FOUR Js
The Power of Simplicity

Four Js Genero Application Server User Guide

Contents

Genero Application Server User Guide

Manual organization at a glance.

Genero Application Server overview on page 10	**GAS Quick Start Guide on page 15**	**GAS Basics on page 27**	**Configuring the Genero Application Server on page 57**
• What's new in Genero Application Server (GAS), v 3.00 (Maintenance Releases) on page 9 • What is the Genero Application Server? on page 10 • Standalone Genero Application Server on page 13 • Front Ends and Extensions on page 14	• Quick start guide to exploring resources on page 15 • Quick start guide for applications with UI on page 16 • Quick start guide for web services applications on page 21 • Genero demo applications on page 25	• Architecture of the Genero Application Server on page 27 • Components of the Genero Application Server on page 35 • What is auto logout? on page 37 • What is delegation? on page 37 • What is Single sign-on (SSO)? on page 37 • GAS directories on page 38 • Application environment on page 42 • Internationalization on page 43 • Application Web Address on page 49	• System Requirements on page 57 • GAS Configuration Check on page 58 • Licensing on page 61 • ISAPI Extension Installation and Web Server Configuration on page 65 • FastCGI Installation and Web Server Configuration on page 85 • Java Servlet Installation and Web Server Configuration on page 91 • Validating configuration files on page 93 • Configuring applications on GAS on page 95 • How to implement delegation on page 105 • How to implement Single sign-on (SSO) on page 114 • Compression in Genero Application Server on page 146 • Configuring development environment on page 147 • Configuring Multiple Dispatchers on page 148

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

This topic includes information about new features added for 3.00 Maintenance Releases (MRs) of the GAS and changes in existing functionality.

Important: Please read What's new in Genero Application Server, v 3.00 on page 235, for a list of features that were introduced with Genero 3.00 General Availability release.

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

Overview	Reference
Enhancements for the GWC-JS (v1.00.16): • Canvas elements are now supported. • Front calls *setvar* and *getvar* are supported for session variable management. See the *Genero Business Development Language User Guide* for more details about their usage.	See Features and limitations on page 176.

Table 2: Engine and Architecture, Version 3.00 (Maintenance Releases)

Overview	Reference
The `GWC_JS_LOOKUP_PATH` element (added as a child of `INTERFACE_TO_CONNECTOR` on page 324) allows you to configure the location of your custom GWC-JS front end.	See GWC_JS_LOOKUP_PATH on page 320
A new URI dedicated to the lookup of the GWC-JS directory. The complete format of the URI is `ua/w/$(GWC-JS)/<filename>`.	See Application URIs on page 49

Table 3: Deployment, Version 3.00 (Maintenance Releases)

Overview	Reference
The `WEB_COMPONENT_DIRECTORY` element allows for multiple paths to be specified.	See WEB_COMPONENT_DIRECTORY on page 361

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

Genero Application Server overview

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

What is the Genero Application Server?

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

Manages Communication between Front end and DVM	The GAS creates relationships between various front ends and the Dynamic Virtual Machines (DVMs) which run the applications.
Embeds a Web Server	A Web server to handle requests from the Internet is embedded in the GAS. It includes dispatcher and proxy processes. Communication between the Web server and the GAS is handled by dispatchers.
Simplifies Application Deployment	The GAS simplifies the deployment phase by taking care of the connection to the applications. For applications deployed to Web clients no software installation or configuration is needed on the client; only a browser is required to access the program.
Controls Interaction between DVM and Front ends	The GAS supports the development of Genero Business Development Language (BDL) applications on a single source code stream that can be run on both a browser or on a desktop. If the same application is delivered to either the Genero Web Client for JavaScript (GWC-JS) or the Genero Desktop Client (GDC), the GAS handles the communication with the DVM through its proxy and dispatcher components in much the same way. See the examples, GAS Role in GWC-JS Application Delivery on page 10 and GAS Role in GDC Application Delivery on page 11.
Provides Genero Web Services (GWS) for clients	The GAS can also be configured to provide Genero Web Services (GWS) for clients. GWS DVMs are managed in a pool by the GAS to provide resources to clients when requested. See the example GAS Role in GWS on page 12.

GAS Role in GWC-JS Application Delivery

The GWC-JS front end is provided as part of the GAS installation. This front end allows users to run applications from their browser. The following describes the processes involved, highlighting the role of the GAS and its components as it delivers an application to the browser.

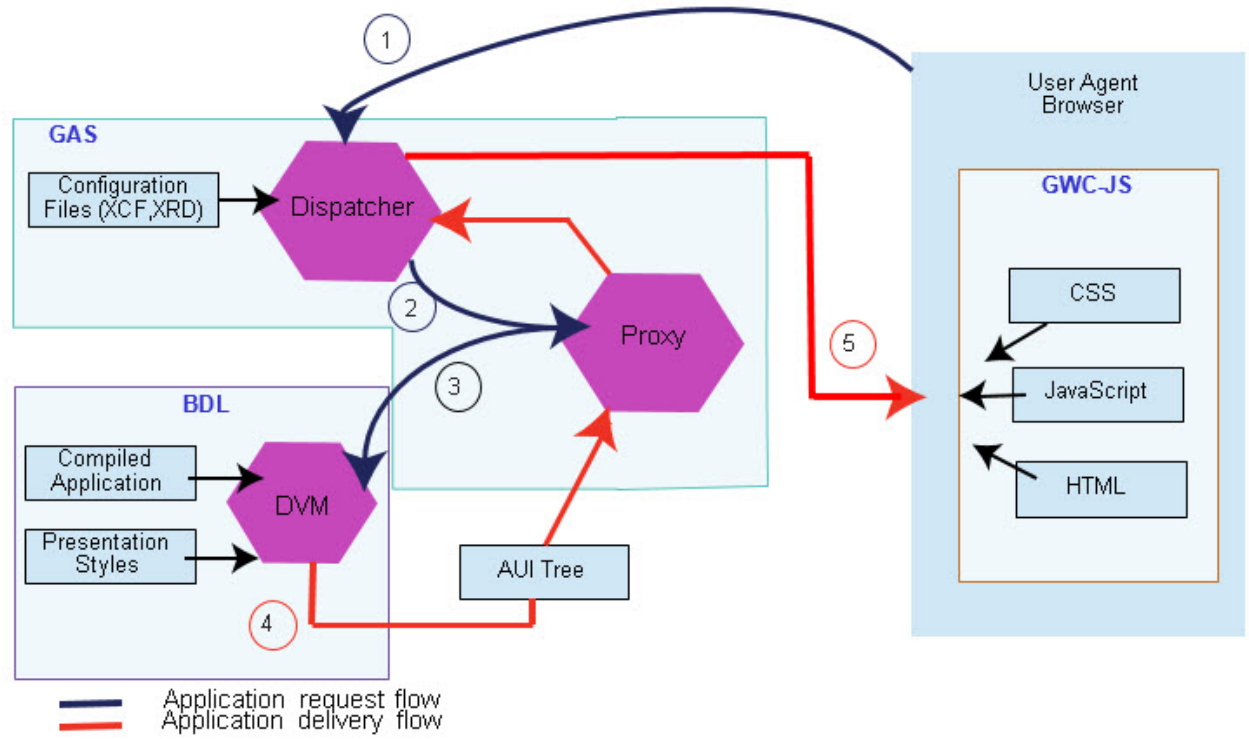


Figure 1: GWC-JS Application Delivery

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

GAS Role in GDC Application Delivery

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

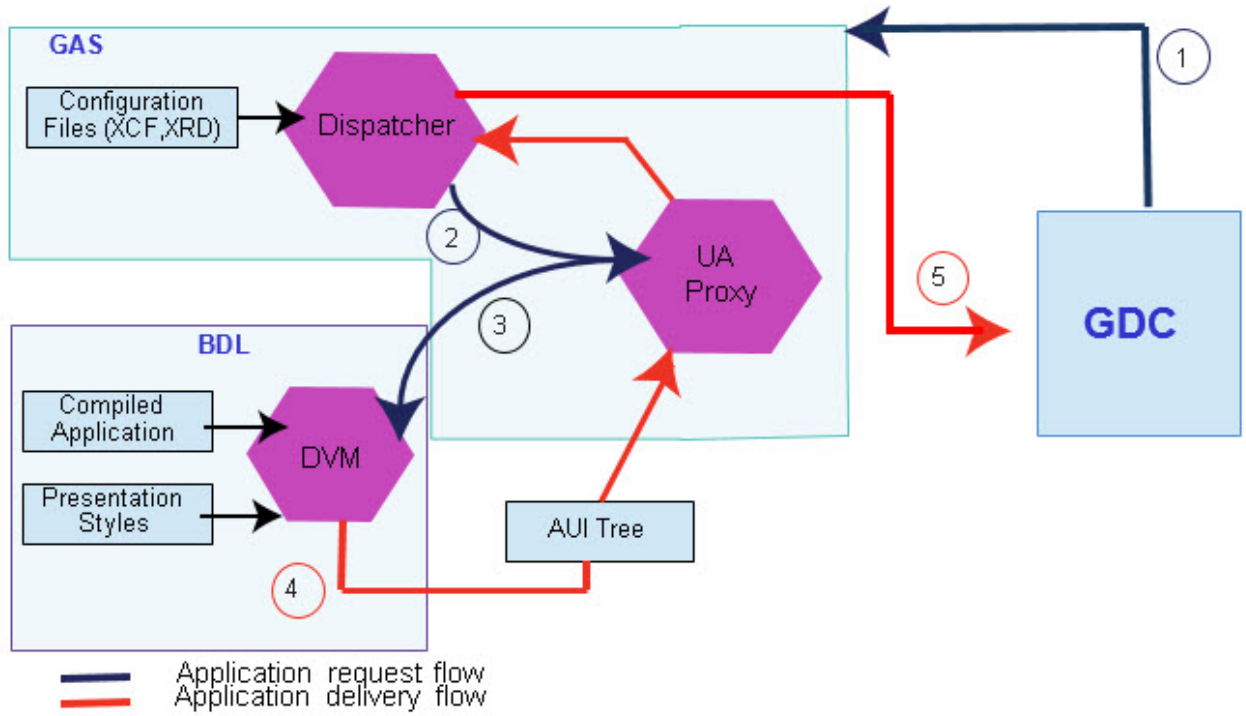


Figure 2: GDC application delivery

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

GAS Role in GWS

The GAS allows you to provide Web services to clients. Web Services configured on your GAS installation are started automatically when the GAS starts and services listen for requests from clients. For more information on Web services, see What is Web Service? on page 21.

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

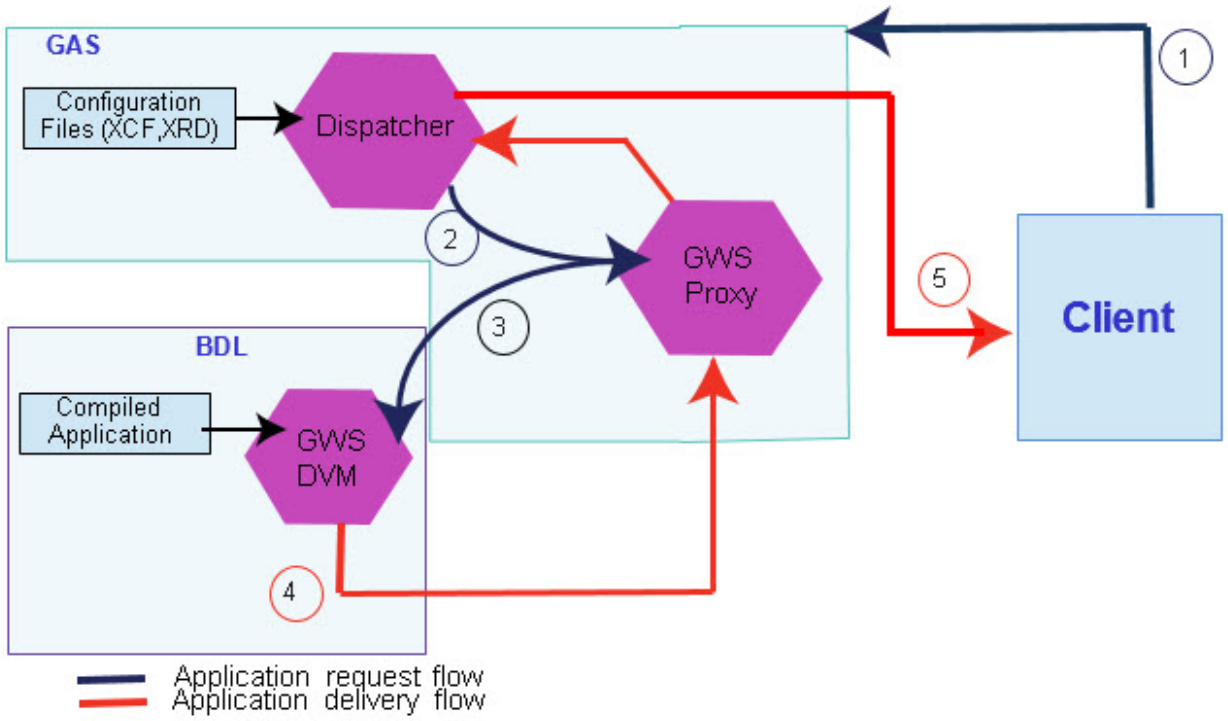


Figure 3: GWS Server/Client

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

- Standalone Genero Application Server on page 13
- What is a dispatcher? on page 35
- What is a proxy? on page 36
- Front Ends and Extensions on page 14

Standalone Genero Application Server

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

The standalone server (see Dispatcher: httpdispatch on page 263) is provided for the development cycle only, allowing you to remove the Web server from your development architecture. For production environments, a Web server is mandatory.

Front Ends and Extensions

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

Genero Desktop Client (GDC)

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

Genero Web Client for JavaScript (GWC-JS)

The GWC-JS allows you to deliver Genero applications in a Web browser on the client machine. It is a JavaScript client that works with Node technology. The Genero Web Client is provided as part of the Genero Application Server installation. For more information about the GWC-JS, see What is GWC-JS? on page 173.

Genero Web Services (GWS)

Genero Web Services allows you to implement Web services. Web services are a standard way of communicating between applications over the internet or an intranet. A web service can be a server that exposes services or a client that consumes a service. For more information, see What is Web Service? on page 21.

Genero Web Client (GWC-HTML5)

The Genero Web Client for HTML5 allows you to deliver Genero applications in a Web browser without having to install any software on the client machine. It uses browser-based themes. The Genero Web Client is provided as part of the Genero Application Server installation. This version of the GWC has been deprecated; new development should use the GWC-JS instead. For more information about the Genero Web Client, refer to the *Genero Application Server 2.50 User Guide*.

GAS Quick Start Guide

To give you an idea of what the GAS does and help you to get started, this section guides you with examples for configuring, running, and deploying basic types of applications on the GAS.

Quick start guide to exploring resources

This section provides a brief introduction to the relative path mechanism of the GAS installation and application data directories, and provides you with a quick start guide to finding files you need when working with GAS.

Paths to files and directories of your GAS installation are set by resources in the GAS configuration file,`as.xcf`, see GAS configuration file on page 289.

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

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

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

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

```
$(res.path.as)/etc/as.xcf
$(res.path.as)/bin/httpdispatch.exe
$(res.fgldir)/fglrun.exe
$(res.appdata.path)/app
$(res.appdata.path)/deployment
```

> **Note:** Installation directories may also be identified by environment variables which are set at installation time by script files, see Table 4: $FGLASDIR directories and files on page 39.

1. To find the absolute path of resources, you will need to first locate your `as.xcf` file where these predefined resources are set.

   The `as.xcf` file is an XML file which contains the default configuration for the GAS. You must search your disk for it in directories where the file is likely to be located.
2. In a text editor or with Genero Studio, open the `as.xcf` file and locate the source of your `$(res.path.as)` resource. You should find its source path amongst the `RESOURCE` list elements for your platform (e.g. WNT or UNX).

   The source path for `$(res.path.as)` is platform dependent.

   - On UNIX™, it is also represented by the environment variable *$FGLASDIR*.

- On Windows™ by the environment variable *%FGLASDIR%*.

Knowing the source of this resource, you will be able to locate the binary file of, for example the Dispatcher: httpdispatch on page 263, the standalone dispatcher used by the GAS.

