</tr>
<tr>
<td>jgas.jar</td>
<td>This jar file is the actual JGAS implementation.</td>
</tr>
<tr>
<td>jgas-standalone.jar</td>
<td>This jar file is the standalone version of JGAS. It contains an embedded Jetty (Java Enterprise Edition container) that can be used for development without requiring an actual J2EE deployment.</td>
</tr>
<tr>
<td>web.xml</td>
<td>The web.xml file is the Web deployment descriptor. It may be user modified to add servlet declarations, etc.</td>
</tr>
</tbody>
</table>

### Override the WebContent

JGAS installation files are generally subject to change with upgrades. If you need to keep a consistent version of the WebContent directory, then you can override it when deploying a war file. The override option is explained.

It is not recommended to modify the WebContent directory provided in the installation but instead you can override it with one provided by you when building a war file.

This allows you to keep the WebContent directory as part of your project and avoid modifying the original installation or risk losing your modifications after a JGAS product upgrade.

The following command to build a war shows an example using the --web-content option to specify the override:

```
fglgar war --input-gar myApp.gar --output myApp.war --web-content <myWebContent_directory>
```

When building the war file, the WebContent directory provided in the installation is first copied. Then it is overwritten with the WebContent directory provided on the command line.

### Override the GBC

If you wish to use a customized Genero Browser Client (GBC) instead of the one installed with the FGLGWS package, then you can override it when deploying a war file. The override option is explained.

The Genero Browser Client (GBC) packaged with the FGLGWS product is found in the $FGLDIR/web_utilities/gbc path. When building the war file, this GBC client is embedded in the JGAS implementation.
While it is not recommended to modify the contents of the gbc directory provided in the installation, instead you can override it with one provided by you when building a war file.

This means that you can have your customized GBC as part of your project and avoid modifying the original installation, or risk losing your modifications after a product upgrade.

The following command to build a war shows an example using the --gbc option to specify the override:

```
fglgar war --input-gar myApp.gar --output myApp.war --gbc c:\dev\gbc_customized
```

## Deploy and run standalone

Applications or services packaged in a Genero archive and deployed in a war file can be run directly in standalone mode.

### Deploy and run as standalone

A JGAS implementation includes the jgas-standalone.jar. This jar file features a small Java Enterprise Edition (J2EE) server. It is included in the war file allowing you to deploy and run applications without having to install and administer a complete J2EE server. Just an installation of the Java Runtime Environment (JRE) is required on the deployed system.

A Java Runtime Environment (JRE) is required to run the application war file with fglgar run.

To deploy a war and run applications, the following fglgar command is run:

```
fglgar run --war myApp.war
```

With this feature, the JGAS provides the same kind of function as the standalone dispatcher (httpdispatch) of the standard GAS for the development and testing of applications.

### Related tasks

- Run an application in JGAS on page 13
- When developing application, the standalone feature of the JGAS allows you to test them.

## JGAS Quick Start

To give you an idea of what the JGAS does and help you to get started, this section guides you with examples for building an archive with applications, packaging them in a Genero Archive and war file, and running applications on the standalone JGAS.

### Compile applications

Compile applications and services.

**Before you begin:**

Run the script file `envcomp` in the Genero BDL installation directory to make sure that your FGLDIR and PATH are set correctly to run compiler and runtime system tools.

1. With a text editor, create an application source file called `helloapp.4gl`.
   
   In it copy the code shown in the example and save:
   
   ```
   IMPORT FGL fgldialog
   ```
MAIN
    OPEN FORM f FROM "helloapp"
    DISPLAY FORM f

    MENU "Exit"
    COMMAND "Hello"
    CALL fgl_winmessage("Hello", "Hello World!", "exclamation")
    COMMAND "exit"
    EXIT MENU
END MENU
END MAIN

2. To create the application form, create a file called helloapp.per
   Copy the code shown in the example and save:

   LAYOUT(TEXT="Hello World")
   GRID
   { Hello World! }
   END

3. To create a service application, create a file called helloservice.4gl
   Copy the code shown in the example and save:

   IMPORT XML
   IMPORT COM

   &define L(s) DISPLAY SFMT("(%1:%2) %3", __FILE__, __LINE__, s)
   &define LF(s) L(SFMT s)

   # Query string parameters
   TYPE QSParams DYNAMIC ARRAY OF RECORD
   name STRING,
   value STRING
   END RECORD

   MAIN
   DEFINE url,
   qs,
   requestUrl STRING
   DEFINE request com.HTTPServiceRequest
   CALL com.WebServiceEngine.Start()

   WHILE TRUE
     TRY
       #Retrieve the HTTPServiceRequest object to handle the HTTP request
       in various FUNCTIONs
       LET request = com.WebServiceEngine.GetHttpServiceRequest(-1)

       #Separate the incoming URL from query string
       LET requestUrl = request.readFormEncodedRequest(false)
       CALL splitUrl(requestUrl) RETURNING url, qs
       CALL processRequest(request, requestUrl)
     CATCH
       LF(("Exception %1 - %2", status, SQLCA.SQLERRM))
       L("Got an exception - Exiting")
       EXIT WHILE
     END TRY
   END WHILE
END MAIN
FUNCTION splitUrl(url)
  DEFINE url STRING
  DEFINE i INTEGER
  IF url IS NULL THEN
    RETURN NULL, NULL
  END IF
  LET i = url.indexOf("?", 1)
  IF i >= 1 THEN
    RETURN url.substring(1, i - 1), url.substring(i + 1, url.getLength())
  END IF
  RETURN url, NULL
END FUNCTION

FUNCTION processRequest(request, requestUrl)
  DEFINE request com.HTTPServiceRequest,
  requestUrl STRING
  LF("(processRequest: %1", requestUrl))
  CALL request.sendTextResponse(200, "OK", "Hello World!")
END FUNCTION

4. Compile the application and service source files (4gl and per).
   Run the commands shown at the command line:

   fglcomp -r -M helloapp.4gl
   fglform -M helloapp.per
   fglcomp -r -M helloservice.4gl

   The application and service 42m and 42f files are created.

Build Genero archive

Build a Genero Archive for deployment of applications and services.

Before you begin:

Run the script file envcomp in the Genero BDL installation directory to make sure that your FGLDIR and PATH are set correctly to run compiler and runtime system tools.

1. At the command line of the directory where you build the archive, create two subdirectories, one called app and one called service.

   Note: This example shows a valid organization of files. There is no requirement for this other than that all your application files, along with your application configuration (xcf), and MANIFEST file, must sit under one directory.

   mkdir app
   mkdir service

2. Copy or move your compiled application files (42m and 42f) to these subdirectories.

   The archive directory now contains the compiled application files in the app directory, and the compiled service files in the service directory.

3. With a text editor, create the MANIFEST file.
In it list the application and service xcf file located in the archive directory, as shown in the example:

```
<MANIFEST>
  <APPLICATION xcf='app/helloapp.xcf'/>
  <SERVICE xcf='service/helloservice.xcf'/>
</MANIFEST>
```

4. With a text editor, create the application and service configuration files (xcf).
   This example shows a sample application configuration file, helloapp.xcf. Save it to the app directory.
   ```xml
   <?xml version="1.0"?>
   <APPLICATION Parent="defaultgwc">
     <RESOURCE Id="log.console.categories">ERROR WARNING INFO SESSION VM</RESOURCE>
     <EXECUTION>
       <PATH>$(res.deployment.path)/app</PATH>
       <MODULE>helloapp.42m</MODULE>
       <ACCESS_CONTROL>
         <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
       </ACCESS_CONTROL>
     </EXECUTION>
     <AUTO_LOGOUT>
       <TIMEOUT>20</TIMEOUT>
     </AUTO_LOGOUT>
   </APPLICATION>
   ```

   The example shows a sample application service configuration file, helloservice.xcf. Save it to the service directory:
   ```xml
   <?xml version="1.0"?>
   <APPLICATION Parent="ws.default">
     <EXECUTION>
       <PATH>$(res.deployment.path)/service</PATH>
       <MODULE>helloservice.42m</MODULE>
     </EXECUTION>
   </APPLICATION>
   ```

5. Build the Genero Archive file (gar) using the fglgar tool.
   At the command line of the archive directory type the command:
   ```
   fglgar gar --output ../hello.gar
   ```
   The hello.gar is created in the parent directory of the archive directory. The gar file is ready to be deployed in a war file. See Build war file on page 12.

**Build war file**

Build a war file for deployment of the JGAS and applications. The war file embeds the Genero archive with the applications and web services and the JGAS implementation to serve these applications.

**Before you begin:**

Run the script file envcomp in the Genero BDL installation directory to make sure that your FGLDIR and PATH are set correctly to run compiler and runtime system tools.

You must have created a gar file with the applications and web services you wish to deploy. See Build Genero archive on page 11.

Run the fglgar war command to package the Genero Archive and JGAS in a war archive.

**Note:** Genero Browser Client (GBC) customization project, and any additional files can also be included in the war.
At the command line of the directory with the gar file, type the command:

```
fglgar war --input-gar hello.gar --output hello.war
```

The hello.war file is created. The war is ready to be deployed in any existing Java Enterprise Edition container such as Apache Tomcat®, Jetty, or Glassfish or as the standalone JGAS. See Run an application in JGAS on page 13.

## Run an application in JGAS

When developing application, the standalone feature of the JGAS allows you to test them.

### Before you begin:

Run the script file `envcomp` in the Genero BDL installation directory to make sure that your FGLDIR and PATH are set correctly to run compiler and runtime system tools.

A Java Runtime Environment (JRE) is required to run the application war file with `fglgar run`.

1. The `fglgar run` command is used to start the JGAS in standalone mode. At the command line type the command:

```
fglgar run --war hello.war
```

The output at the command line indicates that the JGAS is started as shown.

![Figure 1: JGAS started in standalone mode](image)

2. By default JGAS listens on port 8080. The name of the war file, `hello` in the example, provides the path to locate the deployed war.

   Open the browser at the following URL:

   `http://localhost:8080/hello`

   The JGAS user interface home page opens showing the applications and services deployed in the war file. See JGAS home page on page 44.

### Related concepts

- **Application Web address** on page 43
  
  To access an application deployed in a war file, there are different options for entering the URL in the browser's address bar.

## Tutorial: Run fgl demos through JGAS

Demo applications are packaged with the FGLGWS product. You can create a war file that allows you to run these applications through the JGAS.

### Before you begin:
• Run the script file `envcomp` in the Genero BDL installation directory to make sure that your FGLDIR and PATH are set correctly to run compiler and runtime system tools.

• A Java Runtime Environment (JRE) is required to run the application `war` file with `fglgar run`.

1. Create a new directory, for example, called "demos".

   mkdir demos

2. With a text editor create a configuration file named `gwc-demo.xcf`. Add the following configuration and save the file to the "demos" directory:

   ```xml
   <APPLICATION version="1.0" encoding="UTF-8">
     <EXECUTION>
       <PATH>$(res.fgldir)/demo</PATH>
       <MODULE>demo</MODULE>
     </EXECUTION>
   </APPLICATION>
   ```

3. Create a second configuration file named `simple-demo.xcf` with the following and save to the "demos" directory:

   ```xml
   <APPLICATION version="1.0" encoding="UTF-8">
     <EXECUTION>
       <PATH>$(res.fgldir)/demo/Widgets</PATH>
       <MODULE>Edit.42m</MODULE>
     </EXECUTION>
   </APPLICATION>
   ```

4. Create the `gar` file by running the following command at the command line of the demos directory:

   ```bash
   fglgar gar --output demos.gar --application gwc-demo.xcf --application simple-demo.xcf
   ```

   The `demos.gar` file is created.

5. Create the `war` file by running the following command at the command line of the demos directory:

   ```bash
   fglgar war --input-gar demos.gar --output demos.war
   ```

   The `demos.war` files is created.

6. Run the demos application on standalone JGAS

   a) Use the following command at the command line of the demos directory:

   ```bash
   fglgar run --war demos.war
   ```

   b) You can now run the applications via your browser at the following URLs:

   ```text
   http://localhost:8080/demos/ua/r/gwc-demo
   http://localhost:8080/demos/ua/r/simple-demo
   ```

7. Or you can run the demos application on your J2EE server.

   a) Deploy the `war` file using your J2EE server tool or deployment portal interface.

   b) Run the applications from the J2EE server.
The Genero Ghost Client (GGC) demo applications are packaged with the FGLGWS product. You can create a war file that allows you to run these application tests through the JGAS.

**Before you begin:**

- The **demos.war** file is created as described in Tutorial: Run fgl demos through JGAS on page 13.
- Run the script file **envcomp** in the Genero BDL installation directory to make sure that your FGLDIR and PATH are set correctly to run compiler and runtime system tools.
- A Java Runtime Environment (JRE) is required to run the application war file with **fglgar run**.

1. Compile the GGC demo application source files (4gl) in the **FGLDIR\testing_utilities\ggc\samples** directory.
   
   Type the following commands at the command line.

   ```
   fglcomp -r -M Console.4gl
   fglcomp -r -M SimpleTest.4gl
   ```

   The 42m files are created.

2. Run the **demos** applications using the following command:

   ```
   fglgar run --war demos.war
   ```

3. Run the **SimpleTest** to test the simple-demo application.

   Type the command shown at the command line of the **FGLDIR\testing_utilities\ggc\samples** directory

   ```
   fglrun SimpleTest http://localhost:8080/ggc_demos/ua/r/simple-demo
   ```

   You should see output from the test as shown.

   ![Figure 2: Running GGC test demos through JGAS](image)

For more information on Genero Ghost Client, see the *Genero Ghost Client User Guide*.

### Configuring applications on JGAS

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

The topics in this section provide details of configuring applications for delivery on the JGAS.
Application configuration file

An application configuration file defines an application environment and provides the JGAS with the information needed to run an application.

Configuration file

Tip: Application configuration files defined for standard GAS are compatible with JGAS.

Typically, the name of the configuration file matches the name of the application and has an .xcf suffix. For example, if the application name was "app1", create a configuration file named app1.xcf.

The configuration starts with the APPLICATION (for an application) element. Within this element, you can define local resources, set the execution environment, the timeout, and the output settings.