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

Quick start guide for applications with UI

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

- Quick start guide to launching first application on page 16
- Quick start guide to configuring an application on page 18
- Quick start guide to running an application on page 18
- Quick start guide to deploying an application on page 20

Quick start guide to launching first application

This quick start guide provides you with the steps to launch the Genero Application Server and view demo applications delivered by the GAS to both Genero Web Client and Genero Desktop Client.

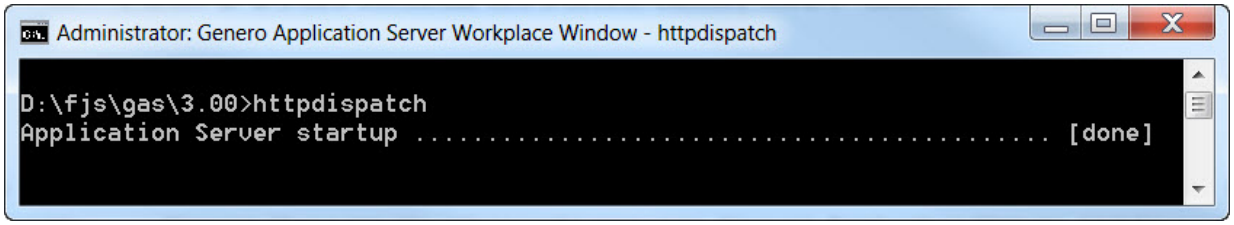
Before you begin, you must:

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

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

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

1. Start the standalone dispatcher from the command line, for details about starting `httpdispatch` see Dispatcher: httpdispatch on page 263.
   The application server (the `httpdispatch` dispatcher) is started.



2. Open the GAS demos page from a browser by entering the address, `http://localhost:6394/demos.html`.

   > **Note:** By default, access to the demos page is allowed only to localhost. The GAS configuration `ACCESS_CONTROL on page 298` element defines where access is allowed. For more information on GAS configuration see GAS configuration file on page 289.

   The Genero Application Server responds, and you should see the Genero Application Server welcome page displayed. This indicates that the GAS dispatcher is working.
3. Examine the demo application delivered by the Genero Application Server to two frontend clients; the GWC-JS and the GDC (see Front Ends and Extensions on page 14).
   a) To view the demo application displayed using the Genero Web Client, click on the **demos** link.

Alternatively, you can enter the address `http://localhost:6394/ua/r/gwc-demo`. (For more information on the application URI see Application URIs on page 49)

The application displays in the browser.

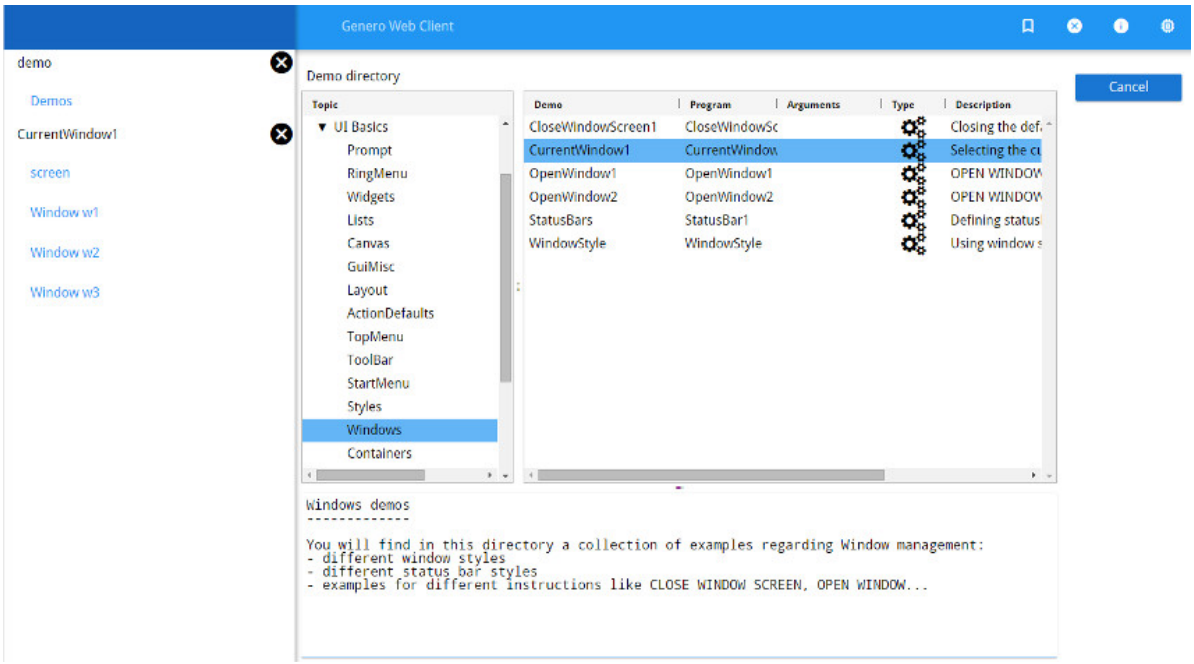


Figure 4: Demo application launched by the Genero Web Client for JavaScript

b) To view the demo application displayed using the Genero Desktop Client for HTTP, enter the address `http://localhost:6394/da/r/gdc-demo` in a new browser tab.

If asked, elect to **Open** with `gdc-demo.gdc`. For more information on GDC shortcuts see URI Examples on page 55.

The **Demos** application opens in the Genero Desktop Client.

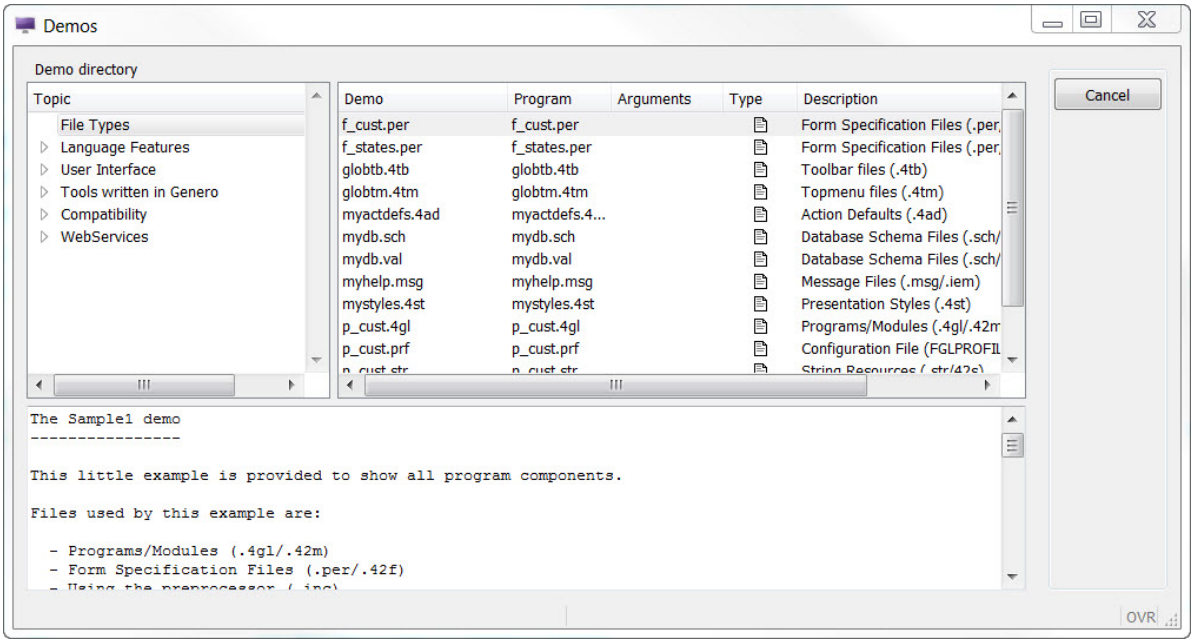


Figure 5: Demo application launched in the Genero Desktop Client for HTTP

Quick start guide to configuring an application

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

The goal of this quick start is to provide you with some basic experience in configuring information needed by the Genero Application Server to start an application. You provide these details in a separate application-specific configuration file, see Create an application configuration file on page 98 (one per application).

For the purposes of this quick start, you can create a custom configuration file for the **HelloWorld** application located in your Genero Studio's installation *$GSTDIR*/samples directory.

1. Create a new directory (e.g. you can name it "HelloWorld_config") on your disk where you will store the **HelloWorld** application source files.
2. Copy all the files from *$GSTDIR*/samples/HelloWorld directory to your new local directory.
3. Create a minimal configuration file for your **HelloWorld** application. Provide an absolute path to the location of your compiled application files in the PATH element, and in the MODULE element specify the module required to launch your application.

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

   > **Note:** The Parent attribute references **defaultgwc**, which provides default configuration for all GWC applications (see GAS configuration file on page 289).

   ```
   <?xml version="1.0" encoding="UTF-8" ?>
   <APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/
   gas/3.00/cfextwa.xsd">
    <EXECUTION>
     <PATH><path_to_your_local_directory></PATH>
     <MODULE>HelloWorld.42r</MODULE>
    </EXECUTION>
   </APPLICATION>
   ```
4. Name your file with the same name as the application (this is not mandatory but it may help you identify the file), e.g. HelloWorld, with the xcf extension.
5. Save the configuration file in your $(res.appdata.path)/app directory.

   The default directory for external application configuration files is the $(res.appdata.path)/app directory, see Quick start guide to exploring resources on page 15. The default directory resource is set in the GAS configuration file on page 289 file's GROUP element:

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

   You have successfully configured an application.

What to do next

When you have completed the above steps, your next task is to test your application to see if it is configured correctly as detailed in Quick start guide to running an application on page 18.

Quick start guide to running an application

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

For the purposes of this quick start, you can run the **HelloWorld** application you have already configured, see Quick start guide to configuring an application on page 18.

Before you begin, you must:

- Have the Genero product suite installed locally.

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

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

The goal of this quick start is to provide you with some basic experience of running applications in GAS using the two front-end clients, GWC-JS and GDC, and from Genero Studio.

1. Run your application on the GWC-JS front-end client.
   a) If your standalone dispatcher is not running, start it from the command line, for details about starting `httpdispatch` see Dispatcher: httpdispatch on page 263
   b) To run your **HelloWorld** application, in a browser enter the application address: `http://localhost:6394/`**`ua`**`/r/HelloWorld`

      For more information on the application URI see Application URIs on page 49.

      You have launched the application in the GAS.
2. To run your application on GDC from the browser, enter the address `http://localhost:6394/`**`da`**`/r/HelloWorld` in a browser tab.

   If asked, elect to **Open** gdc-demo.gdc. For more information on GDC shortcuts see URI Examples on page 55.

   The **HelloWorld** application opens in the Genero Desktop Client.
3. To run your application from within Genero Studio, complete the following sub-steps:
   a) If your standalone dispatcher is running, close your open applications and shut down the dispatcher. To shut down the standalone dispatcher, see Dispatcher: httpdispatch on page 263.

      You do not need to start the GAS dispatcher if you are running an application from within Genero Studio.
   b) In Genero Studio, the combobox in the bottom right corner of the main window displays the currently active configuration. Make sure the selected option is ***<GAS version>* Desktop**.
   c) Select the **HelloWorld** application from the **Projects** panel.

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

      The **HelloWorld** project opens in the **Projects** panel.
   d) Run your **HelloWorld** application.

      Select `Debug >> Execute`.

      The application opens in a GDC window.
4. To run your application from within Genero Studio to the GDC using HTTP, complete the following sub-steps:
   a) In Genero Studio, the combobox in the bottom right corner of the main window displays the currently active configuration. Select the ***<GAS version>* (GWC)** configuration option instead of ***<GAS version>* Desktop** to have Studio launch the application for GWC client.
   b) Run your **HelloWorld** application.

      Select `Debug >> Execute`.

      The application opens in a browser.

   You have successfully run an application.

What to do next

When you have completed the above steps, your next task is to deploy your application as detailed in Quick start guide to deploying an application on page 20.

Quick start guide to deploying an application

When you have your application configured correctly and running on the GAS, you are now ready to package the files required to deploy it as an application.

The goal of this quick start is to provide you with some basic experience in packaging an application for deployment so that it can be launched in GAS installations on other hosts. The example in this topic provides you with steps to configure and deploy an application that you can test on your own machine.

For the purposes of this quick start, you can use the configuration files created for the **HelloWorld.xcf** application, see Quick start guide to configuring an application on page 18.

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

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

2. Create a new directory where you will archive the application's source files (you can name it, for example, "helloworld_deploy").
3. Copy all the **HelloWorld** application source files from `$GSTDIR/samples` to the archive directory.

   Note: If you are deploying resources (e.g. images or Web components) with your application, these need to go in dedicated directories in the archive. For details about building an archive with public resources, please see Quick start: Genero Archive on page 220.
4. Copy the updated application configuration file (e.g. **HelloWorld.xcf**) to the directory with the application source files.
5. In the same directory, create a `MANIFEST` file (see The MANIFEST file on page 224) and save it with the name "MANIFEST" (without extension).

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

6. Create an archive (`gar`) file to deploy your application by performing the following steps:
   a) From the command line, navigate to the directory that contains the application source files AND the `MANIFEST` file.
   b) Enter the command: `fglgar --gar`.

      A Genero archive file (`gar`) is created in your current directory that has the same name as the directory. See The fglgar command on page 267.

Deploy your application on your machine

About this task:

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

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

   To deploy an archive named `HelloWorld_deploy.gar`:

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

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

To list all deployed archives:

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

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

```
gasadmin --enable-archive HelloWorld_deploy
```

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

Run the deployed application

About this task:

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

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

Quick start guide for web services applications

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

What is Web Service?

Web service is an interface where data is exchanged between applications instead of users. You can add web services to your applications to offer specific functionality to users.

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

Examples of web services that you may be familiar with are those providing weather or news updates that you can use on your site or application; you can see an example of this type of web service in Genero's **RSS** demo application. See Genero demo applications on page 25. For more information on Web service, refer to the *Genero Business Development Language User Guide*.

Quick start guide to exploring Genero Web service (server side)

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

The goal of this quick start is to provide you with some basic experience in using the GAS's GWS server to start a web service and launch an application to use the service.

1. Start the standalone dispatcher from the command line, for details about starting `httpdispatch` see Dispatcher: httpdispatch on page 263.
   The application server (the `httpdispatch` dispatcher) is started.

2. To show for example that the Web service called **Calculator** is working, you can retrieve its Web Service Description Location (WSDL)

   Enter the address `http://localhost:6394/ws/r/demo/Calculator?WSDL` in a browser

   > **Note:** The WSDL provides you with details such as the address location of the service. For more information on WSDL, please see the *Genero Business Development Language User Guide*.

```
<?xml version="1.0" encoding="UTF-8" ?>
-<wsdl:definitions targetNamespace="http://tempuri.org/" name="Calculator"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://
schemas.xmlsoap.org/wsdl/soap12/" xmlns:soap="http://schemas.xmlsoap.org/
wsdl/soap/" xmlns:fjs="http://tempuri.org/" xmlns:wsdl="http://
schemas.xmlsoap.org/wsdl/">
...
 <wsdl:service name="Calculator">
  <wsdl:port name="CalculatorPortType" binding="fjs:CalculatorBinding">
    <soap:address location="http://localhost:6394/ws/r/demo/Calculator"/>
  </wsdl:port>
  <wsdl:port name="CalculatorPortTypeSoap12"
 binding="fjs:CalculatorBindingSoap12">
    <soap12:address location="http://localhost:6394/ws/r/demo/Calculator"/
>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

3. To show for example that the Web service called **Calculator** is working, you can also start an application that uses it by performing the following steps:

   a) Now that the GWS has started the Web service, return to the **Demos** tab of your browser. In the **Topic** tree, navigate to **WebServices >> Calculator >> Client**. Double-click on the demo **Calculator Soap 1.2**.
   The **Web Services URL** dialog appears. From the drop down menu select the URL stating `http://localhost:6394/ws/r/demo/Calculator`.

   b) In the **Web Services URL** dialog, click **OK**.
   You should see a calculator screen and be able to interact with it.

Quick start guide to configuring a Web service

Before a client application can use your Web service, you need to create a configuration file with the details the Genero Application Server needs to deliver the Web service to client applications.

The goal of this quick start is to provide you with some basic experience in configuring information needed by the Genero Application Server to deliver Web services to client applications. You provide the necessary details in a separate configuration file, see Configuring applications for Web service on page 101, very similar to that used for applications.

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

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

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

Note: The `Parent` attribute reference to **ws.default** provides default configuration for all applications of the Web service type (see GAS configuration file on page 289).

```
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="ws.default" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/
gas/3.00/cfextws.xsd">
 <EXECUTION>
   <PATH><path_to_your_local_directory></PATH>
   <MODULE>CalculatorServer.42r</MODULE>
 </EXECUTION>
</APPLICATION>
```

4. Save your Web service configuration file, e.g. mycalculator.xcf, in the directory *`$(res.path.services)`*

   The default `services` directory is set by the *res.path.services* resource in the GAS configuration file on page 289 file:

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

Running a client application on Web service using GWC

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

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

For the purposes of this quick start, you can run the **mycalculator** web service you have already configured, see Quick start guide to configuring a Web service on page 22 and use the calculator client application available from the the Genero Application Server demos page, see Display demo applications with the Genero Web Client on page 25, to interact with it.

The goal of this quick start is to provide you with some basic experience in starting a Web service and running and testing a client application on the GWC front end, which you can use for testing your Web services and client applications. For production, you will use the GAS.

1. Start the standalone dispatcher from the command line, for details about starting `httpdispatch` see Dispatcher: httpdispatch on page 263.
2. To check that the **mycalculator** web service is reachable, you can retrieve its WSDL. To perform this, enter the address of the Web service application, e.g. `http://localhost:6394/ws/r/mycalculator?WSDL`, in a browser tab.
3. To interact with the **mycalculator service** and show it is working, open another browser tab and enter the address of the demo applications, `http://localhost:6394/ua/r/gwc-demo`.
4. In the **Topic** tree, navigate to **WebServices >> Calculator >> Client**. Double-click on the demo **Calculator Soap 1.2**.
   The **Web Services URL** dialog appears. From the drop down menu select the option **(Customize) Click here to add your own URL** and provide the URL to your configured web service, e.g. `http://localhost:6394/ws/r/mycalculator`.
5. In the **Web Services URL** dialog, click **OK**.
   You should see a calculator screen and be able to interact with it.

Quick start guide to deploying a Web service

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

The goal of this quick start is to provide you with some basic experience in deploying Web services. It provides you with steps to configure and deploy a web service on your own machine that you can easily adapt for deployment on GAS installation on other hosts.

For the purposes of this quick start, you can use the configuration files created for the **Calculator** Web service , see Quick start guide to configuring a Web service on page 22.

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

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

2. Create a new directory where you will archive the Web service source files (e.g. you can name it "calculator_deploy") and in it create a sub-directory called "server" .
3. Copy the application source files to the archive directory as follows:
   a) Copy all the files from *$FGLASDIR*/demo/WebServices/calculator/server to the server subdirectory in your archive directory
   b) Copy the updated Web service configuration file, **Calculator.xcf**, to the server subdirectory.
4. Create a MANIFEST file and save it with the name "MANIFEST" (without extension) to the directory that contains the application source files. For more information see The MANIFEST file on page 224.

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

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

Test the deployment on your standalone dispatcher.

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

   To deploy an archive named calculator_deploy.gar:

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

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

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

   To list all deployed archives:

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

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

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

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

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

8. Start the standalone dispatcher from the command line, for details about starting `httpdispatch` see Dispatcher: httpdispatch on page 263.
9. To start the **Calculator** web service, in a browser tab enter the address of the web service application, e.g. `http://localhost:6394/ws/r/Calculator`
10. To interact with the **Calculator service** and show it is working, open another browser tab and enter the address of the demo applications, `http://localhost:6394/ua/r/gwc-demo`.
11. In the **Topic** tree, navigate to **WebServices >> Calculator >> Client**. Double-click on the demo **Calculator Soap 1.2**.
    The **Web Services URL** dialog appears. From the drop down menu select the URL stating `http://localhost:6394/ws/r/demo/Calculator` or select the option **(Customize) Click here to add your own URL** and provide the url to your configured web service.
12. In the **Web Services URL** dialog, click **OK**.
    You should see a calculator screen and be able to interact with it. You have successfully deployed a Web service.

Genero demo applications

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

Find the demo applications

A variety of demo applications have been provided to showcase Genero features.

The demo applications for the Genero Application Server are provided as part of the Genero Business Development Language with Web Services installation.

Demos included with Genero Studio

We have demos that are bundled with Genero Studio. Access the demo applications from the **Tutorials & Samples** tab.

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

Demos bundled with Genero BDL

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

Display demo applications with the Genero Web Client

The Genero Application Server displays the demos application. The demos application is restricted to localhost by default.

Accessing the demos page

The demos application is defined in the Genero Application Server configuration file with an `Id` of `gwc-demo`

```
<!--Sample application for GWC-->
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
```

```
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
      <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```

Note:

To avoid having this application working on production sites, the default configuration restricts access to the localhost (127.0.0.1). If you want to enable it for other client machines / IP addresses, you must customize the `ALLOW_FROM` tag or remove the `ACCESS_CONTROL` tag.

To access the Genero Application Server `gwc-demo` application, you can use the Genero Web Client user interface, see User interface: home page on page 180 or you can enter this URL:

```
http://localhost:6394/demos.html
```

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

The demos application can also be accessed directly

You can access the demos application directly by entering the following URL: `http://localhost:6394/ua/r/gwc-demo`

GAS Basics

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

Architecture of the Genero Application Server

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

Architecture overview

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

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

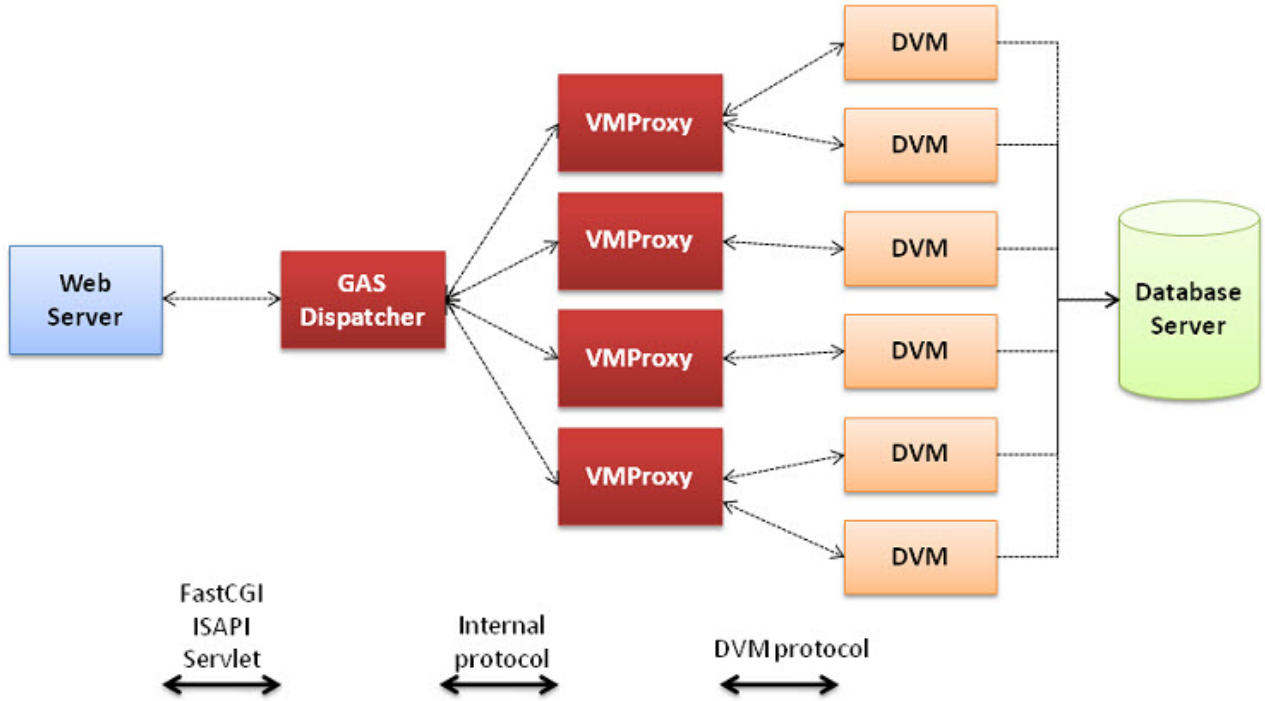


Figure 6: GAS with VMProxy Architecture

Components

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

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

How it works: the high-level overview

1. In order to request an application, the end-user enters a URI that specifies which application to launch (based upon the GAS configuration file and application configuration files). For example, the alias to serve up the GWC demo application via a web server would be `http://mywebserver/gas/ua/r/gwc-demo`. In development environments, it is possible to exclude the Web Server. For more information, see Architecture for Development (Standalone GAS).
2. The Web Server routes the request to the GAS dispatcher. GAS dispatchers refer to the connectors in charge of dispatching a GAS request to the appropriate proxy. There are different GAS dispatchers, each designed for a specific Web Server. For example, the fastcgidispatch.exe is for use with FASTCGI-compliant Web Servers such as Apache, while the isapidispatch.dll is for the Information Internet Services (IIS) Web Server.
3. The GAS Dispatcher starts the VMProxy to handle the request. Each session requesting an application results in a VMProxy starting up; as a result, you will likely see multiple proxies running concurrently. The type of proxy started (uaproxy or gwsproxy) will depend on the application being requested. The dispatcher will route to the correct proxy. The dispatcher tracks the session and proxy information in a persistent session table. The presence of this information in the session table ensures that if a dispatcher is killed or restarted, the information needed to return to the proxy and running application is still present. For more information on the responsibilities of the GAS dispatcher, see GAS Dispatcher responsibilities.

4. The VMProxy then launches the DVM for the requested application. It handles any child DVMs, keeps the DVM connections up, and handles the requests and responses appropriate for the type of proxy. For more information of the VMProxy responsibilities, see VMProxy responsibilities.
5. The DVM interacts with the database server, as needed.

Reliability inherent in the architecture

The architecture of the Genero Application Server supports reliability.

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

Development architecture (standalone GAS)

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

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

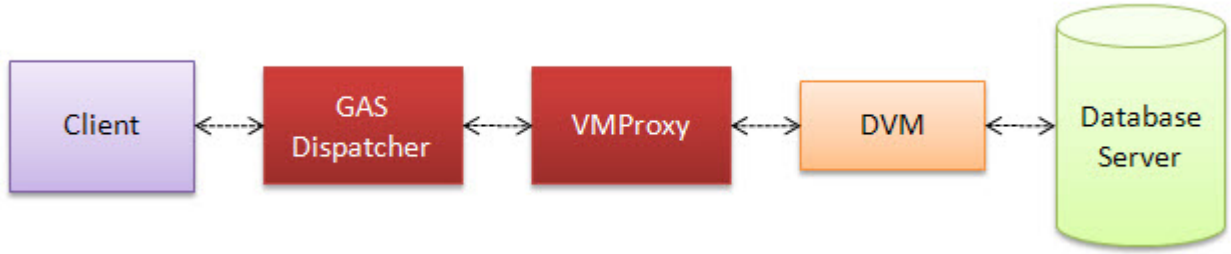


Figure 7: Architecture for stand-alone GAS

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

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

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

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

Deployment (production) architecture

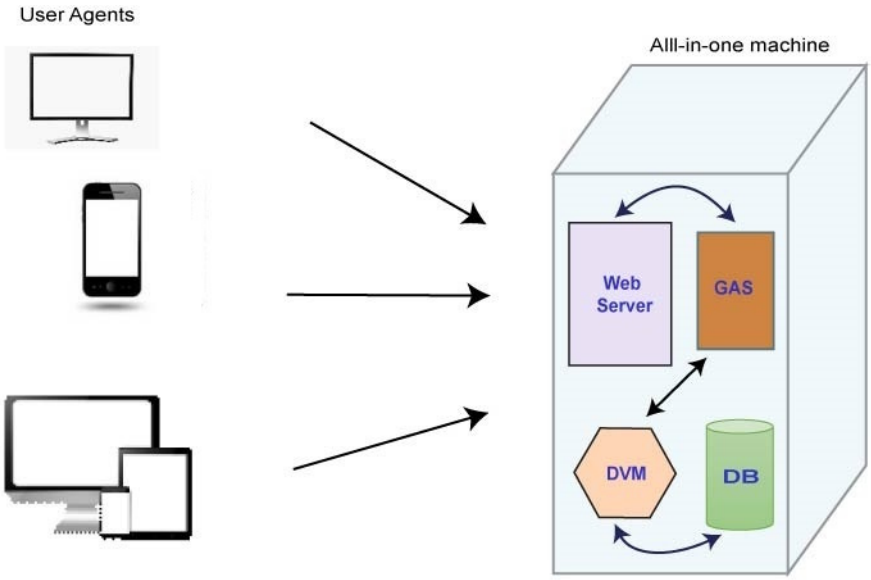
For deployment and production systems, you must include a Web server.

A Web server:

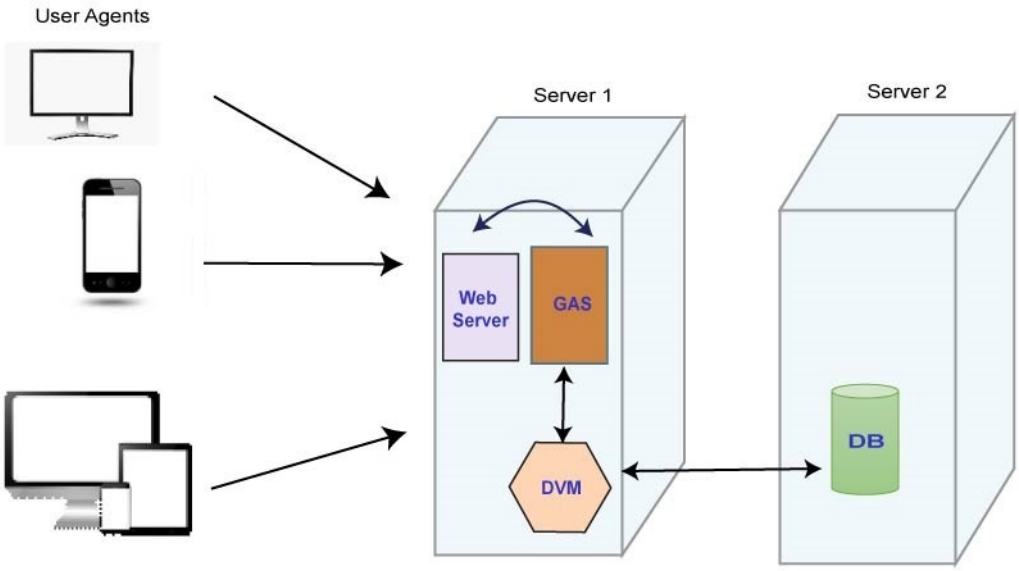
- Enables load balancing.
- Handles HTTPS.
- Handles authentication. You can use the Web server to handle authentication. Alternatively, you can use a single sign-on solution.

GAS deployment example 1: all-in-one machine

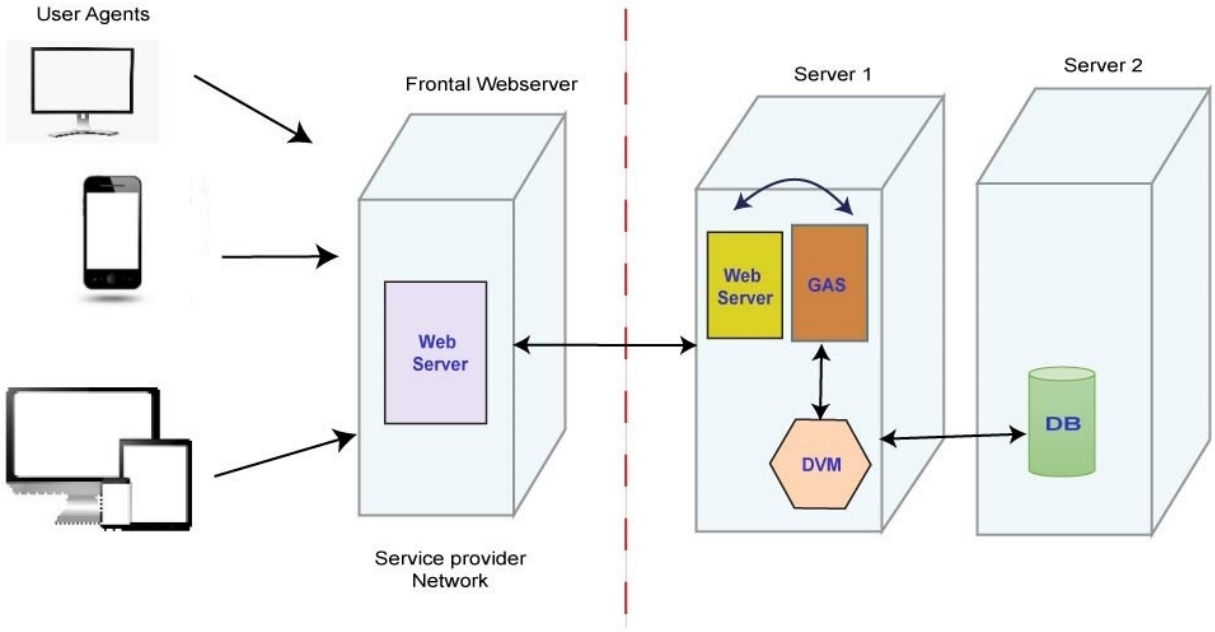
You can have the Genero Application Server on the same machine as the Web server.