Schema validation

The JGAS configuration does not reference an XML Schema Definition (XSD) file to validate the configuration as in standard GAS. Therefore the xmlns:xsi and xsi:noNamespaceSchemaLocation attributes are not used in the APPLICATION element. This simplifies the configuration and allows for flexibility in, for example, the ordering of elements. Validation takes place internally. For compatibility with existing application .xcf files, if schema attributes are defined, they are just ignored.

Inheritance mechanism

The JGAS configuration has no inheritance mechanism as in standard GAS. Therefore the Parent attribute is not used in the APPLICATION element. For compatibility with existing .xcf files, if a Parent attribute is defined, it is just ignored. Template application and service configurations provide defaults that applications can use. See Application and service templates on page 17.

Element order

Configuration elements are well-formed and defined hierarchically as in the standard GAS. This means that child elements must appear within the parent node. As schema validation in JGAS is flexible, the order child elements appear within a parent node is not important. For example if PATH appears after AUTO_LOGOUT, it is not an issue with JGAS.

Example 1 - A simple application configuration file

The simplest application configuration file specifies a path to the compiled application files.

```
<APPLICATION>
  <EXECUTION>
    <PATH>${res.deployment.path}/app</PATH>
    <MODULE>helloapp.42m</MODULE>
    <AUTO_LOGOUT>
      <TIMEOUT>20</TIMEOUT>
    </AUTO_LOGOUT>
  </EXECUTION>
</APPLICATION>
```

Note: The MODULE on page 89 element specifies the name of the .42m module to run. If the module name is the same as the configuration file name, the MODULE element is not necessary.
Application and service templates

Templates define configuration and environment settings that are used by the JGAS. The JGAS configuration file (as.xcf) defines two templates, one for applications and one for services.

The JGAS configuration file does not have an inheritance mechanism as in standard GAS. For example, there is no abstract application definition. These two elements provide default configurations that any application or service deployed in a war file will inherit and use.

- APPLICATION (for JGAS) on page 62
- SERVICE (for JGAS) on page 100

Important:

Only one APPLICATION and SERVICE template is allowed in the JGAS configuration file (as.xcf).

How to implement delegation

The JGAS is able to delegate the start of a web application or a web service to another Genero REST service in order to perform some controls before granting access and starting the application.

How delegation works

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 JGAS 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 JGAS as a standard REST Web service. The delegation service should reside on the same JGAS as the application.
3. The REST service instructs the JGAS 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.
In this figure:

- (1) Start a new application or service of the form /ua/x or /ws/x.
- (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 4: 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 20
To delegate the start of an application or service to a Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

**Configure delegation for application or service**

To delegate the start of an application or service to a Genero REST service, specify a DELEGATE element in the EXECUTION component of your application or service.

The DELEGATE element requires an attribute called service. For the service attribute, specify the Genero REST service that will be in charge of all delegated requests for the application or service. The REST service must be correctly configured in the JGAS.

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**

In JGAS as the application GROUP concept is not supported, there is no requirement to reference a group in the DELEGATE element; as you would in the GAS, for example, "<DELEGATE service="MyGroup/MyDelegateService">".

```
<EXECUTION>
  <PATH>${res.path}</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="MyDelegateService">
    <anyparameter>MyFirstParameter</anyparameter>
    <other>MySecondParameter</other>
  </DELEGATE>
</EXECUTION>
```

Parameters defined in the DELEGATE configuration, such as anyparameter and other in this example, are transmitted to the REST service using HTTP headers. See Pass parameters to the REST service on page 21 for details on 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 21

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 72

The DELEGATE element specifies the Genero REST service in charge of all starting requests for applications.

Application configuration file on page 16
An application configuration file defines an application environment and provides the JGAS with the information needed to run an application.

**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.

```java
IMPORT com
DEFINE req com.HTTPServiceRequest
...
LET req = com.WebServiceEngine.GetHttpServiceRequest(-1)
LET path = req.getUrlPath()
IF path.getIndexOf("/ws/r/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 as HTTP headers.

There is one HTTP header per parameter set in the configuration, and it is of the form `X-FourJs-Environment-Parameter-XXX` where `XXX` is the parameter name and the parameter value is the HTTP header value.

**REST sample**

```java
IMPORT com
DEFINE req com.HTTPServiceRequest
...
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
...
```

**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.

The sample is based on the example application configuration in [Configure delegation for application or service](#) on page 20.
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 req.getURLQuery() function 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/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:

```java
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/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 GURLPath() 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 22.
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("/GetCurstomerInfo",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 23
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.

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 the proxy
When you need to pass additional parameters to a starting proxy, you can do it via environment variables that are then set in a ua or gdc proxy environment before the dispatcher starts it. Each parameter must be set in the HTTP header response when you specify the 307 HTTP return code. The HTTP header name must be of the form:

X-FourJs-Environment-envvar:value

Where envvar is the name of the variable to pass to the proxy and the HTTP header value is the value of the environment variable.

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

Then, with a call to the Genero function `fgl_getenv(VARIABLENAME)` (with the variable name in uppercase), you can retrieve them in your Genero program.

REST sample

Passing parameters to a proxy is not possible for Web services requests, as the proxy is already started. The only way to send the environment with a Web services request is through the HTTP header.

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

Genero program sample:

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

Related concepts

REST service example on page 24
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 26
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")`.

```
IMPORT COM
IMPORT XML
IMPORT FGL WSHelper

PRIVATE CONSTANT C_BASEURL = "/ws/r/MyService/
PRIVATE CONSTANT C_X_FOURJS_ENVIRONEMENT_ = "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.indexOf(C_BASEURL,1)
    IF ind<1 THEN
      CALL req.sendResponse(400,"Invalid request")
    ELSE
      # Dispatch request based on operation name (ex : /ws/r/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.indexOf("/",1)
IF ind>0 THEN
  CALL req.sendResponse(400,"invalid operation")
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_ENVIRONEMENT_||"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 JGAS.

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 JGAS 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 JGAS will be instructed to allow the launch of the application.

Samples for implementing OpenID Connect and SAML authentication and authorization services are provided in $FGLDIR/web_utilities/services, ready for you to use.

**Monitoring or logging requests for a Genero Web service**

You can develop a simple delegation service to monitor and log all requests to a given service. Each request can be logged in a dedicated database by the delegation service. The Genero Application Server can then be instructed to pass the request to the GWSProxy. The delegation for Web services is called each time a request is sent to that service.

**Note:** For applications, logging is only performed at application start up.

### How to implement Single sign-on (SSO)

You can add Single sign-on (SSO) to your applications to allow users to enter one name and password in order to access multiple applications. JGAS supports different kinds of Single sign-on.

**OpenID Connect SSO**

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

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 JGAS delegation mechanism.

OpenID Connect implementation creates a circle of trust between the JGAS 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 JGAS on page 27)
- Providing the redirect URL of the JGAS to the IdP, (see Add OpenID Connect SSO to Web application on page 30)

**Quick start: Set up OpenID Connect in JGAS**

Follow these steps to set up OpenID Connect for your JGAS and Genero Web applications.

Before you begin, you must Configure JGAS for OpenID Connect SSO on page 29.

In this quick start, you configure Genero Browser Client applications for OpenID Connect Single sign-on (SSO), and add them to a gar file that is embedded in a war file with the JGAS. Then you execute the applications with SSO in JGAS.

1. Copy OpenIDConnectServiceProvider.xcf from the FGLDIR/web_utilities/services to your work directory.

   **Note:** In JGAS it is recommended that you only embed the xcf file referencing the delegation REST Web service in the $FGLDIR you will use. This allows you to redeploy the war without having to recreate the circle of trust and reconfigure SSO each time.

2. Add a 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="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>
```

3. Build the Genero Archive file (gar) using the fglgar tool.
At the command line of your work directory type the command that includes your application files and the OpenID Connect service xcf file as shown:

```
fglgar gar --application myApp.xcf --service OpenIDConnectServiceProvider.xcf --output work.gar
```

The `work.gar` is created.

4. Run the fglgar war command to package the Genero Archive and JGAS in a war archive.

```
fglgar war --input-gar work.gar --output work.war
```

The `work.war` file is created.

The war is ready to be deployed.

5. Run the fglgar run command to start the JGAS in standalone mode.

```
fglgar run --war work.war
```

**Note:** SSO requests require HTTPS, and as JGAS has limited HTTPS support (for instance, there is no option to use your own SSL certificate as in the standard GAS), therefore it is only recommended to deploy the war in the standalone JGAS for testing. For development, deploy in any existing Java Enterprise Edition container such as Apache Tomcat®, Jetty, or Glassfish where HTTPS is configured.

6. Execute a Genero Browser Client application with SSO.

   a) Start your browser and enter the application URL. See Run an application in JGAS on page 13.

   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 JGAS 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 JGAS - 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 this section for details on features provided by OpenID Connect SSO support in the JGAS, including attributes gathering or authorization control.
Related concepts
Genero OpenID Connect log file  on page 32
The Genero OpenID Connect implementation produces a log file that helps to identify issues.

Related tasks
Retrieve the OpenID Connect user identifier  on page 30
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 31
Authorize whether a user already authenticated by OpenID Connect SSO can access a Genero application.

Configure JGAS for OpenID Connect SSO
Configure the JGAS 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 29.
2. If the JGAS 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 JGAS is ready to use OpenID Connect SSO to authenticate end users.

Related concepts
Genero OpenID Connect FGLPROFILE  on page 32
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)
   c) In the Authorized redirect URIs field, specify the URI redirection where the JGAS is listening for the response (i.e. https://host:port/jgas/ws/r/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 30.

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

Related concepts
Genero OpenID Connect FGLPROFILE  on page 32
Genero OpenID Connect implementation uses its own FGLPROFILE file.

**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="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"><EXECUTION>
  <PATH>$(res.path.qa)/applications/myapp</PATH>
  <MODULE>App.42r</MODULE>
  <DELEGATE service="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 JGAS configuration, the delegation points to the `OpenIDConnectServiceProvider.xcf` added to the Genero Archive (gar) file referencing the delegation REST Web service in the `$FGLDIR`.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 17.

The JGAS 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 72

The `DELEGATE` element specifies the Genero REST service in charge of all starting requests for applications.

**Delegation use cases** on page 26

Three examples of possible uses for the delegation mechanism.

**Related tasks**

**Retrieve the OpenID Connect user identifier** on page 30

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 31

Authorize whether a user already authenticated by OpenID Connect SSO can access a Genero 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 JGAS on page 27), 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:

```
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.

### Authorization and OpenID Connect SSO

Authorize whether a user already authenticated by OpenID Connect SSO can access a Genero application.

The JGAS must be configured for OpenID Connect Single sign-on (SSO). See Configure JGAS for OpenID Connect SSO on page 29.

With the Genero OpenID Connect implementation, you can add an external program to determine whether an already authenticated user can access a Genero Web application. For example, you may need to restricted access to certain applications or to certain specific email addresses; otherwise any user authenticated by an OpenID Connect provider can access your Genero Web application. It is therefore recommended that you add an authorization program to filter access to your applications.

This external program can be written in Genero or in another programming language. The application AccessProgram.4gl in $FGLDIR/web_utilities/services/openid-connect provides an example of an authorization application written in Genero.

**Note:** The external program must be deployed beside the OpenIDConnectServer.42r program, because it will be executed by that service program. This is by default under $FGLDIR/web_utilities/services/openid-connect/bin.

The authorization program expects two mandatory arguments and the list of OpenID Connect attributes received from the OpenID Connect provider:

```
access-program ocid-userid app-xcf-path \ attribute value [...] ]
```

- The first argument is the OpenID Connect identifier (typically an opaque value returned by the IdP)
- The second argument is the application path.
- Next arguments are optional and define OpenID attributes/value pairs.

Example with a Genero authorization program:

```
fglrun AccessProgram
 "101516043183449889392" \ 
 "qa-test/application" \ 
 "fullname" "genero test" \ 
 "email" "genero@4js.com" \ 
 "country" "France"
```

The external authorization program is specified in the application configuration element by adding an AUTHORIZATION element in the DELEGATE element.

1. Add an AUTHORIZATION element as a child of the DELEGATE element in the application configuration (xcf) file.

2. Within the AUTHORIZATION element, specify the command to execute the external authorization program.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc" >
 <EXECUTION>
  <PATH>$\{res.path.qa\}/applications/myapp</PATH>
```

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 72

The `DELEGATE` element specifies the Genero REST service in charge of all starting requests for applications.

**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 JGAS 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:
SAML SSO

Security Assertion Markup Language (SAML) is a Single sign-on (SSO) protocol supported by the JGAS. It is based on a Genero REST delegation service that 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 JGAS) 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 JGAS

Follow these steps to set up SAML for your JGAS and Genero Web applications.

In this quick start, you configure Genero Browser Client applications for SAML Single sign-on (SSO), add them to a gar file that is embedded in a war file with the JGAS. Then you execute the applications with SSO in JGAS.

1. Copy SAMLServiceProvider.xcf from the FGLDIR\web_utilities\services to your work directory.

    Note: In JGAS it is recommended that you only embed the xcf file referencing the delegation REST Web service in the $FGLDIR you will use. This allows you to redeploy the war without having to recreate the circle of trust and reconfigure SSO each time.

2. Add a DELEGATE element to all Genero Web applications requiring SSO.

    `<APPLICATION Parent="defaultgwc">
    <EXECUTION>
    <PATH>$\{res.path.mypath\}/myapplication</PATH>
    <MODULE> myapp.42r</MODULE>
    <DELEGATE service="SAMLServiceProvider">
    </DELEGATE>
    </EXECUTION>
    </APPLICATION>`

3. Build the Genero Archive file (gar) using the fglgar tool.

    At the command line of your work directory type the command that includes your application files and the OpenID Connect service xcf file as shown:

    fglgar gar --application myApp.xcf --service SAMLServiceProvider.xcf --output work.gar

    The work.gar is created.

4. Run the fglgar war command to package the Genero Archive and JGAS in a war archive.

    fglgar war --input-gar work.gar --output work.war
The work.war file is created.

The war is ready to be deployed.