GAS deployment example 2: Database on separate server



GAS deployment example 3: Behind frontal web server



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

Production deployment should be supervised by a Web specialist to avoid any security issues.

Services Pool (GWS Only)

When you define a Web service application, you specify execution parameters that determine the number of Dynamic Virtual Machines (DVMs) that can be available to service a request for that Web service. These parameters are defined in the Application Server configuration file.

Why do we need Services Pool for GWS? The main reason is to due to the way the DVM is unable to support more than one application at a time, see Reliability inherent in the architecture on page 29. The GWS proxy needs to launch new DVM (fglrun process) for each request on the Web server. If traffic to a Web server was to become very high with, for example, hundreds of requests at any period, this could potentially result in slow response time or even overloading of the server.

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

This topic details the configuration of the pool element of a Web service application and shows some examples of how the GWS proxy manages the DVM pool.

Defining the Pool Element of a Web service Application

When you define a Web service application, the `EXECUTION` element sets the runtime environment for that application by specifying the parameters for executing the Web service application. This application configuration can either reference a predefined `SERVICE_APPLICATION_EXECUTION_COMPONENT` (inheriting the runtime environment settings defined for that component) or the individual execution elements can be explicitly set for the application.

Within the `EXECUTION` element, the `POOL` element sets the limitations regarding the number of DVMs that are attached to a Web service.

- The START element specifies the number of Virtual Machines to start when the Genero Application Server starts.
- The MIN_AVAILABLE element specifies the minimum number of Virtual Machines to have alive while the Genero Application Server is running.
- The MAX_AVAILABLE element specifies the maximum number of Virtual Machines to have alive while the Genero Application Server is running.
- The MAX_REQUESTS_PER_DVM element specifies the maximum number of requests a DVM can handle before being stopped by the pool manager.

Releasing DVMs not Actively Processing Requests

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

Time to Start a New DVM	The gwsproxy waits at least the time it takes to start a new DVM before deciding if a DVM can be shutdown.
Three Times the Last Request Execution Time	If their is only one request, a DVM shutdown takes place after waiting three times the request execution time from when the last request took place.
Three Times the Average Request Frequency	If there is more than one request, a DVM is shutdown after waiting three times the average request frequency. For example, the GWS proxy calculates the elapsed time since it received the last new incoming request. If this is three times greater than the average frequency of new requests it has been receiving, it stops one inactive DVM.

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

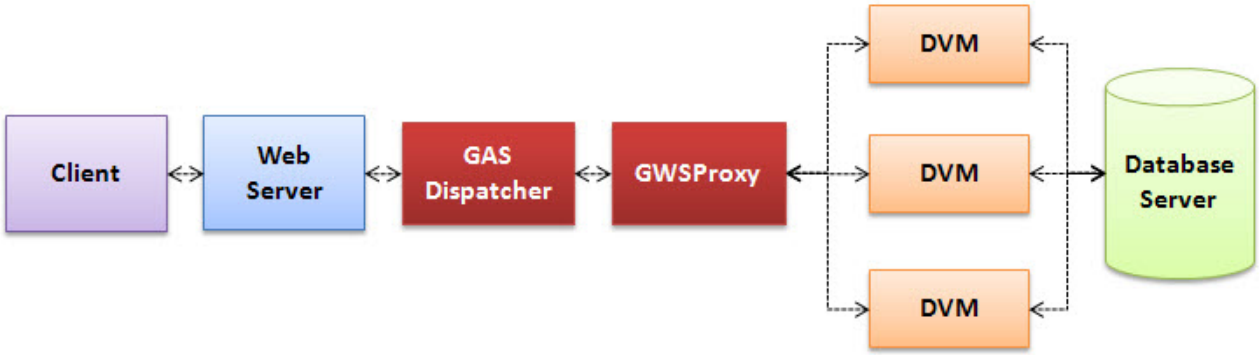
Minimum (one second)	If time to shutdown is **less than one second**, the gwsproxy waits a full one second before shutting down the DVM.
Maximum (ten minutes)	If time to shutdown is **more than ten minutes**, the gwsproxy waits no longer than ten minutes before shutting down a DVM.

Example 1: One GWSProxy Starts three DVMs

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

```
<POOL>
  <START>3</START>
  <MIN_AVAILABLE>2</MIN_AVAILABLE>
  <MAX_AVAILABLE>5</MAX_AVAILABLE>
</POOL>
```

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



Example 2: GWSProxy Releases Inactive DVMs

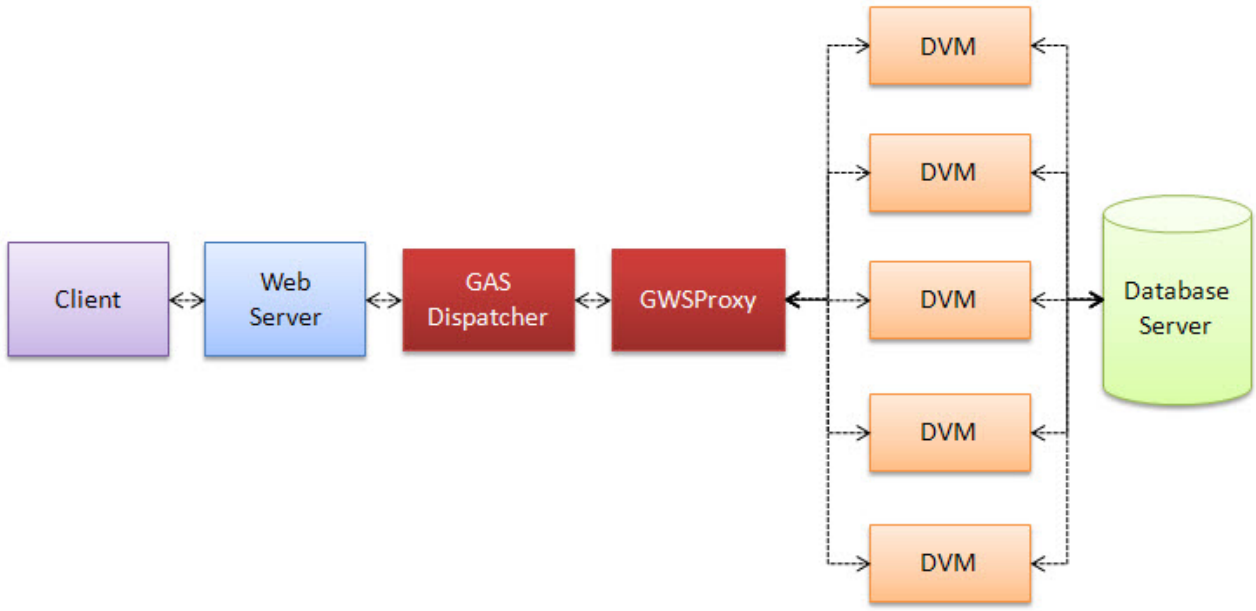
While the START element defines the number of DVMs to start initially, DVMs that are not actively processing requests can be released based on GAS statistics. For example, the GWS proxy calculates the elapsed time since it received the last new incoming request. If this is three times greater than the average frequency of new requests it has been receiving, it stops one inactive DVM. See Releasing DVMs not Actively Processing Requests on page 32.

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

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

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

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

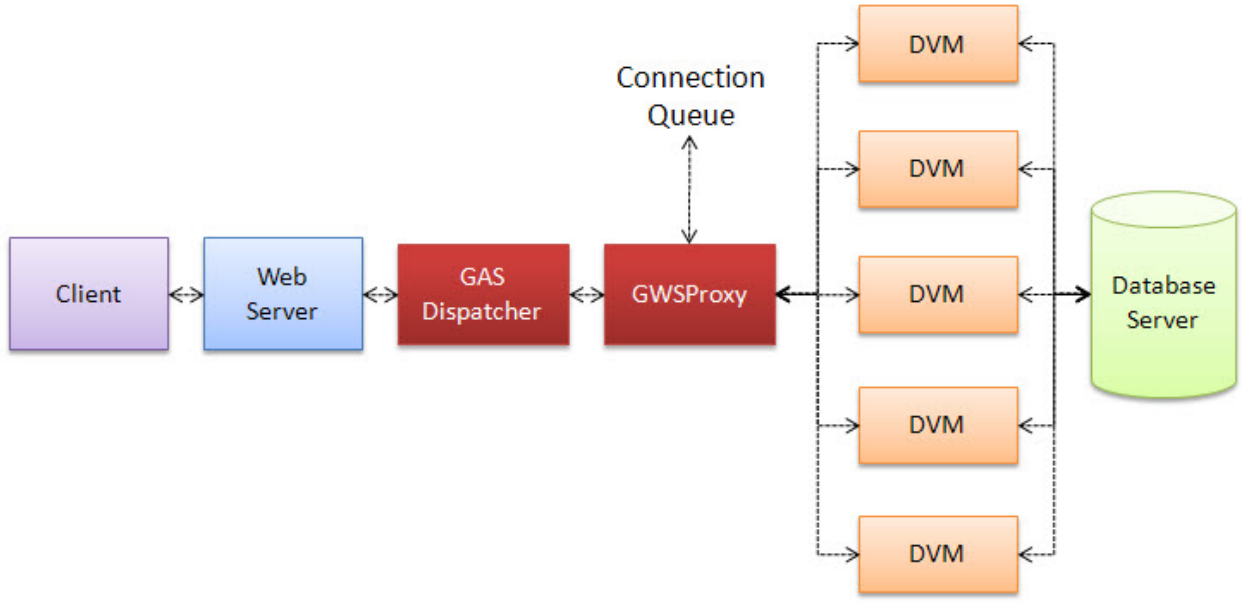


Example 4: GWSProxy Managing a Connection Queue

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

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

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



Components of the Genero Application Server

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

What is a dispatcher?

The Genero Application Server uses dispatchers specific to the web server to start proxies that handle application requests. Dispatchers play a vital role in GAS configuration integration with the web server and provide performance reliability by handling reconnection to the application in case of failure.

When an end-user enters a URI that specifies an application to launch, the Web Server routes the request to the GAS dispatcher based upon the GAS configuration file and the application configuration files.

Types of GAS Dispatchers

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

- `httpdispatch`: standalone dispatcher for development only, provided to simplify your development setup and configuration through a direct connection without a Web Server, see Dispatcher: httpdispatch on page 263
- `isapidispatch`: dispatcher for Internet Information Services (IIS)
- `fastcgidispatch`: dispatcher for Fast CGI compliant web servers like Apache, see Dispatcher: fastcgidispatch on page 265
- `java-j2eedispatch`: dispatcher for Java™ Server (like JBoss, Tomcat or WebSphere®) , see Dispatcher: java-j2eedispatch on page 266

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

GAS Dispatcher and VMProxy

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

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

GAS Dispatcher responsibilities

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

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

What is a proxy?

The Genero Application Server launches VMProxies specific to the application type. The VMProxy starts the Dynamic Virtual Machines (DVM) for the application and handles the connection, requests and responses appropriate for the type of proxy.

Types of VMProxies

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

- `uaproxy`: a universal proxy for applications using GDC and GWC-JS interfaces, see Proxy: uaproxy on page 266
- `gwsproxy`: proxy for Genero Web Service (GWS) type applications, see Proxy: gwsproxy on page 267
- `html5proxy`: proxy for the legacy Genero Web Client (GWC) type applications (now deprecated), see Proxy: html5proxy on page 267

VMProxy responsiblities

In general, a VMProxy is responsible for the following:

- Launching the DVM (see What is a DVM? on page 37)
- Handling child DVMs
- Maintaining the DVM connections

Additional responsibilities depend on the VMProxy type:

- The `uaproxy` is also responsible for the following:
  - Handling HTTP client-side front end (CSF) requests (see USER_AGENT on page 355)
  - Generating JavaScript responses
  - Handling sessions when the client is GDC or GWC-JS
  - Generating the HTML page when the client is GWC-JS
- The `gwsproxy` is also responsible for the following:
  - Handling the GWS DVM pool (see Services Pool (GWS Only) on page 31)
  - Handling HTTP Web Services requests
  - Forwarding HTTP Web Services responses (SOAP, REST, XML over HTTP)

What is a DVM?

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

What is auto logout?

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

What is delegation?

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

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

> **Note:** Delegation is required if you want to enable Single-sign-on (SSO) authentication for remote access to applications, see What is Single sign-on (SSO)? on page 37).

What is Single sign-on (SSO)?

Single sign-on allows a user to enter one name and password in order to access multiple applications.

For more information on Single Sign-on and how to implement it see How to implement Single sign-on (SSO) on page 114.

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

OpenID Connect

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

See OpenID Connect SSO on page 115.

OpenID

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

See OpenID SSO on page 120.

SAML

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

See SAML SSO on page 128

GAS directories

This section describes the relative path mechanism of the GAS installation and application data directories, and provides you with a lookup guide to file locations you need to know about when working with GAS.

The two main directories of the GAS are:

Installation directory

Where the installation files are located (identified by the environment variable `$FGLASDIR` and set by the resource `$(res.path.as)` in the GAS configuration file, see GAS configuration file on page 289).

Application directory

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

GAS has the following prerequisites for application data:

- The *appdata* directory and all of the *$FGLIMAGEPATH* (search paths for VM server image files) must be located on the same filesystem. For more details on *$FGLIMAGEPATH*, see the *Genero Business Development Language User Guide*.
- The DVM uses hard links to avoid file copies. As hard links are direct pointers to data on the disk, they can not span filesystems.

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

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

Caution: A partition A and a partition B are considered two distinct filesystems even if stored on the same physical device.

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

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

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

- Or alternatively, at the command line you can override the GAS configuration file

`$(res.appdata.path)` resource with dispatcher option `-E`. For more information on using dispatcher options, see the relevant dispatcher page: Dispatcher: httpdispatch on page 263,Dispatcher: fastcgidispatch on page 265,Dispatcher: java-j2eedispatch on page 266

Table 4: $FGLASDIR directories and files

Directory	File or Details	Description
`$FGLASDIR`		GAS installation directory.
	On UNIX™: `envas` On Windows™: `envas.bat`	Script for setting environment variables.
`$FGLASDIR/bin`		GAS dispatchers and VMProxy executables.
`$FGLASDIR/etc`	`as.xcf`	Default GAS configuration file.
`$FGLASDIR/ISAPI`	`isapidispatch.ini`	ISAPI extension configuration file.
`$FGLASDIR/lib`		Library files.
`$FGLASDIR/license`	`*.txt`	License agreement text files.
`$FGLASDIR/pic`		Default image directory.
`$FGLASDIR/release`		Licensing and release notes.
`$FGLASDIR/tpl`		`Project file for Genero Web Client for JavaScript (GWC-JS) application customization, e.g. fjs-`*`gbc-xxx`*`-build`*`123456789`*`-project.zip`. See Configuring your Environment on page 198.

`$FGLASDIR/tpl/common`
`$FGLASDIR/tpl/setHTML5`
`$FGLASDIR/tpl/shortcut`
		Contain template and snippet files for the rendering and display of HTML5 v1 type application objects in the UI. GWC-HTML5 is now deprecated.

`$FGLASDIR/war`		Java dispatcher directory.
`$FGLASDIR/web`	`demos.html`	Root directory for direct communication to the application server. The `demos.html` file provides access to Demos applications.
`$FGLASDIR/web/gwc-js`	bootstrap.html	Bootstrap mechanism for rendering default GWC-JS type applications. See Starting GWC-JS applications on page 188. Root directory to files to be used by all `GWC-JS` clients -

Directory	File or Details	Description
		images, cascading style sheets, JavaScripts, etc.

Table 5: *appdata* directories and files

Directory	File or Details	Description
appdata	On UNIX™: `$FGLASDIR/appdata` On Windows™: `C:\ProgramData\`*`vendor`*`\gas\`*`gas_version`*	GAS application data directory.
appdata`/app`	`.xcf`	Default application group; the default location for applications configuration files.
appdata`/deployment`	Within the `deployment` directory, a different directory is created for each application.	Applications deployed with Genero Archive.
appdata`/public`	**Caution:** Public images should not be placed in the `/public` root directory as the `fglrun` does not look for images to be served via the GAS there; searches start in subdirectory paths, i.e. `public/common` and `public/deployment`.	A public resource path for all applications.
appdata`/public/common`		Default *PUBLIC_IMAGEPATH* directory, a resource for common images used by applications.
appdata`/public/deployment`	Within the `/public/deployment` directory, a different directory is created for each application.	Public images deployed with applications by deployment framework (`fglgar`), see Application deployment overview on page 218
appdata`/log/`*`dispatcher_name`*`/`*`date`* *appdata*`/log/gasadmin/`*`date`*	Within the `log` directory, a different directory is created for each dispatcher. The *dispatcher_name* refers to the dispatcher: • `httpdispatch` (for the standalone GAS) • `isapidispatch` (for the ISAPI dispatcher) • `j2eedispatch` (for the Java™ dispatcher)	Log directories for dispatchers; the `gasadmin` directory is the log directory for `gasadmin` actions. **Note:** This directory is created with write permissions for users that are in the same group as the user who installed the Genero Application Server. If the user that starts the Genero Application Server (for example

Directory	File or Details	Description
	The logs for a given day are stored in a directory named for that date.	apache) is not that group, then you need to grant write permission to that user. **Note:** DVM logs are redirected to files when `DAILYFILE` is set for the log output type.
appdata/session/*dispatcher_name*	Within the `session` directory, a different directory is created for each dispatcher. The *dispatcher_name* refers to the dispatcher: • `httpdispatch` (for the standalone GAS) • `isapidispatch` (for the ISAPI dispatcher) • `j2eedispatch` (for the Java™ dispatcher) **Caution:** The `session` directory should only be used to store session files, which are required to reconnect dispatchers to proxies. It should not contain other files or subdirectories.	Persistent session table information. **Note:** This directory is created with write permissions for users that are in the same group as the user who installed the Genero Application Server. If the user that starts the Genero Application Server (for example apache) is not that group, then you need to grant write permission to that user. **Note:** *appdata*`/session/gasadmin` may exist, but it is not used by `gasadmin`.
appdata`/tmp/`*dispatcher_name* *appdata*`/tmp/gasadmin`	Within the `tmp` directory, a different directory is created for each dispatcher. The *dispatcher_name* refers to the dispatcher: • `httpdispatch` (for the standalone GAS) • `isapidispatch` (for the ISAPI dispatcher) • `j2eedispatch` (for the Java™ dispatcher)	Default file transfer directory. The `gasadmin` directory is for the result of `gasadmin` actions, such as exploring the configuration (`xcf`) files. **Note:** This directory is created with write permissions for users that are in the same group as the user who installed the Genero Application Server. If the user that starts the Genero Application Server (for example apache) is not that group, then you need to grant write permission to that user.
appdata`/services`	`*.xcf`	Default services group; the default place for web services configuration files.

Application environment

When the Genero Application Server starts an application process, it sets environment variables from various sources.

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

Environment inheritance

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

1. The environment of the dispatcher that starts the proxy.
2. The environment variables defined in the application configuration file (`ENVIRONMENT_VARIABLE` elements).
3. Additionally, some specific environment variables can be defined by the front-end; whether this is Genero Desktop Client, Genero Web Client, (see How GDC and GWC handle environment variables on page 42) or is a Genero Web Service (see How GWS handle HTTP request headers on page 42) or is a Genero Web Service using delegation, see How GWS with delegation handles HTTP request headers on page 42.

How GDC and GWC handle environment variables

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

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

The following exceptions are supported for backward compatibility:

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

How GWS handle HTTP request headers

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

`X-FourJs-Environment-Parameter-`*`name`*

Where *name* is the header name. To get the header value, you must use the `com.HTTPRequest.getRequestHeader()` method. For example:

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

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

> **Note:** For more information on `com.HTTPRequest` methods, please see the *Genero Business Development Language User Guide*.

How GWS with delegation handles HTTP request headers

When delegation is used in Genero Web Services, the `X-FourJs-Environment-Parameter-<`*`parameter_name`*`>` is used by the delegate mechanism to pass any additional parameters

defined in the `as.xcf` file of the application to the delegate service. For instance, in the following `xcf` delegation setting example using Security Assertion Markup Language (SAML) service :

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

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

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

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

Internationalization

How the Genero Application Server handles international applications.

- Encoding Architecture on page 43
- Charsets Configuration on page 44
- Translations for GWC-JS on page 47

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

Encoding Architecture

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

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

This diagram summarizes the GAS character set encoding architecture:

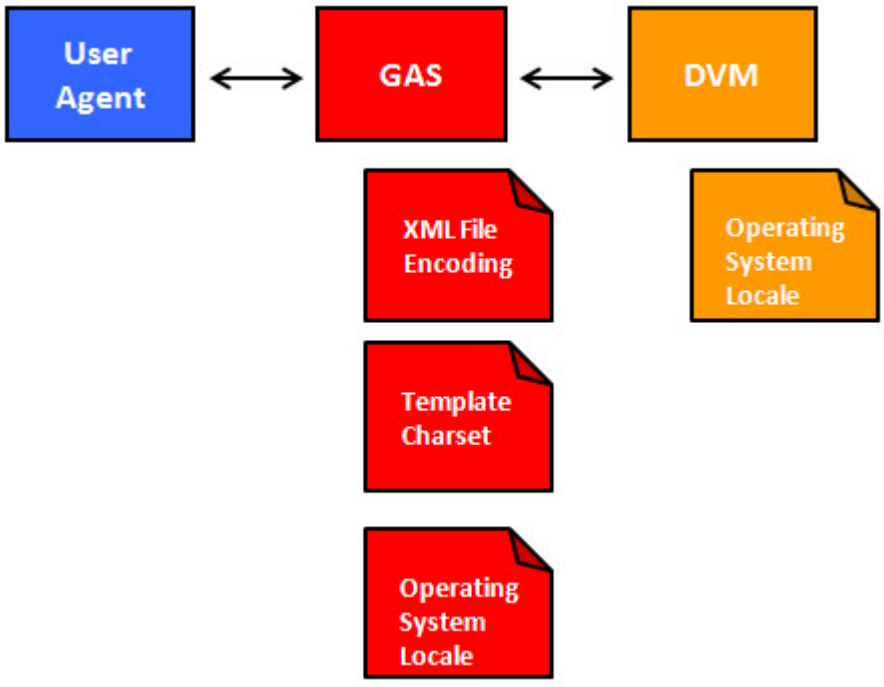


Figure 8: GAS Encoding Architecture

The GAS uses:

- XML File Encoding
- Template Charset
- Operating system locale

The DVM uses:

- Operating system locale

Charsets Configuration

Charsets can be defined in four places.

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

- DVM Locale on page 44
- HTML charset on page 45
- XML Encoding on page 45
- GAS System Encoding on page 46
- Default Encoding on page 47

DVM Locale

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

Setting a DVM locale is described in the *Genero Business Development Language User Guide*, chapter "Localization".

On UNIX, the DVM locale is defined by the *LANG* or *LC_ALL* environment variables, which can be specified in the GAS executing environment, or with the `ENVIRONMENT_VARIABLE on page 314` child of a `WEB_APPLICATION_EXECUTION_COMPONENT on page 356` element, inside the **as.xcf** file.

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

```
<?xml version="1.0" encoding="UTF-8"?>
...
  <COMPONENT_LIST>
     <WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
       <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</
ENVIRONMENT_VARIABLE>
       <ENVIRONMENT_VARIABLE Id="FGLGUI">$(res.fglgui)</
ENVIRONMENT_VARIABLE>
       <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
       ...
       <ENVIRONMENT_VARIABLE Id="LC_ALL">ru_RU.KI08-R</ENVIRONMENT_VARIABLE>
       <DVM>$(res.dvm.wa)</DVM>
     </EXECUTION_COMPONENT>
...
</COMPONENT_LIST>
```

Such `WEB_APPLICATION_EXECUTION_COMPONENT` definition can then be referenced by an `APPLICATION (for an application) on page 301` element using the `<EXECUTION Using="identifier">` tag.

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

HTML charset

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

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

Example

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

```
<!DOCTYPE html>

<!--
   FOURJS_START_COPYRIGHT(D,2014)
   Property of Four Js*
   (c) Copyright Four Js 2014, 2015. All Rights Reserved.
   * Trademark of Four Js Development Tools Europe Ltd
     in the United States and elsewhere

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

<html>

<head>
  <meta charset="utf-8">
  <title>Genero Browser Client</title>

  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  ...
  };
</script>
</head>
...
```

XML Encoding

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

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

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

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

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

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

```
...
```

GAS System Encoding

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

For example, the Genero Application Server:

- Writes log files.
- Opens files defined in the Genero Application Server configuration file.
- Reads arguments from the command line.

In these cases and more, the Genero Application Server uses the character encoding set by the operating system. In the UNIX™operating system encoding is defined via environment variables *LANG* or *LC_ALL*. For more information see the "Localization" topics in *Genero Business Development Language User Guide* or see The Single UNIX - Specification Version 2 - Locale.

In the Windows™ operating system typically the GAS defaults to the system locale as defined in the language and regional settings. There should be no need to set the *LANG* variable, except your application uses a different character set to the Windows system locale.

Displaying locale settings

Locales supported by a UNIX operating system can be displayed with the `locale -a` command.

On Windows™ platforms, code pages that define the encoding can be displayed in the console window with the `chcp` command.

Setting system encoding

By default GAS uses the UTF-8 character set as an Unicode standard to support international characters and multiple languages. If your operating system doesn't support the desired character encoding, or if you need a specific encoding, you can set the system encoding with the *LANG* variable in Windows or in UNIX systems with either the *LANG* or *LC_ALL* environment variables. For example, to set encoding to UTF-8 on a Windows platform:

```
C:\ set LANG=.fglutf8
```

For example setting encoding on UNIX to US English:

```
$ LC_ALL=en_US.iso88591; export LC_ALL
```

How GAS does character set conversion

The GAS software takes care of character set conversions:

- For `xcf` files, it does the conversion based on what the XML *prolog* specifies as charset to the GAS locale, for example:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  ```
- For templates files (GWC-JS), the charset in the *metadata* of *$FGLASDIR*`/web/gwc-js/bootstrap.html` file is used.
- For the front end clients such as GWC-JS that use UTF-8 encoding, the DVM (V3) does the conversion in the DVM locale, see DVM Locale on page 44.

  Note: Operating system character sets may have different names across operating systems. To unify character set names in the application server environment, the GAS manages a character set encoding name conversion to map the operating system character encoding name to a canonical name:

- A `charset.alias` file is provided. This file is located in the *`$FGLASDIR`*`/etc` directory of the GAS.

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

See Unicode characters for more information.

Translations for GWC-JS

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

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

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

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

Excerpt from US English Locale File

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

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

Retrieving Localized Texts

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

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

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

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

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

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

In the JavaScript

Internationalization is done by the GWC-JS JavaScript function `i18n.t`. This function takes the localization key as argument and returns the translation text.

For example retrieving the value of the key "gwc.app.restart":

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

Selecting a Language

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

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

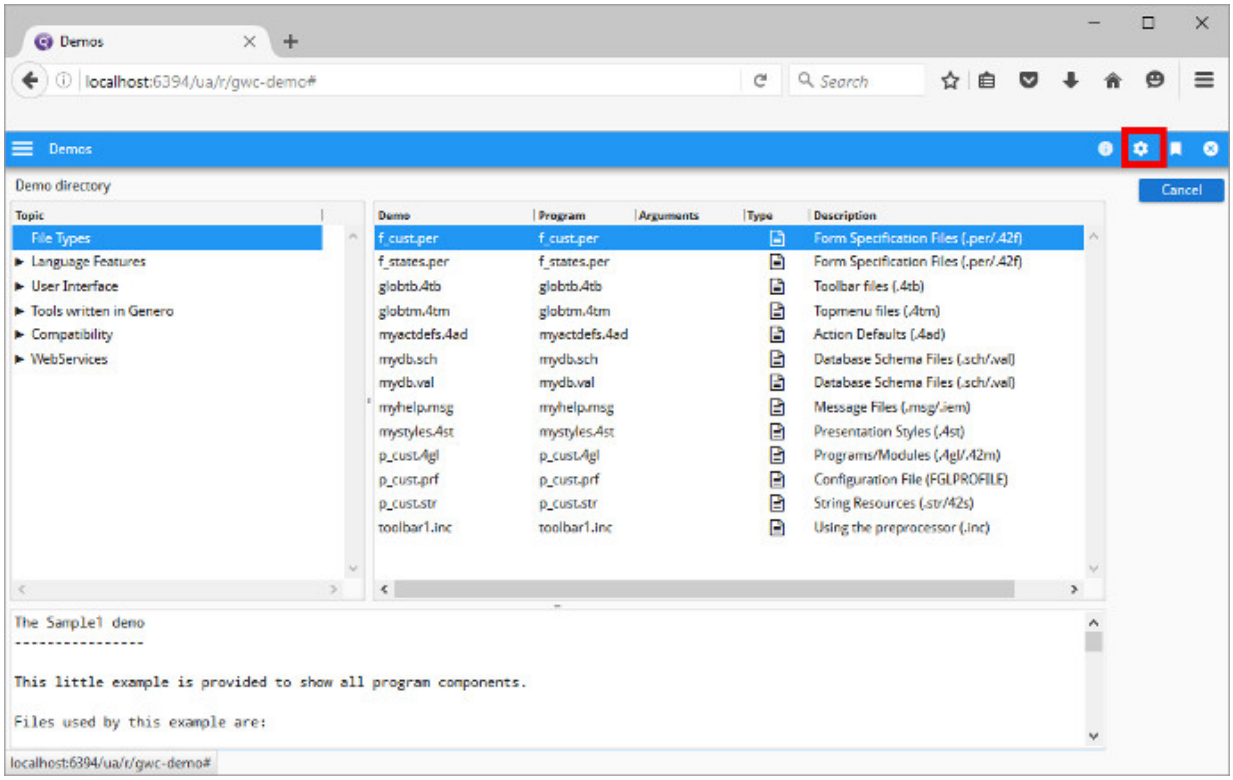


Figure 9: GWC-JS User Interface Menus

- Choosing the language from the **Interface language** list in the **Stored Settings configuration** window that pops up.

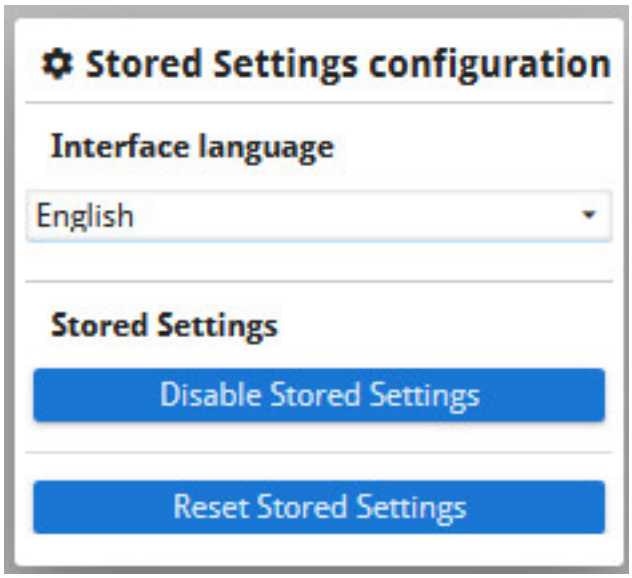


Figure 10: Stored Settings configuration Window

The name of the language, like "English", that you see displayed is defined in the locale file by the entry `gwc.lngName`, for example, as in the en-US.json:

```
{
  "gwc": {
    "lngName" : "English",

[…]
}
```

Application Web Address

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

Note: For details on the URI for integrated application servers, see:

- GAS ISAPI Installation / Web Server Configuration
- GAS FastCGI Installation / Web Server Configuration

- URIs acknowledged by the GAS on page 49
- URI Examples on page 55

URIs acknowledged by the GAS

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

- Application URIs on page 49
- File serving URIs on page 53

Application URIs

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

URI Syntax

This topic describes `the` URIs acknowledged by the GAS based on the standard syntax shown below and explained in Table 6: Explanation of URI syntax options on page 50. This topic provides you with URIs

to access your Genero applications. The information is intended to help you tune and monitor your web server configuration for Genero.