5. Run the fglgar run command to start the JGAS in standalone mode.

   fglgar run --war work.war

**Note:** SSO requests require HTTPS, and as JGAS has limited HTTPS support (for instance, there is no option to use your own SSL certificate as in the standard GAS), therefore it is only recommended to deploy the war in the standalone JGAS for testing. For development, deploy in any existing Java Enterprise Edition container such as Apache Tomcat®, Jetty, or Glassfish where HTTPS is configured.

6. Execute a Genero Browser Client application with SSO.
   a) Start your browser and enter the application URL. See Run an application in JGAS on page 13. 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 next time you start the same application - or any application delivered by the same JGAS - you will not be prompted for your credentials. The application will start (and be authenticated for) the same SAML user.

**Tip:** Read all of the topics in this section for details on features provided by SAML SSO support in the JGAS; including attributes gathering or authorization control.

**Related concepts**

- Genero SAML configuration on page 40
  Specify entries in the FGLPROFILE file to configure the Genero SAML service provider.

**Related tasks**

- Retrieve identity attributes with SAML on page 38
  Follow these steps to retrieve attributes about user identity when authenticating to SAML IdP.

- Authorization and SAML SSO on page 39
  Authorize whether an user already authenticated by SAML Single sign-on (SSO) can access a Genero application.

**Configure JGAS to support SAML SSO**

Follow these steps to setup Genero SAML service.

Before you can use SAML Single sign-on (SSO) with the JGAS, a circle of trust must be established between the service providers (the JGAS) 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 42.

1. If the JGAS 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 JGAS and a SAML provider. Import the IdP metadata file into the JGAS 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 JGAS.
   a) Start the dispatcher (if needed).
   b) Log in to your SAML provider and create a circle of trust based on the JGAS SAML metadata.
      Generate the metadata from this URL: `http[s]://host:port/jgas/ws/r/SAMLServiceProvider/Metadata`
      Refer to the SAML Identity Provider (IdP) documentation for information about importing the Genero Application Server SAML metadata.

The JGAS is ready to support SAML SSO.

Related concepts
Genero SAML FGLPROFILE on page 42
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 JGAS for SAML Single sign-on (SSO).
- Lists all registered IdPs
- Remove the Idp identified by its URI.

To register a new IdP, you must execute the ImportIdP program with the IdP metadata file or URL. Using a URL can require a proxy configuration in the FGLPROFILE file.

The `ImportIdP.4gl` source code is provided in `$FGLDIR/web_utilities/services/saml/src`, and the compiled version is in the `bin` directory.

Syntax

```
fglrun ImportIdP [options] <url|file>
```

Note:
1. `options` are described in Table 4: `ImportIdP options` on page 36.
2. `url` is the URL of a SAML identity provider.
3. `file` is the metadata file of a SAML identity provider.
Table 4: ImportIdP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-import</td>
<td>Import the IdP specified by the URL or metadata file.</td>
</tr>
<tr>
<td>-list</td>
<td>List all registered IdPs.</td>
</tr>
<tr>
<td>-remove</td>
<td>Remove the registered IdPs.</td>
</tr>
</tbody>
</table>

### Add SAML SSO to a Genero Web application

Follow these steps to add SAML SSO to a Genero Web application.

This task must be performed in the `.xcf` application configuration file for the Genera Application Server.

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

```xml
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.10/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="SAMLServiceProvider"/>
  </EXECUTION>
</APPLICATION>
```

With the above configuration and default JGAS configuration, the delegation points to the `SAMLServiceProvider.xcf` added to the Genero Archive (`gar`) file referencing the delegation REST Web service in the `$FGLDIR`.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 17.

The JGAS 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 72

The `DELEGATE` element specifies the Genero REST service in charge of all starting requests for 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
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="SAMLServiceProvider"/>
    <IDP>http://idp.4js.com</IDP>
  </EXECUTION>
</APPLICATION>
```
The JGAS uses the specified IdP as its Single sign-on (SSO) identity provider.

**Related concepts**

*The ImportIdP program* on page 35

Use the ImportIdP program to register a SAML identity provider.

*DELEGATE* on page 72

The `DELEGATE` element specifies the Genero REST service in charge of all starting requests for 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 JGAS 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 JGAS, it defines what piece of data is sent from the IdP to the JGAS 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 JGAS 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/3.10/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="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 35

Use the ImportIdP program to register a SAML identity provider.

*DELEGATE* on page 72
The **DELEGATE** element specifies the Genero REST service in charge of all starting requests for applications.

**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:

```txt
LET id = fgl_getenv("SAML_ID")
```

**Set the authentication context**
At the JGAS 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 (JGAS) to define how a user must be authenticated by the Identity Provider (IdP). The JGAS 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"
>
  <EXECUTION>
    <PATH>${res.path.qa}/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="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 JGAS will return an access denied page.

**Related concepts**
**DELEGATE** on page 72
The **DELEGATE** element specifies the Genero REST service in charge of all starting requests for applications.

**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 JGAS during authentication setup.
To retrieve the SAML attributes returned by the IdP in your Genero application, add a `fgl_getenv()` call for each attribute specified in the XCF file with a prefix of `SAML_`.

```plaintext
LET email = fgl_getenv("SAML_email")
LET fullname = fgl_getenv("SAML_fullname")
LET country = fgl_getenv("SAML_country")
```

The Genero application retrieves the requested identity attributes.

**Authorization and SAML SSO**

Authorize whether an user already authenticated by SAML Single sign-on (SSO) can access a Genero application.

The JGAS must be configured for SAML SSO. See Configure JGAS to support SAML SSO on page 34.

With the Genero SAML implementation, you can add an external program to determine whether an already authenticated user can access a Genero Web application.

This external program can be written in Genero or in another programming language.

The authorization program expects two mandatory arguments and the list of SAML attributes received from the Identity Provider (IdP):

```plaintext
access-program saml-userid app-xcf-path ↓ attribute value [...] ]
```

- The first argument is the SAML identifier. It depends on the ID format specified in the JGAS configuration and by the IdP.
- The second argument is the application path.
- Next arguments are optional and define SAML attributes/value pairs.

Example with a Genero authorization program:

```plaintext
fglrun AccessProgram
 "AZEd3R4" \n "qa-test/application" \n "fullname" "genero test" \n "email" "genero@4js.com" \n "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
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
>
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="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 72
The DELEGATE element specifies the Genero REST service in charge of all starting requests for applications.

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 JGAS, you will not be prompted
for your credentials. The Web application will start and authenticate with the same SAML user.

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 5: 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 JGAS, which is how the JGAS is represented to other SAML partners. Mandatory. Default is urn:genero.</td>
</tr>
<tr>
<td>saml.allowUnsecure</td>
<td>Defines whether the JGAS accepts unsecured authentication mechanisms. Default is false (recommended). A SAML authentication mechanism is unsecured if communication between the Identity Provider (IdP) and the JGAS 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 41.</td>
</tr>
</tbody>
</table>
### FGLPROFILE file entry | Description

| saml.wantAssertionsSigned | 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 JGAS returns an access denied HTML page.

This entry also adds the `wantAssertionsSigned` attribute to the SAML metadata describing the SAML needs of the JGAS.

| saml.wantResponseSigned | 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 JGAS returns an access denied HTML page.

---

**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 JGAS 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 JGAS 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 JGAS 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 42

Genero SAML Single sign-on (SSO) implementation uses its own FGLPROFILE file.

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 JGAS 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/services/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">
  <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.

### Application Web address

To access an application deployed in a war file, there are different options for entering the URL in the browser's address bar.

#### Application URI

Application URIs used in the JGAS are simplified so that you can use the same URIs as standard GAS if you wish. The application URI is defined by the format:

http://host:port/war_file_name/ua/r/application_name.

For example, to run an application called `HelloWorld` in standalone mode in the `hello.war` file, an URI can be written in two ways:

- http://localhost:8080/hello/ua/r/HelloWorld
- http://localhost:8080/ua/r/HelloWorld

The first option uses the real URL. The second one performs a simple redirect to reach the first one.

#### Web Services URI

Web services URI contain `/ws/r`. No redirection is required in this case as standard GAS Web services URLs are used in JGAS.
Serving static files

The JGAS standalone mode has a Web server capability and is able to serve static files from its document root directory. If you want to change the document root, you can specify this by setting the document-root property.

Static file server

The URL /files is used to serve static files. For example, from the JGAS deployed on the localhost, static files may be viewed in the browser at:

http://localhost:8080/files

The default document root is set by the path ${HOME}/web in Linux®/UNIX® and HOMEPATH\web in Windows®. For example, in a Windows® system this resolves to C:\Users\user_name\web.

Change the document root directory

If you want the static files on your JGAS implementation to come from, for example /Users/my/projects directory instead of the default, you can specify it with the option (-E) and by setting the document.root property at the command line as shown in the example:

fglgar run --war hello.war -E document.root=/Users/my/projects

JGAS User Interface

The JGAS provides a graphical front-end for running applications on the standalone JGAS. From here you can also enable and configure applications deployed in a war file.

JGAS home page

The home page of the JGAS user interface provides access to the applications or services deployed in a war file. Resources can be added or modified for each application or globally.

To open the JGAS user interface, type the URL into your browser as follows:
http://host:port/war_file_name, where war_file_name is the name of the war file deployed.

For example working on the standalone JGAS, enter http://localhost:8080/hello to use applications deployed in the hello.war file. By default JGAS listens on port 8080. The name of the war file, hello in the example, provides the path to locate the deployed war.

The JGAS user interface home page opens.
Registered applications and services

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global settings</td>
</tr>
<tr>
<td>helloapp</td>
</tr>
<tr>
<td>helloservice</td>
</tr>
</tbody>
</table>

Note: Global settings affect all applications and of all deployed war files on the server. The global settings you customize, persist when you redeploy the war file.

To configure global resources, click on the Edit configuration button opposite the Global Settings to open the Resources Global page. See JGAS global resources page on page 46.

Application settings: a row of icons is shown opposite each application name.

These options allow you to perform the following actions:

**Edit configuration**  
Click on the Edit configuration button opposite an application to open the application Resources page. See JGAS application resources page on page 48.

**Enable/Disable**  
Use this option to enable or disable an application or service. Click on the Enable/Disable button opposite the application name.
**Start**

Use this option to run an application or start a Web service that is configured on your JGAS. Click on the **Start** button opposite the application name. Your application is then displayed in the browser page within the Genero Browser Client (GBC) user interface.

---

**JGAS global resources page**

The JGAS provides default configurations for all applications deployed in a war file. The Resources Global page provides you with options to edit JGAS configurations or override resources by setting new values.

To configure settings for the JGAS, open the Resources Global page. From the home page of the JGAS user interface click on the **Edit configuration** button opposite **Global Settings**. The **Resources Global** page opens.

![Configuration page](image)

**Resources Global**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td></td>
</tr>
</tbody>
</table>

**Add a resource**

Key: Value: Add

**Remove a resource**

Key: Remove

*Figure 6: JGAS resources global page*

In this section you can configure JGAS settings. You can add new resources or override existing ones. These changes will affect all applications deployed in a war file. Any resource modified globally can be overloaded in the application or service configuration.

Under the **Name** and **Value** columns, resource configurations that have been applied are listed. If no resources have been configured, the row displayed is blank.
**Add a resource**

Under the heading **Add a resource** there is a **key** and **value** field. To edit an existing JGAS resource or add a new resource, perform the following steps:

1. Enter the name of a resource in the **key** field. You can override any configuration set by a resource in the as.xcf file by entering the id of the resource, for example `log.console.categories`, `res.gbc`, etc. in the **key** field.
2. In the **value** field, enter the required value and click the **Add** button.
3. The new key and value is displayed under the **Name** and **Value** columns. To apply the change click on the **Update** button.

**Important:** All applications inherit global changes and they are marked as updated. The configuration change must be applied individually to each application on the home page. Started applications continue with their original configuration. New applications use the new configuration.

4. To reload the configuration for an application, click on the **Return to the index** button at the top of the page to open the home page.
5. A **Configuration updated** icon and an **Apply changes** button are displayed opposite application names in the home page.

![Configuration update icons](image)

**Figure 7: Configuration update icons**

To apply the changes for an application, click on the **Apply changes** button. When reloading the configuration for a service, the pool of services is stopped and restarted using the new configuration.

To see what resources are configured for an application, see the **Configuration dump** section in the JGAS application resources page on page 48.

**Remove a resource**

In this section you can remove a resource you configured on your JGAS. To remove a resource, select it from the drop-down list in the **key** field and click on the **Remove** button. If no resources have been configured, the drop-down list is empty.

**Important:** When you remove a resource, default JGAS values are reapplied. The resource is not actually removed or deleted.

**Related concepts**

- JGAS configuration file on page 55
The JGAS configuration file, as.xcf, provides default configurations for embedded applications.

**JGAS application resources page**

The JGAS provides configurations for each applications deployed in a war file. The application resources page provides options for you to edit configurations or override resources by setting new values.

To configure settings for an application deployed by the JGAS, open the JGAS application **Resources** `app_name` page. From the home page of the JGAS user interface click on the **Edit configuration** button opposite the name of an application or service. The **Resources** page for the application opens.

![Resources helloapp](image)

**Resources helloapp**

No user-defined resources

**Add a resource**

Key:  
Value:  
Add

**Configuration is valid**

**Configuration dump**

```
<CONFIGURATION>
  <RESOURCES>
    <RESOURCE Id="res.fgldir" Source="INTERNAL">C:\fgl</RESOURCE>
    <RESOURCE Id="FGLDIR" Source="ENVIROM">FGLDIR</RESOURCE>
    <RESOURCE Id="res.gedir" Source="ENVIRON">GREDIR</RESOURCE>
    <RESOURCE Id="res.fglgui">1</RESOURCE>
    <RESOURCE Id="res.sys.path" Source="ENVIRON">PATH</RESOURCE>
    <RESOURCE Id="res.access.control">ALL</RESOURCE>
    <RESOURCE Id="res.public.resources">deployment</RESOURCE>
    <RESOURCE Id="res.dvm.wa">C:\fgl/bin/fglrun</RESOURCE>
  </RESOURCES>
</CONFIGURATION>
```

*Figure 8: JGAS resources application page*

**Resources**

In this section you can configure JGAS settings for an application. You can add new resources or override
JGAS logging

The logging mechanism allows you to configure logging for both the console and log file. Both can be enabled or disabled independently and may be viewed for troubleshooting.