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

Table 6: Explanation of URI syntax options

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

Option	Description	Valid Values
scope	Scope or protocol you are working on, e.g. uaproxy or Web service	ua, ws
action	Action requested of the Application Server	See Application URIs acknowledge by the GAS
group	Application group defined in `as.xcf`	see GROUP (for an application) on page 318
web-application-id	Web application identifier	e.g. *gwc-demo*
parameter	Parameter to communicate to the Application Server to start application with arguments	See Example: Starting applications with arguments on page 55
parameter-value	Parameter value	See Example: Starting applications with arguments on page 55

Table 7: Application URIs Acknowledged by the GAS

URI path	Description
`/ua/r`	GWC-JS application is started with a `/ua/r` request, see Example: Calling a Web application on page 56;
`/ua/sua`	Once the startup request has been answered, the DVM uses `/ua/sua` requests to submit user actions and communicate with the front-end client.
`/ua/ft`	GWC-JS file transfer
`/ua/interrupt`	Front-end client uses the interrupt request to send an interrupt message to the DVM.
`/ua/i`	Resource, e.g. a fully-qualified URL pointing to an application's image file, see Table 8: File serving URIs on page 54
`/ua/components`	Web component resource, e.g. a fully-qualified URL pointing to an application's web component file, see Table 8: File serving URIs on page 54
`/ua/close`	Front-end client sends a close request to the DVM to stop an application.
`/ua/newtask`	Front-end client sends a newtask request to ask the GAS if it has any pending applications that have not been already notified to the client.
`/ua/report/viewer`	URL with the `report/viewer` prefix is used by the Genero Report Engine (GRE) running in **local mode** to load the HTML report viewer implementation. It is bound to the

URI path	Description
	$(GREDIR)/viewer directory by default, see REPORT_VIEWER_DIRECTORY on page 340.
/ua/report/public	URL with the report/public prefix is used by the GRE to access public resources or documents shared between applications on the local server, for example, report fonts.
/ua/report/private/$(session.id)/reports	URL with the report/private prefix is used by the GRE to access private resources or documents locally. These resources are not shared between applications or users and are deleted when the session ends.
/ua/report-r/viewer	URL with the report-r/viewer prefix is used by the GAS running a GRE in **distributed mode** to load the HTML report viewer implementation. Adding a REPORT_REMOTE_URL_PREFIX configuration entry in the GAS as.xcf configuration file enables distributed mode support.
/ua/report-r/public	URL with the report-r/public prefix is used for by GAS to access public resources or documents on remote server shared between applications, for example, report fonts.
/ua/report-r/private/$(session.id)/reports	URL with the report-r/private prefix is used by the GAS to access private resources or documents on a remote server. These resources are not shared between applications or users and are deleted when the session ends.
/reports/delete/${sessionid}	The GAS will send an HTTP request to the remote host to notify it that the session has ended. The remote server is responsible for removing recursively private data identified by the session identifier. For example, `DELETE /reports/delete/ebf19aba4d09e80cd19cfea7868a9a2a`
/ua/w	URL dedicated to the look up of the GWC-JS directory. The complete format of the URI is ua/w/*$(GWC-JS)*/<filename>, where files served directly by the dispatcher for the customized GWC-JS are found. The GAS searches for the GWC-JS

URI path	Description
	directory in one of the paths specified as document root in GWC_JS_LOOKUP_PATH on page 320.
`/ws/r`	GWS request to start Web service application, see Example: Calling a Web Service application on page 56
`/da/r`	Request to start application using the GDC from URI, see Example: Calling Desktop application on page 55
`/monitor`	Monitor requests
`/monitor/log`	Monitor logs requests
`/monitor/configuration/application/group`	Returns a list of all groups configured for applications, see Example: Get the groups for all applications on page 56.
`/monitor/configuration/application/name?abstract`	Returns a list of all application names (or only the abstract applications if **?abstract** is specified).
`/monitor/configuration/application/dua?appId`	Returns a list of all DUAs allowed for the application identified by **appId**
`/monitor/configuration/service/group`	Returns a list of all groups configured for services.
`/monitor/configuration/service/name?abstract`	Returns a list of all service names (or only the abstract services if **?abstract** is specified).
`/monitor/configuration/service/invalid`	Returns a list of static and dynamic services that have been invalidated by the dispatcher because first request did not succeed in starting a DVM. See Example: Get list of services invalidated by dispatcher on page 56

Note: A URI path is a string without line breaks. Line breaks have been added to the URIs above to support the printed version of our documentation.

File serving URIs

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

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

Table 8: File serving URIs

URI path	Description

/	The slash (/) is the path to the document root configured in the Genero Application Server (GAS) configuration file. The DOCUMENT_ROOT element is located at /CONFIGURATION/APPLICATION_SERVER/INTERFACE_TO_CONNECTOR/DOCUMENT_ROOT, see DOCUMENT_ROOT on page 312.
	
	For example, this excerpt is from the default GAS configuration file (as.xcf).
	
	`...`
	`<RESOURCE_LIST>`
	`<PLATFORM_INDEPENDENT>`
	`...`
	`<RESOURCE Id="res.path.docroot" Source="INTERNAL">`
	`$(res.path.as)/web</RESOURCE>`
	`...`
	`</PLATFORM_INDEPENDENT>`
	`</RESOURCE_LIST>`
	`...`
	`<INTERFACE_TO_CONNECTOR>`
	`...`
	`<DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>`
	`...`
	`</INTERFACE_TO_CONNECTOR>`
	`...`
	
	In this example, when calling the demo page using the URL http://localhost:6394/demos.html, you expect to find the demos.html file in the directory specified by the DOCUMENT_ROOT entry, i.e. the /web directory in your application server path.

/ua/i/	In this URL the slash (/) following the ua/i / is the path to the public resource directory, where resources such as images are found, see Paths to application resources on page 219. In this example, when accessing the photo.jpeg with the URL http://localhost:6394/**ua/i**/photo.jpeg, you expect to find the photo.jpeg file in the directory specified by the *$(res.public.resources)*, which expands to *$(res.appdata.path)*/public/common directory in your application server's application data path.

/ua/components	In this URL the slash (/) following the ua/components/ is the path to the web component directory configured in the Genero Application Server (GAS) configuration file where web components are found. The WEB_COMPONENT_DIRECTORY element is located at /CONFIGURATION/APPLICATION_SERVER/WEB_APPLICATION_EXECUTION_COMPONENT/WEB_COMPONENT_DIRECTORY, see WEB_COMPONENT_DIRECTORY on page 361 .
	
	For example, this excerpt is from the default GAS configuration file (as.xcf).
	
	`...`
	`<WEB_APPLICATION_EXECUTION_COMPONENT`
	`Id="cpn.wa.execution.local">`
	`...`
	`<WEB_COMPONENT_DIRECTORY>$(application.path)/webcomponents</WEB_COMPONENT_DIRECTORY>`
	`...`
	`</WEB_APPLICATION_EXECUTION_COMPONENT>`

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

URI Examples

Several URI examples.

This topic provides you with some typical examples of ways of starting web (GWC-JS or GDC-HTTP) and web services applications (GWS).

Example: Direct connection to standalone GAS

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

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

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

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

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

Example: Connection through a Web Server

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

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

Example: Connection using a group

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

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

Example: Starting applications with arguments

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

```
http://myWebServer/gas/ua/r/myApp?Arg=Val1&Arg=Val2
```

Note:

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

Example: Calling Desktop application

Using the URL with "da" launches the "appid" application via the Genero Desktop Client monitor.

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

Note: Prerequisites:

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

Example: Calling a Web Service application

To get the WSDL for a specified service:

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

To access the Web service:

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

If the Web service uses a group:

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

Access through a Web server:

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

Example: Calling a Web application

Calls the "myApp" application:

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

Example: Get the groups for all applications

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

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

and you will get a XML document like following:

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

Example: Get list of services invalidated by dispatcher

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

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

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

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

If there are no invalid services, you should get a page displaying, "There are no invalid configuration files".

Configuring the Genero Application Server

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

System Requirements

Failure to meet system requirements can result in unexpected behavior.

Operating systems

The Genero Application Server (GAS) is supported for a variety of operating systems, to include Linux®, IBM® AIX®, HP-UX, SUN Solaris and Microsoft™ Windows™.

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

Web servers

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

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

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

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

User agents

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

For the Genero Web Client, the supported user agents are agents supporting HTML5.

We support the up-to-date browsers for each of the supported browsers. Users must update their browsers to the latest version.

Supported browsers are:

- Microsoft™ Internet Explorer
- Microsoft™ Edge
- Google Chrome

Note: As development work continues in Genero V3, we will update this page when support for other browsers is available.

Databases

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

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

GAS Configuration Check

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

- Managing Access Rights on page 58
- Validating the Installation with the Genero Web Client on page 58
- Validating the Installation with the GDC on page 59
- Troubleshooting Configuration Issues on page 60

Managing Access Rights

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

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

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

Validating the Installation with the Genero Web Client

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

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

- Validate the installation for GWC without a Web server on page 58
- Validate the integration for GWC with a Web server on page 59

Validate the installation for GWC without a Web server

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

Before you begin:

Due to configuration settings in the Genero Application Server configuration file, access to demo applications needs to be configured. Make sure you have provided access to run the demo applications from the localhost, see Example configuring access control for demo applications on page 299.

1. Set the GAS environment by executing the script `$FGLASDIR/envas`.
2. Launch the GAS Standalone: `httpdispatch`

   For more information on starting the GAS standalone, and the various command options available, see Tools and Commands on page 263.
3. From your web browser, check the connection to the application server by displaying the `Demos` page, using a URI that provides a direct connection to the standalone GAS:

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

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

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

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

Validate the integration for GWC with a Web server

To validate the installation of the Genero Application Server using a Web server, specify the Web server as part of the application URI and run a Genero Web Client demo application.

Before you begin:

Due to configuration settings in the Genero Application Server configuration file, access to demo applications needs to be configured. Make sure you have provided access to run the demo applications from the localhost, see Example configuring access control for demo applications on page 299.

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

1. Before you begin, you should have first validated the installation as a standalone GAS. See Validate the installation for GWC without a Web server on page 58.
2. From your web browser, ensure that your web server is correctly configured by accessing a static page (such as index.html), or simply http://localhost.
3. Display the Demos page:

   ```
   http://myWebServer/gas/demos.html
   ```
4. Launch the GWC Demos program.

   ```
   http://myWebServer/gas/ua/r/gwc-demo
   ```

Validating the Installation with the GDC

The Genero Application Server can deliver clients to the Genero Desktop Client.

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

- Validate the installation for GDC without a Web server on page 60

Validate the installation for GDC without a Web server

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

Before you begin:

Due to configuration settings in the Genero Application Server configuration file, access to demo applications needs to be configured. Make sure you have provided access to run the demo applications from the localhost, see Example configuring access control for demo applications on page 299.

1. Set the GAS environment by executing the script `$FGLASDIR/envas`.
2. Launch the GAS Standalone: `httpdispatch`

   For more information on starting the GAS standalone, and the various command options available, see Startup and Command Options.
3. Within the GDC, create a shortcut pointing to the demo application.

   To create the shortcut, you must start the GDC in administrative mode using the `--admin` or `-a` option. Refer to the *Genero Desktop Client User Guide* for more information on creating shortcuts.

   a) On the first page of the New Shortcut wizard, select the **HTTP, through a web server** option.

   b) On the second page of the New Shortcut wizard - the HTTP connection information page - provide the application URL (`http://myApplicationServer:6394/ua/r/gdc-demo`) and specify the HTTP Proxy mode as **Direct connection**.

   On most systems, you can replace the *myApplicationServer* with `localhost` for this test: `http://localhost:6394/ua/r/gdc-demo`.

   c) For the remaining pages of the Shortcut Wizard, you can accept the defaults and click **Finish**.

   The shortcut is added to the GDC.
4. To run the application, select the shortcut and click **Start!**.

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

Troubleshooting Configuration Issues

Troubleshooting tips are provided for the most common issues encountered.

Proxy errors on Windows™ platform

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

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

```
netsh winsock reset catalog
```

The command resets the Winsock Catalog to a clean state. Be aware that it might affect your installed applications that use the internet. You might need to reconfigure or reinstall such applications, so use the command cautiously. The command will ask to restart Windows™.

Cannot find 127.0.0.1 or localhost on Windows™
Users on Windows™ 64-bit machines who are using a network proxy: The browser cannot open 127.0.0.1 or localhost unless you modify your Advanced Network settings to avoid going through the proxy for these addresses.

Application does not start
If an application does not start, you can debug the problem by manually launching the program from a command shell.

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

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

When the GAS starts an application from a Web browser with a given URI, instead of displaying the application forms in the Web browser, a command window will open with all environment settings for that application. You can then manually run your application in debug mode, for example with `fglrun -d progname` to enter the command-line debugger `fgldb` (application forms will then display in the Web browser).

For more information on starting the standalone dispatcher to open a command window and on using the FGL debugger for your platform, please see the following:

- Using the Debugger for the GAS on UNIX on page 158
- Using the Debugger for the GAS on the Windows platform on page 157

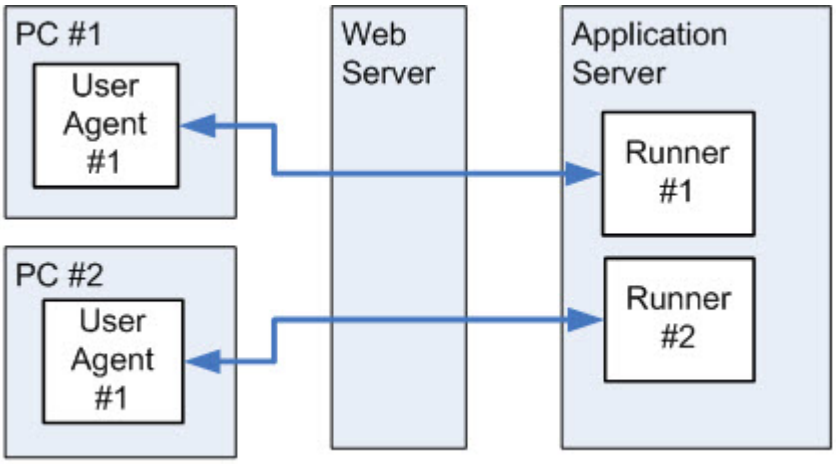
Licensing

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

> **Important:** This topic discusses Genero licensing. Some of the demos accessed via the demos.html page use third-party components which are NOT delivered, licensed or supported by your Genero provider. They are just bundled for the purpose of demonstrating the various capabilities of some of the GWC modes. Before using a demo code snippet for your own development, take care that you fulfill the corresponding licenses from all third-party components.

Licensing - Base Example

This scenario shows two User Agents connected to two DVMs.