You can override logging resources globally for the JGAS, or for each individual application deployed in the war file at runtime from the JGAS user interface.

You can also override them when running the war file at the command line.

Related tasks
Override category log resource on page 50
The JGAS configuration file (as.xcf) provides default configurations for resources that applications use, but you can also override an application resource at runtime.

Override log resource at command line on page 52
When you execute applications packaged in Java Web Archive (war) file with the fglgar command line tool, you can also override JGAS resources with the --resource-overwrite option.

JGAS log file

The JGAS provides for a daily log file to be created for the application you run. If you need to troubleshoot an application, enable the DAILYFILE element to capture messages for requests, proxies, system errors, and VM sessions.

A log file is created in the directory specified by the default resource log.file.path. In a Windows® platform, for example, the log file is found in the C:\Users\<username>\.gas\war_file_name\log\current_date\ directory.

The log file is saved to a directory named with the current date in the format, YYYYMMDD, for example, 20170106. As you can redefine resources for each application or service deployed in the same war file, the logging configuration can vary from one application to another. Therefore, a separate DVM log file is created for each application you run. If the log file is enabled, it is identified by the application name, for example, <appname>.log.

The log file may contain the following information and messages:

- A list of the field identifiers that define the detail in the logged output appears at the top; defaults are set in the log.file.columns resource. See FORMAT (under LOG for JGAS) on page 78.
  
  #Fields: date time relative-time location thread-id contexts event-type event-params
- The configuration file settings for the application are shown at the top.
- As the application runs, messages are written for HTTP requests and responses, the starting of proxies, system errors, etc. This output depends on the categories of logs specified in the CATEGORIES element.
- Messages for different VM sessions are shown. These are prefaced in the contexts column by the output VM=session_number. Session numbering starts at zero and is incremented by one for each new VM session started. Sessions are also identified by their session id (SID).

  2017/01/20 10:20:37.292 413.739 [UAProcess.java:96] 21
  VM=0;SID=77eda293a4604483b10647442eedf929 "VM State" Move state VM
  Initialized --> VM connected.

  VM=1;SID=e5e11c2222fc43fa8121a8cbaad5a8cf "VM State" Move state VM
 Initialized --> VM connected

Related tasks

Override category log resource on page 50

The JGAS configuration file (as.xcf) provides default configurations for resources that applications use, but you can also override an application resource at runtime.

Override log resource at command line on page 52

When you execute applications packaged in Java Web Archive (war) file with the fglgar command line tool, you can also override JGAS resources with the --resource-overwrite option.

Override category log resource

The JGAS configuration file (as.xcf) provides default configurations for resources that applications use, but you can also override an application resource at runtime.

1. Run your application on the JGAS.
   a) If your standalone JGAS is not running, start it from the command line. For details about starting the JGAS, see Run an application in JGAS on page 13
   b) To run your application locally, in a browser enter the application address:
http://localhost:8080/war_file_name

Where war_file_name is the name of the war file deployed.

The JGAS user interface home page opens showing the applications and services deployed in the war file. See JGAS home page on page 44.

2. From the home page, open the Resources page by clicking on the Edit configuration button opposite the application or service you wish to configure.

The Resources page opens. See JGAS application resources page on page 48.

3. In the key field enter the name of the console's log category resource.

   log.console.categories

   Tip: You can see the current setting of the log.console.categories resource under the Configuration dump heading.

4. In the value field enter a valid log filter category, for example, "ERROR INFO" and click the Add button.

   For more information on valid log filter categories that generate error and information messages in the logs, see CATEGORIES (under LOG for JGAS) on page 66.

   The new key and value is displayed under the Name and Value columns at the top of the page.

5. To apply the change click on the Update button.

   Important: Even though the resources have been updated, the configuration is not yet activated. The configuration must be reloaded on the home page. If the application is already started, it continues with its original configuration.

6. To reload the configuration, click on the Return to the index button at the top of the page to open the home page.

7. A Configuration updated icon and an Apply changes button are displayed opposite the application name, to show that configurations have changed, and need to be applied.

   To apply the changes, click on the Apply changes button.

   Figure 9: Configuration update icons

8. Select OK in the application reload pop-up window.

   Figure 10: Reload Configuration pop-up page

Run your application to see the log changes in the console.

Note: If you redeploy the updated war file, the customized resources are reapplied, as long as the war file name is not changed. If the war file is deployed to a different server, customized resource settings need to be reapplied.
Override log resource at command line

When you execute applications packaged in Java Web Archive (war) file with the fglgar command line tool, you can also override JGAS resources with the --resource-overwrite option.

If you use the override resource option -E or --resource-overwrite at the command line, it takes precedence over all resources configured in the interface for the application or service or the global resources set.

1. The fglgar run command is used to start the JGAS in standalone mode and deploy applications. At the command line type the command for your war file, this is shown as hello.war in the example:

   ```
   fglgar run -w hello.war -E log.console.categories=ALL
   ```

   The -E or --resource-overwrite option overwrites the default category of logs written to the standard output for applications deployed in the war.

   The output at the command line indicates that the JGAS is started.

2. In a browser enter the address of your deployed application, for example http://localhost:8080/ua/r/HelloWorld

   As you interact with the application in the browser you should see that the output at the command line includes the expected category of logs.

   For more information on valid log filter categories that generate error and information messages in the logs, see CATEGORIES (under LOG for JGAS) on page 66.

JGAS auto logout

The JGAS 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.

Default settings for auto logout are configured in the configuration file, as.xcf.

You can configure auto logout for your applications by setting the TIMEOUT (for auto logout) on page 105 element. You also have the option of setting conditions when the auto logout feature is, or is not, applied, using the COMMAND (for auto logout) on page 68 or CLASS (under AUTO_LOGOUT for JGAS) on page 67 elements.

This section provides you with an example to override auto logout for your applications. See Override auto logout resource on page 52.

Override auto logout resource

The JGAS configuration file (as.xcf) provides default configurations for resources that applications use, but you can also override resources, such as the auto logout. You can do this globally at runtime from the user interface.

Before you begin:

It is assumed the applications deployed in the war are configured to reference the app.timeout.autologout resource as shown:

```
<AUTO_LOGOUT>
  <TIMEOUT>${app.timeout.autologout}</TIMEOUT>
</AUTO_LOGOUT>
```

1. Run your application on the JGAS.
   a) If your standalone JGAS is not running, start it from the command line. For details about starting the JGAS, see Run an application in JGAS on page 13
b) To run your application locally, in a browser enter the application address:

http://localhost:8080/war_file_name

Where war_file_name is the name of the war file deployed.

The JGAS user interface home page opens showing the applications and services deployed in the war file. See JGAS home page on page 44.

2. From the home page, open the Resources Global page by clicking on the Edit configuration button opposite Global Settings.

The Resources Global page opens. See JGAS global resources page on page 46.

3. In the key field enter the name of the auto logout resource.

app.timeout.autologout

Tip: You can see the current setting of the app.timeout.autologout resource under the Configuration dump heading in the Resources page for an application, see JGAS application resources page on page 48.

4. In the value field enter a valid timeout setting in seconds, for example, “80” and click the Add button.

The new key and value is displayed under the Name and Value columns at the top of the page.

5. To apply the change click on the Update button.

Important: Even though the resources have been updated, the configuration is not yet activated. The configuration must be reloaded on the home page. If the application is already started, it continues with its original configuration.

6. To reload the configuration, click on the Return to the index button at the top of the page to open the home page.

7. Configuration updated icons and an Apply changes buttons are displayed opposite each application name, to show that configurations have changed, and need to be applied.

To apply the changes, click on the Apply changes button.

Figure 11: Configuration update icons

8. Select OK in the application reload pop-up window.

Figure 12: Reload Configuration pop-up page

Run your application to check that the auto-logout changes are taking effect. In the console output, you should see that the application times out after the specified period of inactivity on the application.

Note: If you redeploy the updated war file, the customized resources are reapplied, as long as the war file name is not changed. If the war file is deployed to a different server, customized resource settings need to be reapplied.
Upgrading

These topics describe the steps you need to take to upgrade to the next release of JGAS, 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 JGAS. Failure to do so can result in issues when rendering your Web applications.

New Features of the JGAS

These topics provide a look back at the new features introduced with each release of the JGAS.

JGAS 1.00 new features

A summary of new features and changes in functionality introduced with Genero Application Server for Java (JGAS) 1.00.

Table 6: Engine and Architecture, Version 1.00

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>A GAS written in Java (JGAS) is provided. It can be packaged in a Java Web Archive (war) file with applications for development environment testing as standalone without the need for a separate GAS installation or a Java Enterprise Edition server and you can access your applications and services via a browser. The CACHE_CONTROL_MAX_AGE element allows you to specify the duration files sent by the JGAS are held in front-end cache. The cache.control.max.age resource defines the setting for the front-end cache in CACHE_CONTROL_MAX_AGE.</td>
<td>No additional reference. See CACHE_CONTROL_MAX_AGE on page 65 See CACHE_CONTROL_MAX_AGE on page 65</td>
</tr>
</tbody>
</table>

Table 7: Single Sign-On (SSO) and delegation

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenID Connect and SAML are Single sign-on (SSO) protocols supported by the JGAS. They are based on Genero delegation REST services delivered in the FGLGWS package under $FGLDIR/web_utilities/services/. The JGAS is able to delegate the start of a Genero Web service (/ws/r), or GBC/GDC (/ua/r) application to another REST service in order to perform some controls (such as authentication, authorization, or monitoring) in a single and centralized Genero program.</td>
<td>See How to implement Single sign-on (SSO) on page 27. See How to implement delegation on page 17.</td>
</tr>
</tbody>
</table>

Note: The new features listed in this topic are available in the latest version of the JGAS. Contact your support channel for more details.
JGAS Configuration Reference

These topics provide reference details for JGAS configuration.

JGAS configuration file

The JGAS configuration file, as.xcf, provides default configurations for embedded applications.

As the JGAS is packaged in the FGLGWS product, the default configuration file is located in the FGLDIR/web_utilities/jgas/WebContent/WEB-INF/ directory.

When the JGAS is embedded in a war file for application deployment, it is located in its WEB-INF directory.

Configuration elements

The JGAS configuration file as.xcf is an XML file. The main elements are similar to the standard GAS as.xcf file:

• Its root element is CONFIGURATION.
• RESOURCES is basically the same as RESOURCE_LIST in the standard GAS, except there is only one RESOURCES element allowed. There are also no platform specific resources for Windows® and UNIX®.
• INTERFACE is the same as standard GAS element INTERFACE_TO_CONNECTOR.

The JGAS configuration does not provide an abstract application definition. Instead two templates, one for APPLICATION (for JGAS) on page 62 and one for SERVICE (for JGAS) on page 100 provides default settings. Only one APPLICATION and SERVICE template is allowed in the JGAS configuration file.

Configuration elements not in JGAS

The following configuration elements do not exist in JGAS:

• MONITOR
• GROUP. The application group concept is not supported. There is no requirement, therefore, to reference a group in application or service configuration.
• PROMPT (for AUTO_LOGOUT (for JGAS) on page 64). Re-login after auto logout is not supported.
• FILE_TRANSFER and its child element TIMEOUT
• Component configuration elements are not supported in the JGAS. Where in the standard GAS there is a using attribute in, for example the APPLICATION element, which can reference a component element, this is not used in JGAS.

Note: An implementation of the MONITOR and re-login (after auto logout) features are under development for JGAS and will be delivered in a later release.

Log elements

The LOG (for JGAS) on page 86 element allows you to configure logging for both console and log file.

• CONSOLE (under LOG for JGAS) on page 71
• DAILYFILE (under LOG for JGAS) on page 72

These elements can be enabled or disabled independently. If the CONSOLE log is enabled all DVM output is displayed to the standard output. If the DAILYFILE is enabled, the DVM log is redirected to a log file. Logging filters can be applied for both the console and log file with the CATEGORIES (under LOG for JGAS) on page 66 element.
Schema validation

The JGAS configuration does not reference an XML Schema Definition (XSD) file to validate the configuration as in standard GAS. Therefore the xmlns:xsi and xsi:noNamespaceSchemaLocation attributes are not used in the APPLICATION element. This simplifies the configuration and allows for flexibility in, for example, the ordering of elements. Validation takes place internally. For compatibility with existing application xcf files, if schema attributes are defined, they are just ignored.

Element order

Configuration elements are well-formed and defined hierarchically as in the standard GAS. This means that child elements must appear within the parent node. As schema validation in JGAS is flexible, the order child elements appear within a parent node is not important. For example if PATH appears after AUTO_LOGOUT, it is not an issue with JGAS.

JGAS configuration file hierarchy

A listing of elements valid in the JGAS configuration file, shown hierarchically.

Select an element name to be taken to the topic discussing that element.

- CONFIGURATION (for JGAS) on page 69
  - RESOURCES (for JGAS) on page 98
    - RESOURCE on page 97
  INTERFACE (for JGAS) on page 84
    - GBC_LOOKUP_PATH on page 81
    - REPORT_VIEWER_DIRECTORY on page 94
    - REPORT_REMOTE_URL_PREFIX on page 94
    - HTTP on page 83
      - SESSION_COOKIE on page 101
      - CACHE_CONTROL_MAX_AGE on page 65
      - APPLICATION (for HTTP) on page 61
      - SERVICE (for HTTP) on page 100
  - LOG (for JGAS) on page 86
    - CONSOLE (under LOG for JGAS) on page 71
      - CATEGORIES (under LOG for JGAS) on page 66
      - FORMAT (under LOG for JGAS) on page 78
    - DAILYFILE (under LOG for JGAS) on page 72
      - CATEGORIES (under LOG for JGAS) on page 66
      - FORMAT (under LOG for JGAS) on page 78
  - APPLICATION (for JGAS) on page 62
    - RESOURCE on page 97
    - EXECUTION (under APPLICATION for JGAS) on page 77
      - ENVIRONMENT_VARIABLE
      - COMMAND (under EXECUTION for JGAS) on page 69
      - PATH
      - DVM
      - MODULE
      - PARAMETERS
Syntax diagrams indicate the rules and requirements for the XML-based GAS configuration file elements.

These elements may be found in a JGAS configuration file (default $FGLDIR/web_utilities/jgas/WebContent/WEB-INF/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>
```
• 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 8: 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 (ellipses) 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>
<tr>
<td>↓ e1 ↓ e2 ... ↓</td>
<td>Square braces containing elements separated by a pipe indicate an optional element to be selected from the list.</td>
</tr>
<tr>
<td>↓ e1 ↓ e2 ... ↓</td>
<td>Curly braces containing elements separated by a pipe indicate a mandatory element to be selected from the list.</td>
</tr>
<tr>
<td>[...]</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>[, ... ]</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 JGAS 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* on page 60 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</ALLOW_FROM>
    <ALLOW_FROM>fd::</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 "fd::".

**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.

To allow access from the localhost, in the JGAS configuration file you need to change the application element for *gwc-demo* from:

```xml
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo">
    <EXECUTION>
        <PATH>$(res.path.fgldir.demo)</PATH>
        <MODULE>demo.42r</MODULE>
        <ACCESS_CONTROL>
            <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
        </ACCESS_CONTROL>
    </EXECUTION>
</APPLICATION>
```

To:

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

**Parent elements**

This element is a child of one of the following elements:

- **EXECUTION (under APPLICATION for JGAS)** on page 77
- **EXECUTION (for a service)** on page 76
ALLOW_FROM

The ALLOW_FROM element specifies rules for access to GAS applications or services from specific hosts.

Syntax

```xml
<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

```xml
<ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
```

The resource `res.access.control` is defined with the value `NOBODY` by default. The default GAS deployment for demo applications references this resource.

**Note:** Access control rules will be ignored by the standalone dispatcher (httpdispatch).

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 58

APPLICATION (for an application)

This APPLICATION element defines an application in an external application configuration file.

Syntax

```xml
<APPLICATION Id=app-name
   [mode=mode-name]>
   [...]<DESCRIPTION>...</DESCRIPTION>
   [...]<RESOURCE>...</RESOURCE>
   [...]<EXECUTION>...</EXECUTION>
   [...]<AUTO_LOGOUT>...</AUTO_LOGOUT>
   [...]<UA_OUTPUT>...</UA_OUTPUT>
</APPLICATION>
```

1. The `app-name` identifies the application. For external configuration files, the `Id` attribute is ignored. The `Id` specified is compared to the application name in the request.
2. The `mode-name` when set to "sticky", defines a Web service as a sticky Web service. See the Configure sticky Web services topic in the Genero Application Server User Guide.

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` (under `APPLICATION` for JGAS) on page 77 elements.
4. Zero or one `AUTO_LOGOUT` (for JGAS) on page 64 elements.
5. Zero or one `UA_OUTPUT` on page 105 elements. This element is used for all UI applications.

Usage

You use this element to configure Web applications you wish to make accessible through the JGAS.

Example external Web application configuration file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION>
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
  <AUTO_LOGOUT>
    <TIMEOUT>$(app.timeout.autologout)</TIMEOUT>
  </AUTO_LOGOUT>
  <UA_OUTPUT>
    <PROXY>$(res.uaproxy.cmd)</PROXY>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <GBC>gwc-dev</GBC>
    <TIMEOUT Using="cpn.wa.timeout"/>
  </UA_OUTPUT>
</APPLICATION>
```

For more information, see Configuring applications on JGAS on page 15.

Parent elements

When used in an application configuration file (.xcf), it is the top-most element.

Related concepts

APPLICATION (for a service) on page 63

This `APPLICATION` element defines a service 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

```xml
<HTTP>
  ...
  <APPLICATION>
    [<HEADER>...</HEADER>]
  </APPLICATION>
  ...
</HTTP>
```
Child elements

The **APPLICATION** element may contain zero to many **HEADER** 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.

Example usage

```xml
<INTERFACE>
    <ROOT_URL_PREFIX>${res.rooturlprefix}</ROOT_URL_PREFIX>
    <REPORT_VIEWER_DIRECTORY>${res.gredir}/viewer</REPORT_VIEWER_DIRECTORY>
    <GBC_LOOKUP_PATH>${res.gbc.lookup.path}</GBC_LOOKUP_PATH>
    <HTTP>
        <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>
```

Parent elements

This element is a child of the **HTTP** element.

**APPLICATION** (for JGAS)

This **APPLICATION** element defines an application with default settings within the JGAS configuration file.

Syntax

```xml
<APPLICATION>
    [<EXECUTION>...]</EXECUTION>
    [<AUTO_LOGOUT>...]</AUTO_LOGOUT>
</APPLICATION>
```

Child elements

The **APPLICATION** element may contain the following elements:

1. Zero or one **EXECUTION** (under **APPLICATION** for JGAS) elements.
2. Zero or one **AUTO_LOGOUT** (for JGAS) elements.

Usage

You use the **APPLICATION** element to provide default configuration settings that any application contained in a deployed **war** file inherit and use.

**Note:** In the JGAS there is no abstract application definition. All applications and services inherit configurations from the default templates. Only one **APPLICATION** and **SERVICE** template is allowed in the JGAS configuration file (**as.xcf**).

**Example JGAS as.xcf application configuration**

```xml
<APPLICATION>
    <EXECUTION>
```
<ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>

...<WEB_COMPONENT_DIRECTORY>webcomponents</WEB_COMPONENT_DIRECTORY>
<UA_OUTPUT>
  <PUBLIC_IMAGEPATH>${res.public.resources}</PUBLIC_IMAGEPATH>
  <TIMEOUT>
    ...
  </TIMEOUT>
  <GBC>${res.gbc}</GBC>
  <GDC_SHORTCUT>${res.deployment.root)/tpl/shortcut/gdc-http.gdc</GDC_SHORTCUT>
  </UA_OUTPUT>
</DELEGATE>
</EXECUTION>
<AUTO_LOGOUT>
  <TIMEOUT>${app.timeout.autologout}</TIMEOUT>
</AUTO_LOGOUT>
</APPLICATION>

Parent elements
In the JGAS configuration file, this element is a child of the CONFIGURATION (for JGAS) element.

Related concepts
Application and service templates on page 17
Templates define configuration and environment settings that are used by the JGAS. The JGAS configuration file (as.xcf) defines two templates, one for applications and one for services.

APPLICATION (for a service)
This APPLICATION element defines a service in an external application configuration file.

Syntax

```
<APPLICATION [Id="app-name"]
  [mode="mode-name"]>
  [...]  
  [...]
  [...]  
  [...]  
  [...]  
</APPLICATION>
```

1. The app-name identifies the application. For external configuration files, the Id attribute is ignored. The Id specified is compared to the application name in the request.
2. The mode-name when set to "sticky", defines a Web service as a sticky Web service. See the Configure sticky Web services topic in the Genero Application Server User Guide.

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 JGAS.

Example usage

```
<APPLICATION Id="webapp" >
  <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.

Related concepts

APPLICATION (for an application) on page 60
This APPLICATION element defines an application in an external application configuration file.

AUTO_LOGOUT (for JGAS)

This AUTO_LOGOUT (for JGAS) is a configuration element that defines the auto logout mechanism used for applications.

Syntax

```
<AUTO_LOGOUT>
  [<TIMEOUT></TIMEOUT>]  
  [<COMMAND></COMMAND>]  
  [<CLASS></CLASS>]  
</AUTO_LOGOUT>
```

Child elements

The AUTO_LOGOUT element may contain the following child elements:

- Zero or one TIMEOUT (for auto logout) on page 105 elements.
- Zero or one COMMAND (for auto logout) on page 68 elements.
- Zero or one CLASS (under AUTO_LOGOUT for JGAS) on page 67 elements.

Usage

You use this element to specify an auto logout that detects when an application has no user activity. The DVM waits the number of seconds specified by the TIMEOUT (for auto logout) on page 105 element before triggering an auto logout event and sending the application a log out message. By default an ending page is displayed showing the following message:

You have been logged out.

Usage example - default JGAS auto logout

```
<APPLICATION>
  <EXECUTION/>
  <AUTO_LOGOUT>
    <TIMEOUT>${app.timeout.autologout}</TIMEOUT>
  </AUTO_LOGOUT>
</APPLICATION>
```
The default TIMEOUT, set by the app.timeout.autologout resource, is shown in the example. This default timeout duration is set to 0 seconds, which means the auto logout is ignored and applications keep running. For an example of how you can override this at runtime, see Override auto logout resource on page 52.

**Usage example - use auto logout directly**

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>
  <Execution>
    <Path>my_app_dir</Path>
    <Module>my_module</Module>
  </Execution>
  <AutoLogout>
    <Timeout>30</Timeout>
  </AutoLogout>
  ...
</Application>
```

**Example with COMMAND to validate execution of auto logout**

The auto logout feature allows a command to validate whether the auto logout can take place. For an example of this configuration, see COMMAND (for auto logout) on page 68.

**Example with CLASS to validate execution of auto logout**

The auto logout feature allows you to implement the validation of auto logout in Java using the CLASS element. For an example of this configuration, see CLASS (under AUTO_LOGOUT for JGAS) on page 67.

**Parent elements**

The AUTO_LOGOUT element is a child of the APPLICATION (for JGAS) on page 62 element.

**Related concepts**

TIMEOUT (for auto logout) on page 105
This TIMEOUT element sets a timeout to handle auto logout.

COMMAND (for auto logout) on page 68
This is an element of AUTO_LOGOUT that provides a mechanism for the JGAS to override an application's auto logout.

**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 JGAS 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 JGAS configuration file. When the JGAS 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
```

Usage example

```
<INTERFACE>
  ...
  <HTTP>
    <SESSION_COOKIE Secure="FALSE" />
    <CACHE_CONTROL_MAX_AGE>$(cache.control.max.age)</CACHE_CONTROL_MAX_AGE>
    <!-- Applications custom headers -->
    ...
  </HTTP>
</INTERFACE>
```

The resource `cache.control.max.age` is set to the default value of 300 seconds. You can override this by changing the resource value.

Parent elements

This element is a child of the `HTTP` on page 83 element.

CATEGORIES (under LOG for JGAS)

This configuration element specifies the type of log messages to be captured.

Syntax

```
<CATEGORIES>name [ ... ]</CATEGORIES>
```

1. The `name` can specify multiple category values by listing multiple filter names, separated by spaces. Valid category filter names are shown in Table 9: Category filters on page 66

Table 9: Category filters

<table>
<thead>
<tr>
<th>Filter Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>Log error information.</td>
</tr>
<tr>
<td>INFO</td>
<td>Log information.</td>
</tr>
<tr>
<td>VM</td>
<td>Log communications with the Dynamic Virtual Machine.</td>
</tr>
<tr>
<td>ALL</td>
<td>Activate all categories. This is useful for debugging and should only be used in production or set for a short period of time when required, as it generates many log entries.</td>
</tr>
</tbody>
</table>

Usage

You use this element to specify the categories filtered in a log message. It specifies filters for messages written to the logs or sent to the standard output. The defaults are defined by the JGAS resources:

- `<RESOURCE Id="log.console.categories">ERROR</RESOURCE>`
- `<RESOURCE Id="log.file.categories">ERROR INFO</RESOURCE>`

Usage override

The recommended usage is to override the `CATEGORIES` resource value at the command line when running the war file. By default only the CONSOLE log is enabled with the ERROR category. When troubleshooting, all logs can
be specified by setting the CATEGORY element with the ALL setting. This override example applies the ALL filter category to the console log resource:

```
fglgar run -w hello.war -E log.console.categories=ALL
```

The -E or --resource-overwrite option overwrites the default category of logs. Information and error message for applications deployed in the war are generated and written to the standard output.

**Parent elements**

This element is a child of the following elements:

- **CONSOLE** (under LOG for JGAS) on page 71
- **DAILYFILE** (under LOG for JGAS) on page 72

**CLASS (under AUTO_LOGOUT for JGAS)**

The CLASS element implements a Java interface for auto logout configuration.

**Syntax**

```xml
<CLASS>timeout-interface</CLASS>
```

1. *timeout-interface* specifies the Java class found in `FGLDIR/web_utilities/jgas/WebContent/WEB-INF/lib/jgas.jar` which validates the auto logout.

**Usage**

You use this element to implement the validation of auto logout.

**Usage example**

```
<AUTO_LOGOUT>
    <TIMEOUT>30</TIMEOUT>
    <CLASS>com.fourjs.gas.uaproxy.autologout.VetoableAutoLogout</CLASS>
</AUTO_LOGOUT>
```

The Java class provided in the example validates the auto logout.

**Sample code that prevents auto-logout during work hours**

You must provide the name of the class. It must implement the `com.fourjs.gas.uaproxy.autologout.IVetoableAutoLogout` interface.

```
package com.fourjs.gas.uaproxy.autologout;

public interface IVetoableAutoLogout {
    public boolean autoLogout();
}
```

In this sample class the auto-logout implementation rejects auto-logout between the hours of 9 AM and 6 PM.

```
package com.fourjs.gas.uaproxy.autologout;

import java.util.Calendar;

public class AutoLogoutFrom9To18 implements IVetoableAutoLogout {
    @Override
    public boolean autoLogout() {
        // Implementation
    }
```
```java
{  
    Date date = new Date();
    Calendar cal = GregorianCalendar.getInstance();
    cal.setTime(date);
    int hourOfDay = cal.get(Calendar.HOUR_OF_DAY);
    if (hourOfDay >= 9 && hourOfDay <= 17) {
        return false;
    }
    return true;
}
```

**Parent elements**

This element is a child of the `AUTO_LOGOUT` (for JGAS) on page 64 element.

**COMMAND (for auto logout)**

This is an element of `AUTO_LOGOUT` that provides a mechanism for the JGAS to override an application's auto logout.

**Syntax**

```
<COMMAND Timeout="seconds">command-script</COMMAND>
```

1. `seconds` is the number of seconds as the allowed time for a command to run.
2. `command-script` is the name of the script or command to be run.

**Child elements**

There are no child elements.