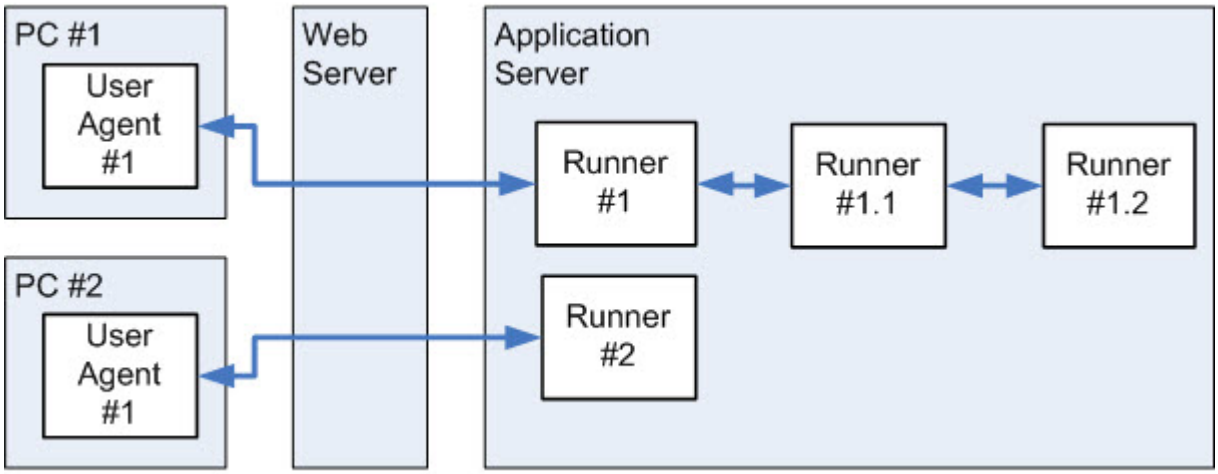
The connection is made via the Web server, the ISAPI or FastCGI extension, and dispatchers.

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

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

Licensing - Using the RUN command

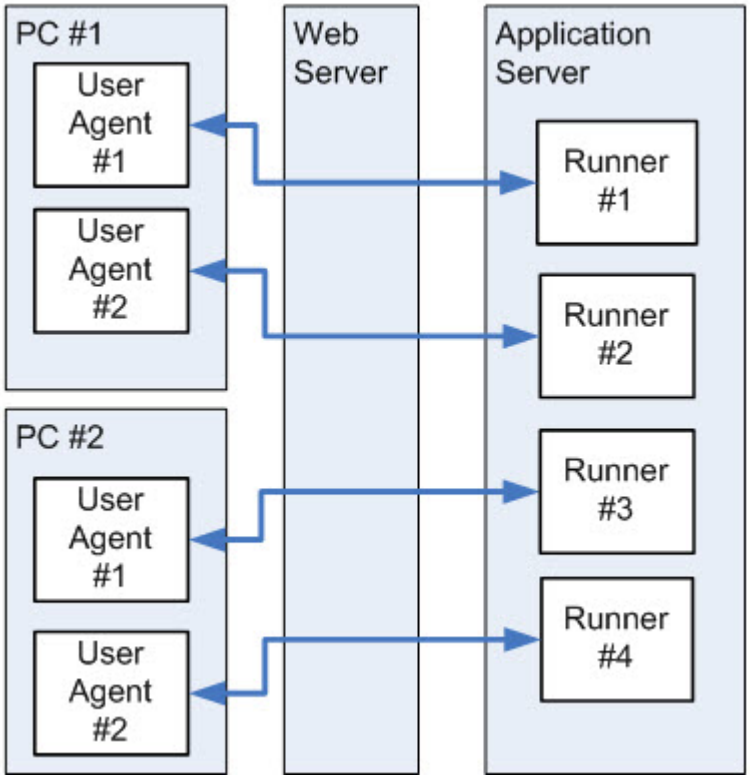
This scenario shows two User Agents connected to an application, which in turn calls other applications using the Genero BDL `RUN` command or the `RUN WITHOUT WAITING` statement.



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

Licensing - Multiple User Agents

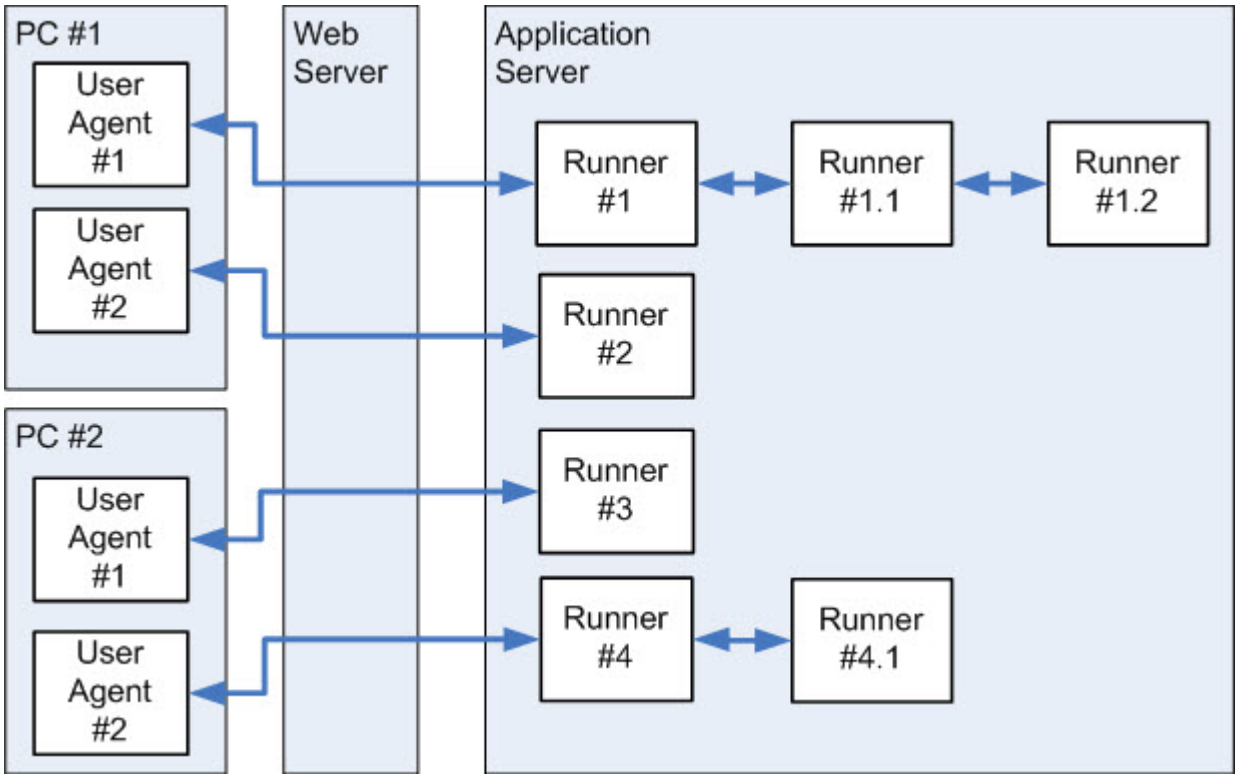
This scenario shows four User Agents running on two different PCs and connected to four DVMs.



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

Licensing - Summary Case

This scenario shows four User Agents running on two different PCs and connected to four DVMs, some of which are running external DVMs using the Genero BDL `RUN` command.



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

Genero Front Ends and License Counting

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

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

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

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

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

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

To determine the number of licenses used, run "`fglWrt -u`" followed by "`fglWrt -a info users`" on the application server where the Genero runtime is installed.

Important: When you do a refresh on a GWC application, if the current page is the first page of the application this will not refresh the application page but start a new application. Thus, consume an extra license.

Licensing Tips and Tricks / Troubleshooting

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

The ADDRESS element

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

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

The USER_AGENT timeout element

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

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

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

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

The MAX_AVAILABLE element for a Web Service

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

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

Evaluate licensing when migrating from GDC

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

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

ISAPI Extension Installation and Web Server Configuration

The Genero Application Server and IIS

The Genero Application Server (GAS) is embedded within the ISAPI extension itself, and is directly loaded by an Internet Information Services (IIS) worker process. This improves performance and allows administration of the server with IIS tools.

The GAS is seen by IIS as a script engine, enabled to handle every request to the virtual directory bound to the GAS.

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

To take full advantage of this type of installation, you must create a new web site, or a new application or virtual directory on an existing web site, e.g. IIS "Default Web Site". This section shows how to:

- Create and set up a virtual directory using the Windows installer, see Installing with the Microsoft Installer on page 66
- Activate IIS for your installation (see the appropriate configuration guide for your installation)
- Configure application pool
- Create an application
- Create a GAS ISAPI Extension configuration file

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

Install the ISAPI dispatcher

Install the ISAPI dispatcher within the IIS framework.

Installing with the Microsoft™ Installer

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

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

Once the file is downloaded, click on the file to execute the installation wizard. You will follow the instructions provided by the wizard, to include selecting the install directory, providing the Start Menu Folder name, and specifying the appropriate FGLDIR to use for this GAS installation.

At this point in the wizard, you are asked whether you want to install the ISAPI Extension. If you select NOT to install the ISAPI Extension, then you will either only have a standalone GAS or you will have to install the ISAPI Extension manually at a later time. By default, it will attempt to install the ISAPI Extension.

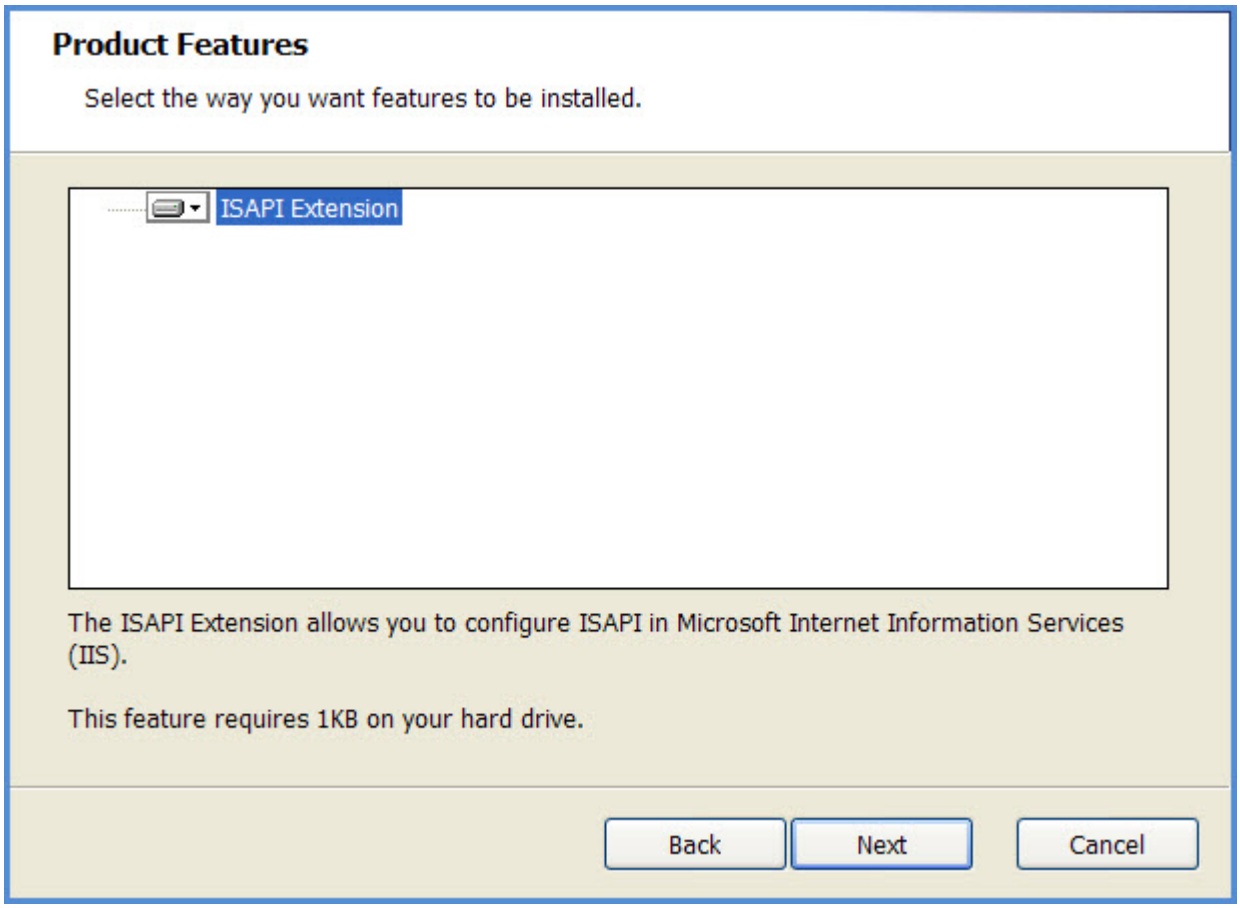


Figure 11: Wizard, Product Features page

If the ISAPI Extension is selected, you must complete three wizard pages.

On the first wizard page, you select the Web Site where you want to add your Web application.

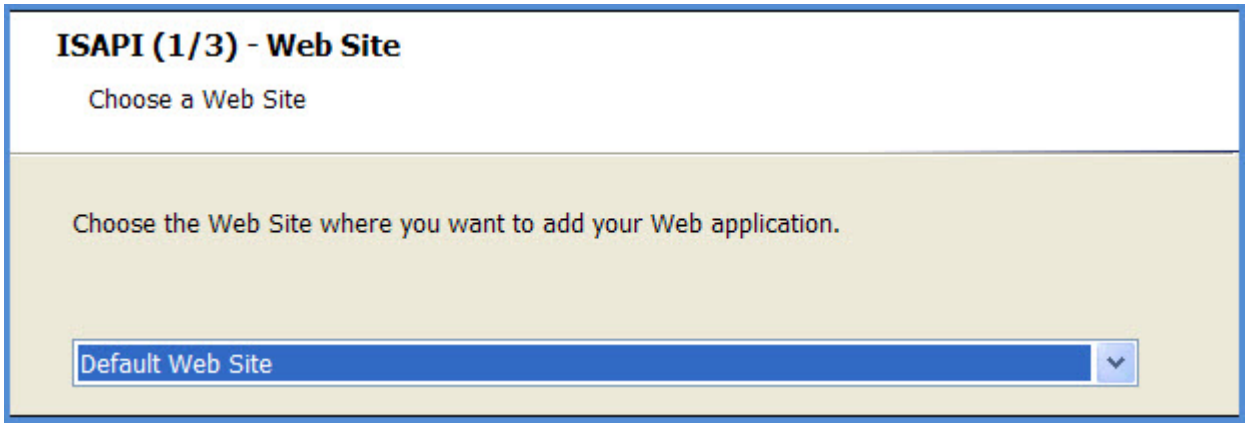


Figure 12: Wizard, ISAPI (1 of 3) - Web Site page

On the second wizard page, you provide the name for the Web Application. This is the name that will be included as the path to the GAS in the URL for your applications.

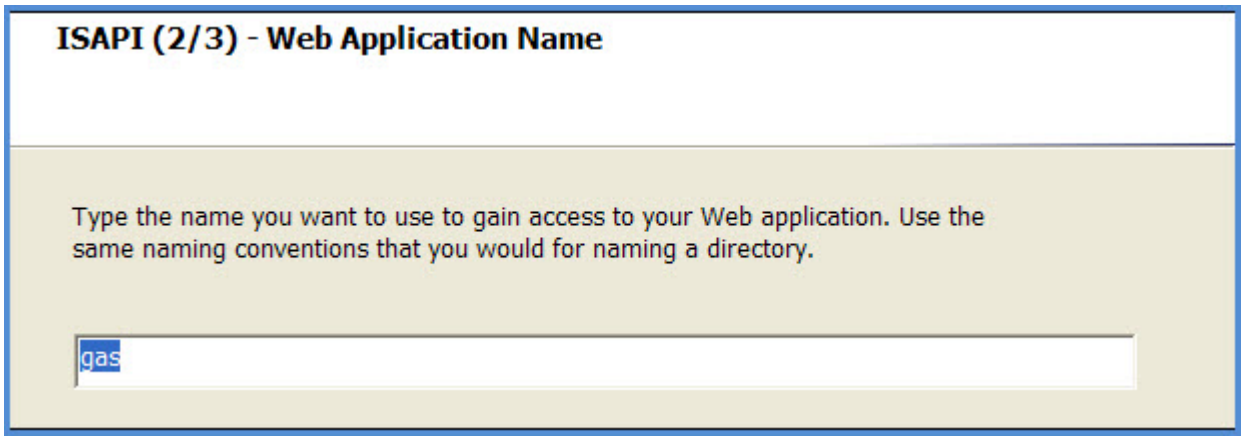
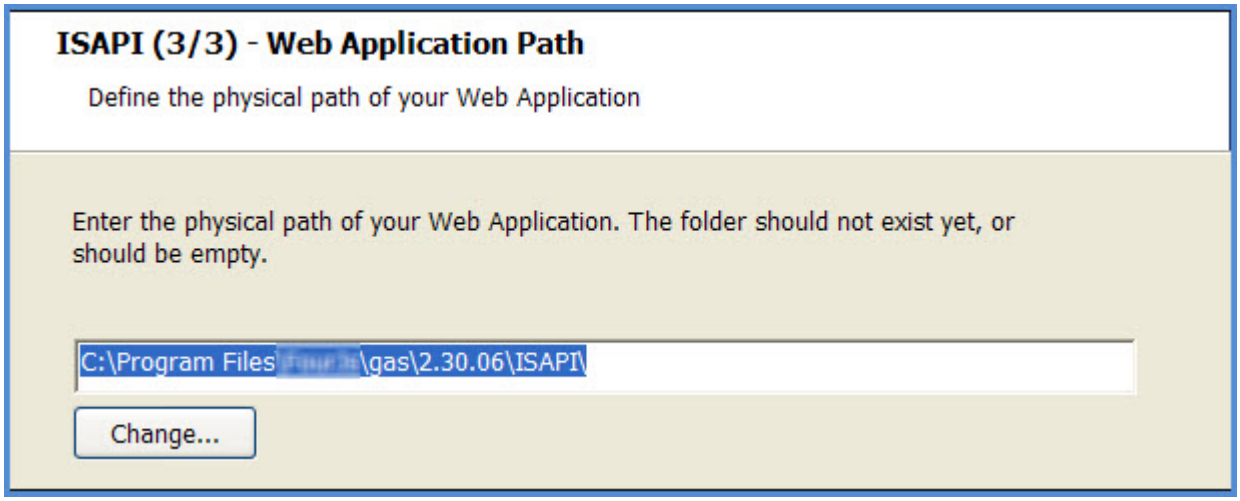


Figure 13: Wizard, ISAPI (2 of 3) - Web Application Name page

On the third wizard page, you specify the web application path. It is within this directory that the isapidispatch.ini file will be created.