**Usage**

A typical use of the `COMMAND` option might be to check when the auto logout process is allowed. 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**

```
<COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND>
```

The command determines whether the auto logout takes place based on the value of an exit code it returns, or whether it times out. This is determined by the following outcome:

1. If the command's exit code is zero, the auto logout is ignored and the application keeps running.
2. If the command's exit code is not zero, the auto logout is performed.

   **Note:** If the command times out, the auto logout process is also performed the same as if the exit code was not zero.

**Parent elements**

This element is a child of the following:

- `AUTO_LOGOUT` (for JGAS) on page 64

**Related concepts**

TIMEOUT (for auto logout) on page 105
This **TIMEOUT** element sets a timeout to handle auto logout.

**COMMAND (under EXECUTION for JGAS)**

This is an optional JGAS configuration element that allows you to specify in a single command the DVM, application module, and parameters required to start an application.

**Syntax**

```xml
<COMMAND>command</COMMAND>
```

1. The *command* specifies the DVM, application module, and parameters required to start an application.

**Usage**

You use this element if you need to specify in a single command the DVM, application module, and parameters required to start an application.

If **COMMAND** is defined:

- The **DVM** on page 74 and **MODULE** on page 89 elements are ignored.
- If the **PARAMETERS** on page 90 element is defined, parameters listed will be appended to the defined command.

**Usage example**

```xml
<COMMAND>$(res.fgldir)/bin/fglrun MyApplication.42r Parameter1 parameter2</COMMAND>
```

**Parent elements**

This element is a child of the **EXECUTION (under APPLICATION for JGAS)** on page 77 element.

**CONFIGURATION (for JGAS)**

This element is the starting point for the JGAS configuration.

**Syntax**

```xml
<CONFIGURATION>...
RESOURCES>...
INTERFACE>...
LOG>...
APPLICATION>...
SERVICE>...
</CONFIGURATION>
```

**Child elements**

The **CONFIGURATION** element contains the following child elements:

- One **RESOURCES (for JGAS)** on page 98 element.
- One **INTERFACE (for JGAS)** on page 84 element.
- One **LOG (for JGAS)** on page 86 element.
- One **APPLICATION (for JGAS)** on page 62 element.
- One **SERVICE (for JGAS)** on page 100 element.
Usage

The CONFIGURATION element is the root element in the JGAS configuration file.

Example JGAS as.xcf

```xml
<CONFIGURATION>
  <RESOURCES>
    <RESOURCE Id="res.fgldir" Source="INTERNAL">$(FGLDIR)</RESOURCE>
    <RESOURCE Id="FGLDIR" Source="ENVIRON">FGLDIR</RESOURCE>
    ...
  </RESOURCES>
  <INTERFACE>
    <ROOT_URL_PREFIX>$(res.rooturlprefix)</ROOT_URL_PREFIX>
    <REPORT_VIEWER_DIRECTORY>$(res.gredir)/viewer</REPORT_VIEWER_DIRECTORY>
    ...
  </INTERFACE>
  <LOG>
    <CONSOLE Enabled="$(log.console.enabled)"
      MaxLength="$(log.console.maxlength)">
      <FORMAT Type="$(log.console.format)">$(log.console.columns)</FORMAT>
      <CATEGORIES>$(log.console.categories)</CATEGORIES>
    </CONSOLE>
    <DAILYFILE Enabled="$(log.file.enabled)"
      Directory="$(log.file.path)"
      MaxLength="$(log.file.maxlength)">
      <FORMAT>
        <CATEGORIES>
      </DAILYFILE>
  </LOG>

  <APPLICATION>
    <EXECUTION>
      <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
      ...
      <WEB_COMPONENT_DIRECTORY>webcomponents</WEB_COMPONENT_DIRECTORY>
      <UA_OUTPUT>
        <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
        <TIMEOUT>
          ...
        </TIMEOUT>
        <GBC>$(res.gbc)</GBC>
        <GDC_SHORTCUT>$(res.deployment.root)/tpl/shortcut/gdc-http.gdc</GDC_SHORTCUT>
      </UA_OUTPUT>
    </EXECUTION>
    <AUTO_LOGOUT>
      <TIMEOUT>$(app.timeout.autologout)</TIMEOUT>
    </AUTO_LOGOUT>
  </APPLICATION>

  <SERVICE>
    <EXECUTION>
      <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
      ...
      <DVM>$(res.dvm.wa)</DVM>
      <POOL>
        <START>$(ws.pool.start)</START>
        <MIN_AVAILABLE>$(ws.pool.min)</MIN_AVAILABLE>
      </POOL>
    </EXECUTION>
  </SERVICE>
</CONFIGURATION>
```
<MAXAVAILABLE>${ws.pool.max}</MAXAVAILABLE>
</POOL>
</EXECUTION>
<TIMEOUT>
<!--REQUEST_RESULT>${ws.timeout.requestresult}</REQUEST_RESULT-->
<DVMAVAILABLE>${ws.timeout.dvmavailable}</DVMAVAILABLE>
<KEEPALIVE>${ws.timeout.keepalive}</KEEPALIVE>
</TIMEOUT>
</SERVICE>
</CONFIGURATION>

**Parent elements**

It has no parent element.

**CONSOLE (under LOG for JGAS)**

This element defines the log messages sent to the console.

**Syntax**

```xml
<CONSOLE Enabled="TRUE|FALSE" MaxLength="max">
    <FORMAT Type="${log.console.format}">
        ${log.console.columns}
    </FORMAT>
    <CATEGORIES>${log.console.categories}</CATEGORIES>
</CONSOLE>
```

1. The `Enabled` attribute can be set with a value of "TRUE" or "FALSE". In the JGAS configuration file the default is `TRUE`, specified by the resource `log.console.enabled`
2. `max` specifies the number of characters as the maximum size of a single log message. The default is 160 characters, set by the resource `log.console.maxlength`.

**Child elements**

The `CONSOLE` element may contain the following child elements:

1. One `CATEGORIES (under LOG for JGAS)` on page 66 element.
2. One `FORMAT (under LOG for JGAS)` on page 78 element.

**Usage**

Use this element to specify the output of log messages to the standard output.

**Example**

```xml
<CONSOLE Enabled="${log.console.enabled}" MaxLength="${log.console.maxlength}"
    <FORMAT Type="${log.console.format}">${log.console.columns}</FORMAT>
    <CATEGORIES>${log.console.categories}</CATEGORIES>
</CONSOLE>
```

In this example, log messages are enabled for the standard output.

**Parent elements**

This element is a child of the `LOG (for JGAS)` on page 86 element.
**DAILYFILE (under LOG for JGAS)**

The **DAILYFILE** element specifies the log messages that are generated and written to the JGAS log files.

**Syntax**

```
<DAILYFILE Enabled="[TRUE|FALSE]" Directory="log-file-path"
    MaxLength="max"></DAILYFILE>
```

1. The **Enabled** attribute can be set with a value of "TRUE" or "FALSE". In the JGAS configuration file the default is TRUE, specified by the resource, log.file.enabled.
2. The **log-file-path** specifies where log files are written. The default is set by the resource log.file.path.
3. **max** specifies the number of characters as the maximum size of a single log message. There is no limit set by default. The resource log.file.maxlength, which defines the default is set to -1, which means no limit is set.

**Child elements**

The **DAILYFILE** element may contain the following child elements:

1. One **CATEGORIES** (under LOG for JGAS) on page 66 element.
2. One **FORMAT** (under LOG for JGAS) on page 78 element.

**Usage**

Use this element to configure the output of logs to the daily log file.

**Example**

```
<DAILYFILE Enabled="TRUE"
    Directory="$(log.file.path)"
    MaxLength="$(log.file.maxlength)">
    <FORMAT Type="$(log.file.format)">$(log.file.columns)</FORMAT>
    <CATEGORIES>$(log.file.categories)</CATEGORIES>
</DAILYFILE>
```

In this example, log messages are enabled and written to logs specified in the log file path.

**Parent elements**

This element is a child of the **LOG (for JGAS)** on page 86 element.

**DELEGATE**

The **DELEGATE** element specifies the Genero REST service in charge of all starting requests for applications.

**Syntax**

```
<DELEGATE service=name>
    [</anyparameter1>any-value</anyparameter1>]
    [</anyparameter2>any-value</anyparameter2>]
</DELEGATE>
```

1. **name** specifies the 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:

```
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.

When working with a single sign-on 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.

The JGAS 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 17.

**Parent elements**

This element is a child of one of the following elements:

- EXECUTION (for a service) on page 76
- EXECUTION (under APPLICATION for JGAS) on page 77

**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>
```
**Parent elements**

This element is a child of one of the following elements:
- APPLICATION (for an application) on page 60
- APPLICATION (for a service) on page 63

**DVM_AVAILABLE**

The DVM_AVAILABLE element specifies a timeout (in seconds) that the JGAS allows for the DVM to start.

**Syntax**

```xml
<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 JGAS sends an error message to the front-end and logs the message: DVM_AVAILABLE timeout expired

```xml
<DVM_AVAILABLE>10</DVM_AVAILABLE>
```

In this usage example, the DVM available timeout is set to 10 seconds.

**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:
- TIMEOUT (for a service) on page 104
- TIMEOUT (for an application) on page 103

**DVM**

This element specifies the name of the Dynamic Virtual Machine.

**Syntax**

```xml
<DVM>name</DVM>
```

1. `name` defines the name of the DVM. This value is typically `frlrun` for Linux®/UNIX™ Systems (UNIX) and `frlrun.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

```xml
<DVM>$(res.dvm.wa)</DVM>
```

Parent elements

This element is a child of one of the following elements:

- EXECUTION (under APPLICATION for JGAS) on page 77
- EXECUTION (for a service) on page 76

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>
```

**Parent elements**

This element is a child of one of the following elements:

- EXECUTION (under APPLICATION for JGAS) on page 77
- EXECUTION (for a service) on page 76

**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 >
    [...]
    [<ENVIRONMENT_VARIABLE>...]/<ENVIRONMENT_VARIABLE>
    [...]
    [<PATH>...]</PATH>
    [<DVM>...]</DVM>
    [<MODULE>...]</MODULE>
    [<PARAMETERS>...]</PARAMETERS>
    [<ACCESS_CONTROL>...]</ACCESS_CONTROL>
    [<DELEGATE>...]</DELEGATE>
    [<POOL>...]</POOL>
</EXECUTION>
```

**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 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 setting individual execution elements specific to the application.

Usage example

```xml
<EXECUTION>
  <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="FGLLDPATH">$(res.fglldpath)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="FGLWSDEBUG">$(res.fglwsdebug)</ENVIRONMENT_VARIABLE>
  <DVM>$(res.dvm.wa)</DVM>
  <ACCESS_CONTROL>
    <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
  </ACCESS_CONTROL>
  <POOL>
    <START>$(ws.pool.start)</START>
    <MIN_AVAILABLE>$(ws.pool.min)</MIN_AVAILABLE>
    <MAX_AVAILABLE>$(ws.pool.max)</MAX_AVAILABLE>
  </POOL>
</EXECUTION>
```

Parent elements

This element is a child of APPLICATION (for a service) and SERVICE (for JGAS).

Related concepts

EXECUTION (under APPLICATION for JGAS) on page 77

This EXECUTION element sets the runtime environment and execution parameters for an application deployed with JGAS.

EXECUTION (under APPLICATION for JGAS)

This EXECUTION element sets the runtime environment and execution parameters for an application deployed with JGAS.

Syntax

```xml
<EXECUTION [ AllowUrlParameters="[TRUE|FALSE]" ]>
  ![ENVIRONMENT_VARIABLE]![ENVIRONMENT_VARIABLE]![COMMAND]![COMMAND]![PATH]![PATH]![DVM]![DVM]![MODULE]![MODULE]![PARAMETERS]![PARAMETERS>
```
1. 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**

In the JGAS the order child elements appear within the parent node is not important. The `EXECUTION` element may contain the following child elements:

- Zero to many `ENVIRONMENT_VARIABLE` elements.
- Zero or one `COMMAND` (under `EXECUTION` for JGAS) element.
- Zero or one `PATH` element.
- Zero or one `DVM` element.
- Zero or one `MODULE` element.
- Zero or one `PARAMETERS` element.
- Zero or one `UA_OUTPUT` element. This element is used for all UI applications.
- Zero or one `DELEGATE` element.
- Zero or one `WEB_COMPONENT_DIRECTORY` element.

**Usage**

You use this element to set the runtime environment and execution parameters for an application.

**Usage example**

```xml
<EXECUTION>
  <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
  <PATH>$(res.deployment.path)</PATH>
  <DVM>$(res.dvm.wa)</DVM>
  <MODULE>myapp.42m</MODULE>
  <WEB_COMPONENT_DIRECTORY>webcomponents</WEB_COMPONENT_DIRECTORY>
  <ACCESS_CONTROL>
    <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
  </ACCESS_CONTROL>
  <UA_OUTPUT>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <TIMEOUT>$(res.gbc)</TIMEOUT>
    <GDC_SHORTCUT>$(res.deployment.root)/tpl/shortcut/gdc-http.gdc</GDC_SHORTCUT>
  </UA_OUTPUT>
  <DELEGATE>
</EXECUTION>
```

**Parent elements**

This element is a child of the `APPLICATION` (for JGAS) element.

**FORMAT (under LOG for JGAS)**

`FORMAT` is the JGAS configuration element that specifies the format and details included in a log message.

**Syntax**

```xml
<FORMAT Type="TEXT|MULTILINE">field-id-list</FORMAT>
```
1. The `Type` attribute can have one of two values:
   - TEXT, a log message is formatted without line breaks in a single line (new line characters are escaped).
     The default format for log file is TEXT, as set by the `log.file.format` resource.
   - MULTILINE format type handles formatting for multi-line output. It stops the logger from escaping new line characters, and provides for indentation, for example, when the DVM AUI tree is logged.
     The default for CONSOLE is MULTILINE as specified by the `log.console.format` resource.

2. `field-id-list`, specifies one or more field identifiers that define the detail in the logged output. The fields are separated by spaces. The order of the fields determines the order of the log output. Valid field identifier types are shown in Table 10: Log message field identifiers on page 79.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date of logging the event.</td>
</tr>
<tr>
<td>time</td>
<td>Time of logging the event.</td>
</tr>
<tr>
<td>relative-time</td>
<td>Time elapsed since the dispatcher started.</td>
</tr>
<tr>
<td>thread-id</td>
<td>Identifier of the thread logging the message.</td>
</tr>
<tr>
<td>location</td>
<td>Location of the logged message in the source code; the file name and line number where the event occurred.</td>
</tr>
<tr>
<td>contexts</td>
<td>Internal data representing the successive context the process went through to reach the logged event, for example, which VM, which HTTP request, etc. This data is formatted as <code>type=value</code> pairs separated by semicolons.</td>
</tr>
<tr>
<td>event-type</td>
<td>Type of event logged.</td>
</tr>
<tr>
<td>event-params</td>
<td>Event details or message to be logged. XML fragment representing the structured information attached to the event. This XML fragment will be on a single line, when the attribute <code>Type</code> is set to TEXT.</td>
</tr>
</tbody>
</table>

**Tip:** If used, this field should be placed at the end as it can contain data with spaces.

### Usage

You use this element to specify the format and details included in a log message.

### Example

```xml
<FORMAT Type="TEXT">time event-type event-params</FORMAT>
<FORMAT Type="MULTILINE">date relative-time thread-id contexts event-type event-params</FORMAT>
```

### Reading the logged output

When reading the content of logged output:
- A log message ends with a line feed (LF) and carriage return (CR) character.
- `event-type` values are enclosed in double quotes (".
- `event-params` values may contain characters and spaces. Well-known non-printable characters are replaced by standard C escape sequences; other non-printable characters use a hexadecimal encoding.
- All other field values do not contain spaces.
- A missing value is replaced by a hyphen (-).
• 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.
• The event-params field output may contain:
   • a single line or escaped log message when the attribute Type is set to TEXT.
   • a multi-line log message when the attribute Type is set to MULTILINE.

Parent elements
This element is a child of the following elements:
• CONSOLE (under LOG for JGAS) on page 71
• DAILYFILE (under LOG for JGAS) on page 72

GBC
The GBC element specifies a customization directory for the Genero Browser Client (GBC) front-end.

Syntax

<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 JGAS searches for the dir-file-name in GBC_LOOKUP_PATH on page 81 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: customization project directory

<GBC>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 JGAS searches for the GBC directory at paths defined by the GBC_LOOKUP_PATH.

Usage example: customization file

<GBC>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.

**Parent elements**

This element is a child of UA_OUTPUT on page 105

**GBC_LOOKUP_PATH**

The GBC_LOOKUP_PATH element specifies the location of installed Genero Browser Client (GBC) front-ends.

**Syntax**

```xml
<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 80. Examples are shown of how to specify this using the GBC_LOOKUP_PATH and with dispatcher overrides.

**Example using GBC_LOOKUP_PATH**

```xml
<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 JGAS configuration file by default, however it can be set as required at runtime. See Set GBC_LOOKUP_PATH at JGAS runtime on page 82.
- 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 at JGAS runtime

In this example you override the res.path.gbc.user lookup path for GBC with the fglgar command-line utility when running applications in a war file in the JGAS.

```bash
fglar run --war hello.war -E res.path.gbc.user=<custom_gbc_path>
```

Parent elements

This element is a child of INTERFACE (for JGAS) on page 84.

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 105.

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
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 JGAS, make sure your image files have extensions.
- The "X-Frame-Options" header provides clickjacking protection by not allowing iframes to load on your site.

```
  <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>
```

Parent elements
This element is a child of the following elements:

- APPLICATION (for HTTP) on page 61
- SERVICE (for HTTP) on page 100

HTTP
The HTTP element contains elements defining the communication carried on the HTTP protocol between Web applications and the client.

Syntax
```
<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 101 element
• One `CACHE_CONTROL_MAX_AGE` on page 65 element
• One `APPLICATION (for HTTP)` on page 61 element
• One `SERVICE (for HTTP)` on page 100 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>
  ...  
  <HTTP>
    <SESSION_COOKIE Secure="TRUE"> </SESSION_COOKIE>
    <APPLICATION>
      <HEADER/>
    </APPLICATION>
    <SERVICE>
      <HEADER/>
    </SERVICE>
  </HTTP>
</INTERFACE>
```

Parent elements
This element is a child of `INTERFACE (for JGAS)` on page 84.

`INTERFACE (for JGAS)`
This element specifies the connection between the JGAS and the Web server.

Syntax

```xml
<INTERFACE>
  [...]
  <ROOT_URL_PREFIX></ROOT_URL_PREFIX> [...]
  <GBC_LOOKUP_PATH></GBC_LOOKUP_PATH>
  <REPORT_VIEWER_DIRECTORY></REPORT_VIEWER_DIRECTORY>
  [...]
  <REPORT_REMOTE_URL_PREFIX></REPORT_REMOTE_URL_PREFIX> [...]
  <HTTP> ... </HTTP>
</INTERFACE>
```

Child elements
The `INTERFACE (for JGAS)` element may contain the following child elements:

**Note:** In the JGAS the order child elements appear within the parent node is not important.

• Zero or more `ROOT_URL_PREFIX` on page 99 element.
• One `GBC_LOOKUP_PATH` on page 81 element.
• One `REPORT_VIEWER_DIRECTORY` on page 94 element.
• Zero or one `REPORT_REMOTE_URL_PREFIX` on page 94 element.
• Zero or one `HTTP` on page 83 element.

Usage
You use this element to configure the connection between the JGAS and the Web server. Its elements define the dedicated URL prefixes and the path to directories used by the Report Viewer and the Genero Browser Client (GBC).
You use it also to define the HTTP headers that configure requests sent and received between the JGAS and front-end.

**Example usage**

```xml
<INTERFACE>
  <ROOT_URL_PREFIX>${res.rooturlprefix}</ROOT_URL_PREFIX>
  <REPORT_VIEWER_DIRECTORY>${res.gredir}/viewer</REPORT_VIEWER_DIRECTORY>
  <GBC_LOOKUP_PATH>${res.gbc.lookup.path}</GBC_LOOKUP_PATH>
  <HTTP>
    <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>
```

By default JGAS will listen on the port 8080.

**Parent elements**

This element is a child of the [CONFIGURATION (for JGAS)](#) on page 69 element.

**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:

- [TIMEOUT (for a service)](#) on page 104
LONG
The LONG element contains the long description to be associated with an application definition.

Syntax

```
<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

```
<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 73 element.

Related concepts
SHORT on page 102
The SHORT element contains the short description to be associated with an application definition.

LOG (for JGAS)
The LOG element specifies details of the log files the JGAS creates and/or the messages sent to the console.

Syntax

```
<LOG>
  <CONSOLE> ... </CONSOLE>
  <DAILYFILE> ... </DAILYFILE>
</LOG>
```

Child elements
The LOG element may contain the following child elements:

1. One CONSOLE (under LOG for JGAS) on page 71 element.
2. One DAILYFILE (under LOG for JGAS) on page 72 element.

Usage
You use the LOG element and its child elements to specify details for logs.

With the LOG configuration you specify:

- Whether logging is enabled for console and/or file
- If daily log files are enabled, where these are created
- The format of the log messages
- The type or category of information logged
- The maximum size of a single log message

**Example: default JGAS as.xcf log configuration**

```xml
<LOG>
  <CONSOLE Enabled="$(log.console.enabled)"
       MaxLength="$(log.console.maxlength)">
    <FORMAT Type="$(log.console.format)">$(log.console.columns)</FORMAT>
    <CATEGORIES>$(log.console.categories)</CATEGORIES>
  </CONSOLE>
  <DAILYFILE Enabled="$(log.file.enabled)"
              Directory="$(log.file.path)"
              MaxLength="$(log.file.maxlength)">
    <FORMAT Type="$(log.file.format)">$(log.file.columns)</FORMAT>
    <CATEGORIES>$(log.file.categories)</CATEGORIES>
  </DAILYFILE>
</LOG>
```

**Parent elements**

This element is a child of the **CONFIGURATION (for JGAS)** on page 69 element.

**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 **MAX_AVAILABLE** to control the number of DVMs - and in effect the number of licenses - used by each application. You can control your license usage using **MAX_AVAILABLE**. 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 103 and **MIN_AVAILABLE** on page 88 can not exceed **MAX_AVAILABLE** DVMs.

```xml
START <= MAX_AVAILABLE
MIN_AVAILABLE <= MAX_AVAILABLE
```

**Parent elements**

This element is a child of the **POOL** on page 91 element.
**MAX_REQUESTS_PER_DVM**

The `MAX_REQUESTS_PER_DVM` element specifies the maximum number of requests a DVM can handle before being stopped by the pool manager.

**Syntax**

```xml
<MAX_REQUESTS_PER_DVM> max </MAX_REQUESTS_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 91 element.

**MIN_AVAILABLE**

The `MIN_AVAILABLE` element specifies the minimum number of available DVMs to be attached to a Web Service.

**Syntax**

```xml
<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 cannot exceed `MAX_AVAILABLE` on page 87.

```plaintext
0 <= MIN_AVAILABLE <= MAX_AVAILABLE
```

`MIN_AVAILABLE` can be either less than or greater than the value specified by `START` on page 103.

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 91 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 .jar module) you want to run. If the MODULE element is omitted, the JGAS uses the name of the requested application.

\[ <\text{MODULE}>\text{Edit} </\text{MODULE}> \]

Parent elements

This element is a child of one of the following elements:

- EXECUTION (under APPLICATION for JGAS) on page 77
- EXECUTION (for a service) on page 76

PARAMETER

The PARAMETER element specifies a parameter to provide on the DVM command line.

Syntax

\[ <\text{PARAMETER}>\text{value} </\text{PARAMETER}> \]

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

\[ <\text{PARAMETERS}> <\text{PARAMETER}>\text{Hello world!} </\text{PARAMETER}> <\text{PARAMETER}>\text{Again} </\text{PARAMETER}> </\text{PARAMETERS}> \]
Parent elements
This element is a child of PARAMETERS on page 90.

PARAMETERS
The PARAMETERS element acts as a parent container for parameters provided on the DVM command line.

Syntax

```xml
<PARAMETERS>
  [...]<PARAMETER>...</PARAMETER> [...]
</PARAMETERS>
```

Child elements
• Zero or more PARAMETER on page 89 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 (under APPLICATION for JGAS) on page 77 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:
• EXECUTION (under APPLICATION for JGAS) on page 77
• EXECUTION (for a service) on page 76
PATH (under EXECUTION)
The PATH element specifies the current working directory for the application module.

Syntax

```xml
<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 89.

```xml
<PATH>/home/appdir/sales</PATH>
```

Parent elements

This element is a child of one of the following elements:

- EXECUTION (under APPLICATION for JGAS) on page 77
- EXECUTION (for a service) on page 76

POOL

The POOL element sets the limitations regarding the number of Virtual Machines (DVMs) that are attached to a Web service.

Syntax

```xml
<POOL>
  [<START>...]</START>
  [<MIN_AVAILABLE>...]</MIN_AVAILABLE>
  [<MAX_AVAILABLE>...]</MAX_AVAILABLE>
  [<MAX_REQUESTS_PER_DVM>...]</MAX_REQUESTS_PER_DVM>
</POOL>
```

Child elements

The POOL element may contain the following child elements:

1. Zero or one START elements.
2. Zero or one MIN_AVAILABLE element.
3. Zero or one MAX_AVAILABLE element.
4. Zero or one MAX_REQUESTS_PER_DVM element.

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 JGAS starts
- The minimum number of DVMs to have alive while the JGAS is running
- The maximum number of DVMs to have alive while the JGAS 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 JGAS 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 Services Pool topic in the GAS Architecture section of the Genero Application Server User Guide.

Parent elements

This element is a child of one of the following elements:

- EXECUTION (for a service) on page 76

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 105 element.

Related concepts

- PROXY (for a service) on page 92
  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

```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 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 63 element.

### 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>
  <GBC>gwc-js</GBC>
  <TIMEOUT Using="cpn.wa.timeout"/>
</UA_OUTPUT>
...
```

If the value of the resource $\{res.public.resources\}$ is set by default as in the `as.xcf` file to "common", images are therefore sought in the `appdata/public/common` directory.

```
<RESOURCE Id="res.public.resources" Source="INTERNAL">common</RESOURCE>
```

**Usage example with path**

```
...<UA_OUTPUT>
  <PROXY/></PROXY>
```
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 105

---

**REPORT_REMOTE_URL_PREFIX**

Specifies the URL prefix of the server where the Genero Report Engine (GRE) is running.

**Syntax**

```xml
<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**

```xml
<REPORT_REMOTE_URL_PREFIX>http://remotehost:12345</REPORT_REMOTE_URL_PREFIX>
```

**Parent elements**

This element is a child of **INTERFACE (for JGAS)** on page 84.

---

**REPORT_VIEWER_DIRECTORY**

The **REPORT_VIEWER_DIRECTORY** element specifies the directory used by the Genero Web Report Viewer.

**Syntax**

```xml
<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 the Configuring Genero Report Viewer on GAS topic in the Genero Application Server User Guide.

**Usage example**

```
<REPORT_VIEWER_DIRECTORY>${res.gredir}/viewer</REPORT_VIEWER_DIRECTORY>
...
```

**Parent elements**

This element is a child of [INTERFACE (for JGAS)](#) on page 84.

**REQUEST_RESULT (for an application)**

This `REQUEST_RESULT` element specifies the number of seconds the JGAS waits for the DVM to respond to pending transactions.

**Syntax**

```
<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 JGAS 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 (see [KEEP_ALIVE](#) on page 85). 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 JGAS immediately after a response. Meanwhile, the JGAS stores data sent by the DVM for the application in its buffer, waiting for a GET request from the client. When the GET request is received by the JGAS, 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 JGAS 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 client 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.
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:

- TIMEOUT (for an application) on page 103

REQUEST_RESULT (for a service)

This REQUEST_RESULT element specifies the number of seconds the JGAS 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 JGAS, after which the JGAS 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 client 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.

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 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 never releases the DVM and will wait until DVM responds to the request.

Parent elements

This element is a child of the following:

- TIMEOUT (for a service) on page 104
RESOURCE

This RESOURCE element defines general JGAS settings that can be used in the definition of applications and in external application configuration files.

Syntax

```
<RESOURCE Id=res-id [Source={"INTERNAL"|"ENVIRON"}]>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 JGAS 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 DVM, etc., within individual application or service configurations. Resources are defined in the RESOURCES (for JGAS) on page 98 element of the JGAS 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:

- RESOURCES (for JGAS) on page 98

Related concepts

RESOURCE (for an application) on page 98
- This RESOURCE element defines a resource for an application.

RESOURCE (for a service) on page 98
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 63 element.

**Related concepts**

RESOURCE (for an application) on page 98

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 97.

**Parent elements**

This element is a child of the APPLICATION (for an application) on page 60 element.

**Related concepts**

RESOURCE (for a service) on page 98

This RESOURCE element defines a resource for a Web services application.

**RESOURCES (for JGAS)**

This element contains all RESOURCE elements defined for JGAS.

**Syntax**

```
<RESOURCES>
  <RESOURCE></RESOURCE>
  ...
</RESOURCES>
```

**Child elements**

A RESOURCES element contains one or more RESOURCE on page 97 child elements.

**Usage**

You use this element to define RESOURCE elements, which can then be referenced in your application configuration files.

**Note:**

RESOURCES is the same as the RESOURCE_LIST in the standard GAS. In the context of the JGAS there is no need for platform specific resource definitions for Windows®, UNIX®, etc. Therefore there is only one RESOURCES node.

**Example RESOURCES**

```
<RESOURCES>
  <RESOURCE Id="res.fgldir" Source="INTERNAL">$(FGLDIR)</RESOURCE>
  <RESOURCE Id="FGLDIR" Source="ENVIRON">FGLDIR</RESOURCE>
  <RESOURCE Id="res.gredir" Source="ENVIRON">GREDIR</RESOURCE>
  <RESOURCE Id="res.fglgui" Source="INTERNAL">1</RESOURCE>
  <RESOURCE Id="res.dvm.wa" Source="INTERNAL">$(res.fgldir)\bin\fglrun.exe</RESOURCE>
</RESOURCES>
```
The JGAS 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`.

**Parent elements**

This element is a child of the `CONFIGURATION (for JGAS)` on page 69 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 JGAS.

**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/java-j2eedispatch</ROOT_URL_PREFIX>
```

- Where a reverse proxy server (e.g. server A) is forwarding requests to the JGAS on `serverB`.
- Where "java-j2eedispatch" specifies the `connector.uri` part of the URI; typically this is the same as `connector.uri` acknowledged by the JGAS 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 (for JGAS)` on page 84.
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

```
<SERVICE>
  [...<HEADER>...]</HEADER>
</SERVICE>
```

Child elements

The SERVICE element may contain zero to many HEADER elements.

Usage

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

```
<INTERFACE>
  ...
  <HTTP>
    <SESSION_COOKIE Secure="true"> </SESSION_COOKIE>
    <APPLICATION>
      <HEADER/>
    </APPLICATION>
    <SERVICE>
      <HEADER/>
    </SERVICE>
  </HTTP>
</INTERFACE>
```

Parent elements

This element is a child of the HTTP element.

SERVICE (for JGAS)

This element defines default settings for Web services applications within the JGAS configuration file.

Syntax

```
<SERVICE>
  [...<EXECUTION>...]</EXECUTION>
  [...<TIMEOUT>...]</TIMEOUT>
</SERVICE>
```

Child elements

The SERVICE element may contain the following child elements:

1. Zero or one EXECUTION (for a service)
2. Zero or one TIMEOUT (for a service)
**Usage**

You use the `SERVICE` element to provide default configuration settings that any Web services application contained in a deployed war file inherit and use.

**Note:** In the JGAS there is no abstract application definition. All applications and services inherit configurations from the default templates. Only one `APPLICATION` and `SERVICE` template is allowed in the JGAS configuration file (as.xcf).

**Example JGAS as.xcf SERVICE configuration**

```xml
<SERVICE>
    <EXECUTION>
        <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
        ...
        <DVM>$(res.dvm.wa)</DVM>
    </EXECUTION>
    <POOL>
        <START>$(ws.pool.start)</START>
        <MIN_AVAILABLE>$(ws.pool.min)</MIN_AVAILABLE>
        <MAX_AVAILABLE>$(ws.pool.max)</MAX_AVAILABLE>
    </POOL>
    </EXECUTION>
    <TIMEOUT>
        <!--REQUEST_RESULT>$(ws.timeout.requestresult)</REQUEST_RESULT-->
        <DVM_AVAILABLE>$(ws.timeout.dvmavailable)</DVM_AVAILABLE>
        <KEEP_ALIVE>$(ws.timeout.keepalive)</KEEP_ALIVE>
    </TIMEOUT>
</SERVICE>
```

**Parent elements**

In the JGAS configuration file, this element is a child of the `CONFIGURATION` (for JGAS) on page 69 element.

**Related concepts**

Application and service templates on page 17

Templates define configuration and environment settings that are used by the JGAS. The JGAS configuration file (as.xcf) defines two templates, one for applications and one for services.

**SESSION_COOKIE**

The `SESSION_COOKIE` element specifies that cookies the JGAS uses to validate the session are created with the secure flag.

**Syntax**

```xml
<SESSION_COOKIE Secure="TRUE"/>
```

1. **Secure** is a mandatory attribute. Valid values are `TRUE` or `FALSE`. The default is `FALSE`. When set to `TRUE`, the cookie is restricted to secured connections (HTTPS) only.

**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. See Configure sticky Web services topic in Genero Application Server User Guide.

**Example secure session cookie**

```xml
<INTERFACE>
  ...
  <HTTP>
    <SESSION_COOKIE Secure="TRUE"> </SESSION_COOKIE>
  </HTTP>
  ...
</INTERFACE>
```

**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:

```xml
<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 on page 83](#) 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**

```xml
<DESCRIPTION>
  <SHORT>A short description of the app</SHORT>
  <LONG></LONG>
</DESCRIPTION>
```

**Parent elements**

This element is a child of the [DESCRIPTION on page 73](#) element.
Related concepts
LONG on page 86
The LONG element contains the long description to be associated with an application definition.

START
The START element specifies the number of DVMs to start for Web services.

Syntax

<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 MAX_AVAILABLE on page 87.

START <= MAX_AVAILABLE

Parent elements
This element is a child of the POOL on page 91 element.

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

Syntax

<TIMEOUT >
  [<USER_AGENT>...</USER_AGENT>]
  [<REQUEST_RESULT>...</REQUEST_RESULT>]
  [<DVM_AVAILABLE>...</DVM_AVAILABLE>]
</TIMEOUT>

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 95 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 JGAS and the client.

Timeouts are important for performance, so default values are set, and values can be configured.
Timeout settings may be defined by setting individual timeout settings specific to the application.

**Usage example using local elements**

```xml
<TIMEOUT>
  <USER_AGENT>300</USER_AGENT>
  <REQUEST_RESULT>240</REQUEST_RESULT>
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
</TIMEOUT>
```

**Parent elements**

This element is a child of the [UA_OUTPUT on page 105](#) element.

**Related concepts**

[TIMEOUT (for a service) on page 104](#)

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**

```xml
<TIMEOUT>
  [DVM_AVAILABLE>...]</DVM_AVAILABLE>
  [KEEP_ALIVE>...]</KEEP_ALIVE>
  [REQUEST_RESULT>...]</REQUEST_RESULT>
</TIMEOUT>
```

**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 96` 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 JGAS and the client.

Timeouts are important for performance, so default values are set, and values can be configured.

Timeout settings may be defined by setting individual timeout settings specific to the Web service application.

**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 63](#) element.

**Related concepts**

[TIMEOUT (for an application) on page 103](#)
This element sets timeouts for process requests and the start of the DVM.

**TIMEOUT (for auto logout)**

This `TIMEOUT` element sets a timeout to handle auto logout.

**Syntax**

```
<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**

```
<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 element:

- `AUTO_LOGOUT (for JGAS)` on page 64

**Related concepts**

`COMMAND (for auto logout)` on page 68

This is an element of `AUTO_LOGOUT` that provides a mechanism for the JGAS 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**

```
<UA_OUTPUT Using=component-id >
  <PROXY>...</PROXY>
  <PUBLIC_IMAGEPATH>...</PUBLIC_IMAGEPATH>
  [...<GBC>...</GBC>]
  [...<GDC_SHORTCUT>...</GDC_SHORTCUT>]
  [...<TIMEOUT>...</TIMEOUT>]
</UA_OUTPUT>
```

**Child elements**

The `UA_OUTPUT` element may contain the following child elements:

1. One `PROXY (for an application)` on page 92 element.
2. One `PUBLIC_IMAGEPATH` on page 93 element.
3. Zero or one GBC on page 80 element.
4. Zero or one GDC_SHORTCUT on page 82 element.
5. Zero or one TIMEOUT (for an application) on page 103 element.

Usage
You use this element to specify the configuration parameters for an application delivered by the UA proxy.

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 60 element.

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

Syntax

```xml
<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 JGAS 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 JGAS 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 JGAS 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 JGAS that the user has closed the browser. With the USER_AGENT timeout, the JGAS closes the socket to the DVM, which causes the DVM to shut down.

Usage example

```xml
<USER_AGENT>300</USER_AGENT>
```

In this example, the USER_AGENT timeout is set to 300 seconds.
**Parent elements**

This element is a child of one of the following elements:

- **TIMEOUT (for an application)** on page 103

**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
<EXECUTION AllowUrlParameters="FALSE">
  <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
  <PATH>$(res.deployment.path)</PATH>
  <DVM>$(res.dvm.wa)</DVM>
  <MODULE>myapp.42m</MODULE>
  <WEB_COMPONENT_DIRECTORY>webcomponents</WEB_COMPONENT_DIRECTORY>
  <ACCESS_CONTROL>
    <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
  </ACCESS_CONTROL>
  <UA_OUTPUT>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
  </UA_OUTPUT>
  <GBC>$(res.gbc)</GBC>
  <GDC_SHORTCUT>$(res.deployment.path)/tpl/shortcut/gdc-http.gdc</GDC_SHORTCUT>
</EXECUTION>
```

**Parent elements**

This element is a child of one of the following elements:

- **EXECUTION (under APPLICATION for JGAS)** on page 77

**Related concepts**

PUBLIC_IMAGEPATH on page 93
The PUBLIC_IMAGEPATH element defines the public resources directory used by applications.

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
application configuration 15
application templates 17
application web address 43

C
command reference
  fglrun AccessProgram 39
  fglrun ImportIdP 35
configuration
  as.xcf 55
  create application configuration 16
  JGAS configuration file 55
  JGAS configuration file hierarchy 56
configuration element
  ACCESS_CONTROL 58
  ALLOW_FROM 60
  APPLICATION (for a service) 63
  APPLICATION (for HTTP) 61
  AUTHCONTEXT 38
  CACHE_CONTROL_MAX_AGE 65
  COMMAND auto logout 68
  DELEGATE 72
  DESCRIPTION 73
  DVM 74
  DVM_AVAILABLE 74
  ENVIRONMENT_VARIABLE 75
  GBC 80
  GBC_LOOKUP_PATH 81
  GDC_SHORTCUT 82
  HEADER 82
  HTTP 83
  IDFORMAT 37
  KEEP_ALIVE 85
  LONG 86
  MAXAVAILABLE 87
  MAX_REQUESTS_PER_DVM 88
  MIN_AVAILABLE 88
  MODULE 89
  PARAMETER 89
  PARAMETERS 90
  PATH 91
  POOL 91
  PROXY for app 92
  PROXY for service 92
  PUBLIC_IMAGEPATH 93
  REPORT_REMOTE_URL_PREFIX 94
  REPORT VIEWER_DIRECTORY 94
  REQUEST_RESULT for app 95
  REQUEST_RESULT for service 96
  RESOURCE 97
  RESOURCE (for application) 98
  RESOURCE (for service) 98
  ROOT_URL_PREFIX 99
  SERVICE (for HTTP) 100
  SESSION_COOKIE 101

D
delegation
  configuring 20
  how delegation works 17
  implementation 17
  REST service
    allowing application start 23
    example 24
    user agent request 21
  use cases
    local authentication 26
    Single sign-on (SSO) authentication 26
    Web services monitoring 26

E
examples
application configuration
  access control for demos 58
  secure cookies 101
  with OpenID Connect SSO 30
  with SAML SSO 36
delegation
  approving proxy start 23
  configuration example 20
  examples of use 26
  passing parameters to proxy 23
  REST service
    delegation entry point 21
    non-delegated and delegated requests 21
    pass parameters 21
    pass URL 21
    request in delegation service 21
Single sign-on (SSO)
  OpenID Connect
    authorization program 31
    retrieve user identifier 30
  SAML
    authorization program 39
    retrieve identity attributes 38
    retrieve user identifier 38
Single sign-on (SSO)
  SAML
    execute an application 40
quick start SSO
OpenID Connect, set up on JGAS 27
SAML, set up on JGAS 33

S
Single sign-on (SSO)
OpenID Connect
Add to Web application 30
authorize user 31
configure for GAS 29
customize Google identity 29
FGLPROFILE file 32
IDP parameter
CLIENT_PUBLIC_ID parameter 30
CLIENT_SECRET_ID parameter 30
OIDC log file 32
Retrieve user identifier 30
set up on JGAS 27
SAML
Add SSO to Web application 36
authenticate user 38
authorize user 39
customize GAS 34
database 42
define the ID format 37
digital signatures 34
federation of identities 37
FGLPROFILE file 42
FGLPROFILE file entries 40
IDP parameter 36
importIdP program 35
log file 42
retrieve identity attributes 38
retrieve user identifier 38
Set up on JGAS 33
specify Identity Provider 36
Single sign-on (SSO)
SAML
authenticate user 40

Q
quick start jgas
build archive 11
build war file 12
compiling application 9
run application 13

O
override
app resources 50
gbc 8
gas autologout configuration 52
log configuration 52
webContent 8
overview 6