Figure 14: Wizard, ISAPI (3 of 3) - Web Application Path page



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

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

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

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

```
http://localhost/gas/demos.html
```

If you have an installation failure, or if you opted not to include the ISAPI Extension, the next sections help you to manually configure the ISAPI Extension.

Manual configuration for IIS 6.0

These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 6.0.

Configuring IIS 6.0 application pools

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

To create an application pool:

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

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

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

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

1. In IIS manager, right-click the application pool, and then click **Properties**.
2. Click the **Identity** tab.
3. According to your security policy, either select a built-in account or set a custom account.
4. Click **OK**.

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

Configuring an IIS 6.0 application

To create a virtual directory:

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

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

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

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

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

Post requisites

After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see Finishing the installation on page 83 .

Manual configuration for IIS 7.x

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

Prerequisites

The installer needs some IIS features to be activated:

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

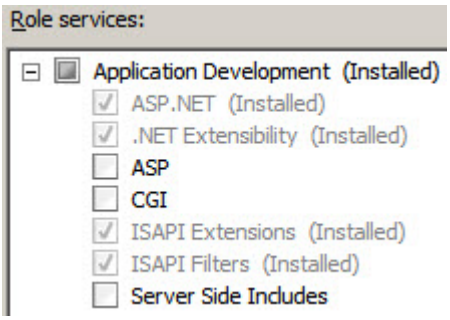
To activate "IIS Management Scripts and Tools" on a Windows™ Server 2008:

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

Do the same for the other features.

Example:

Figure 15: Role services dialog



To activate IIS Management Scripts and Tools on Windows™ 7

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

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

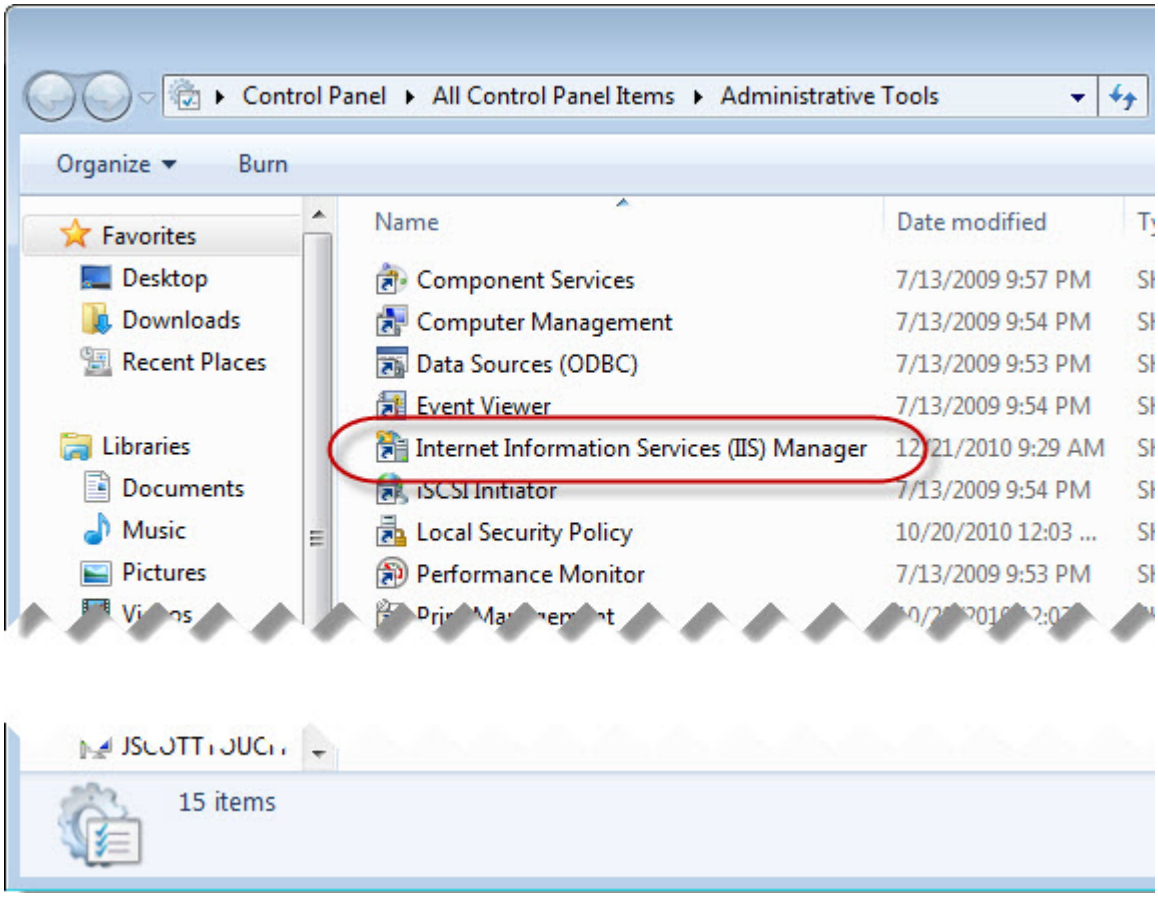


Figure 16: Control Panel with IIS Manager highlighted

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

Figure 17: IIS Welcome Screen

3. Set the appropriate Windows™ Features. Go to **Control Panel** >> **Programs and Features**. Click **Turn Windows™ features on or off**.

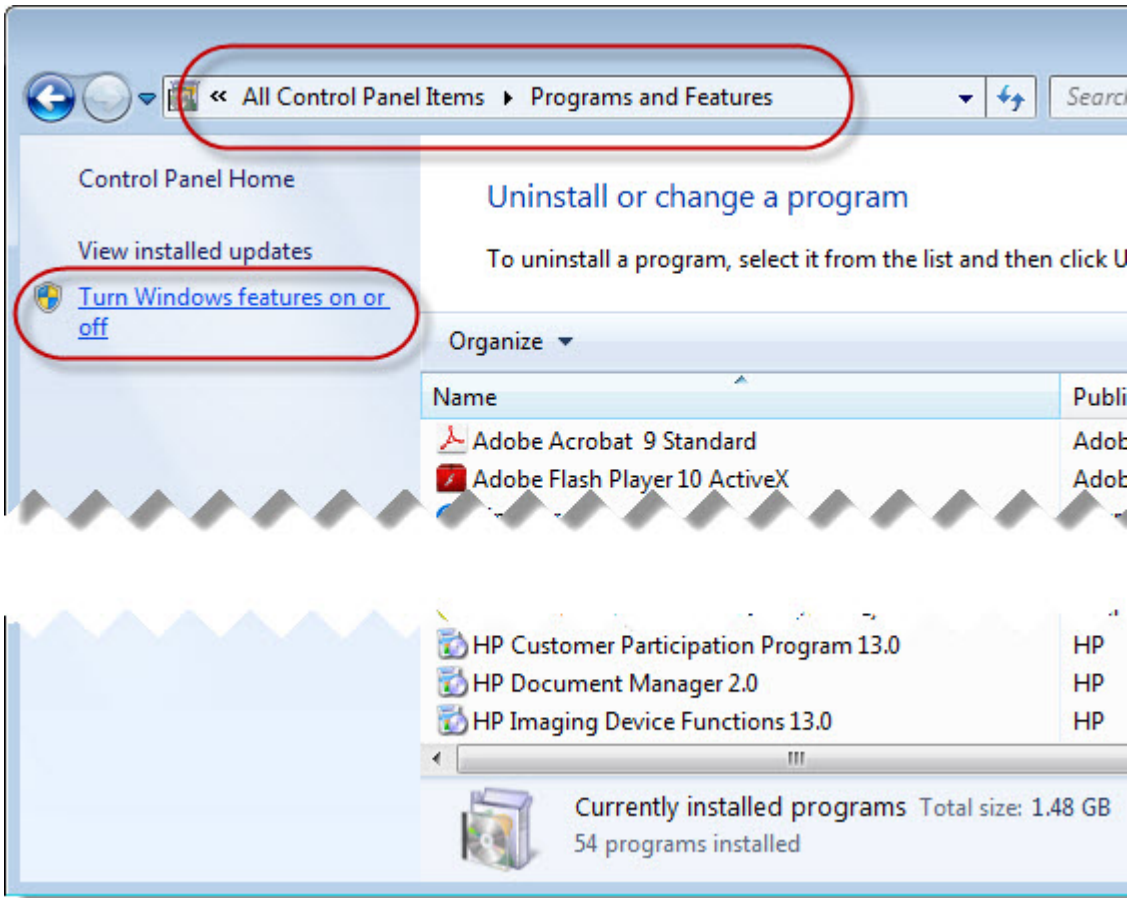


Figure 18: Turn Windows™ features on or off link

4. In the **Windows™ Features** dialog, select **IIS Management Scripts and Tools**, and verify that the ASP and the ISAPI options are also selected. When checking these options, other options may also be automatically checked. That is normal behavior, as they are combined options.

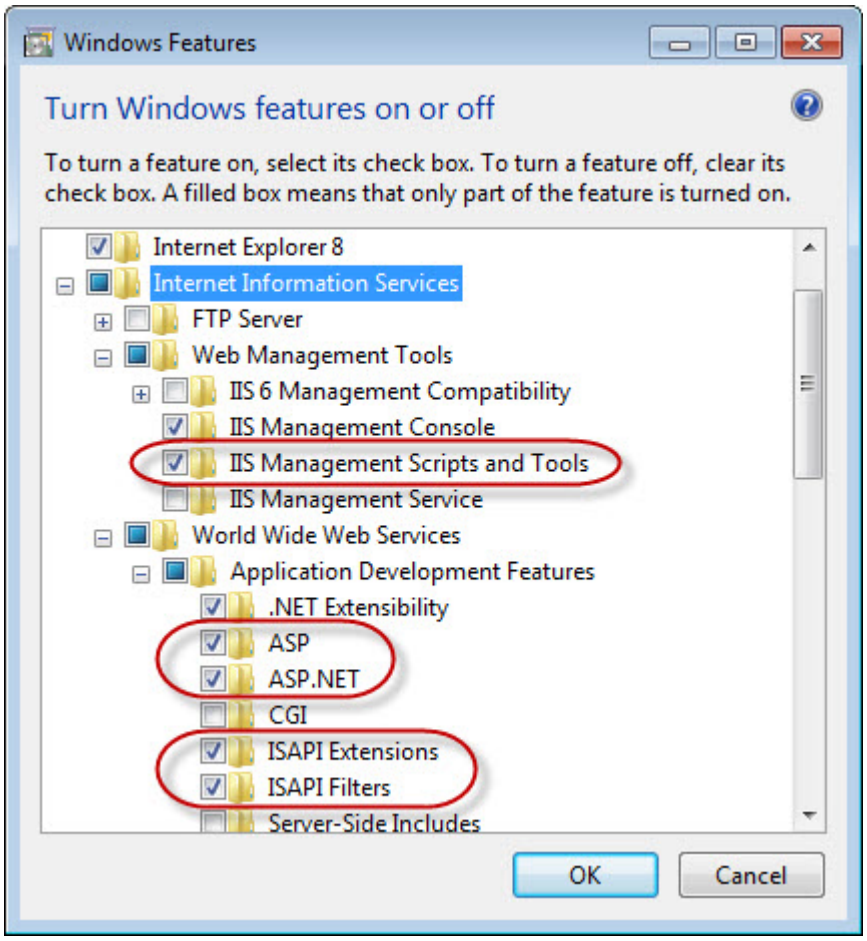


Figure 19: Turn Windows™ features on or off dialog

Configuring IIS 7.x application pools

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

To create an application pool:

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

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

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

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

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

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

Configuring an IIS 7.x application

To create an application:

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

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

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

To bind the application to the GAS ISAPI Extension:

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

11. In the **Actions** area, click **Edit Feature Permissions...**
12. In the **Edit Feature Permissions**, ensure that **Script** is selected and all other are unselected.
13. Click **OK**.

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

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

Post requisites

After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see Finishing the installation on page 83 .

Manual configuration for IIS 8.x and IIS 10.x

These are the instructions for the manual configuration of the ISAPI dispatcher for IIS 8.0 and IIS 10.0.

Prerequisites

The installer needs some IIS features to be activated:

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

To activate IIS on a Windows™ Server 2012:

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

To activate IIS Management Scripts and Tools on Windows™ 8 and 10

To activate IIS Management Scripts and Tools on Windows™ 8 and 10 and validate the basic IIS configuration requirements:

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

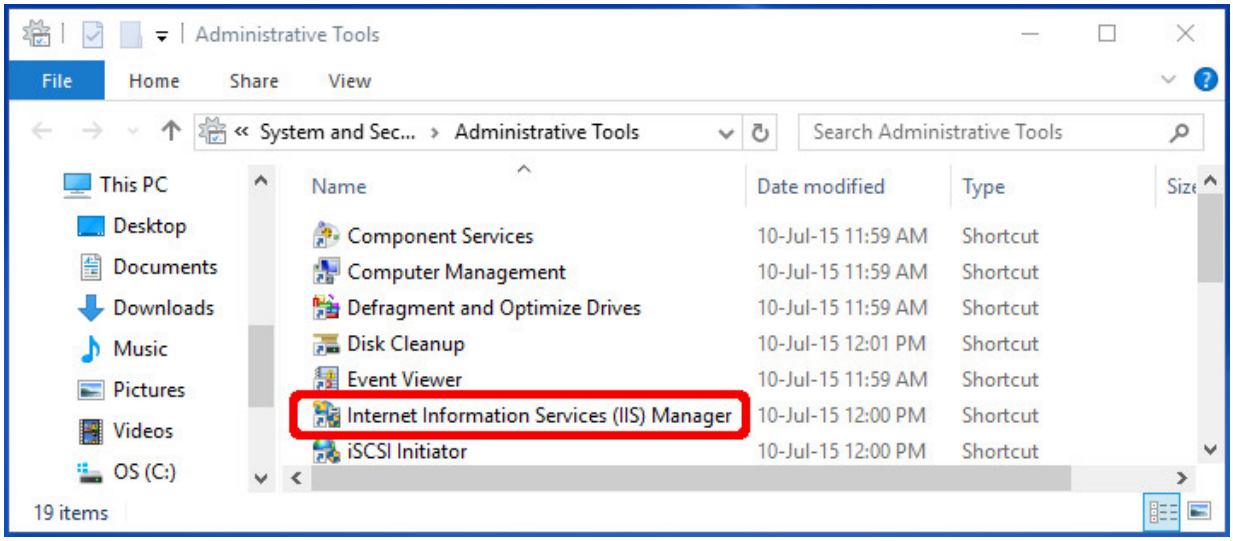


Figure 20: Administrative Tools window with IIS Manager highlighted

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

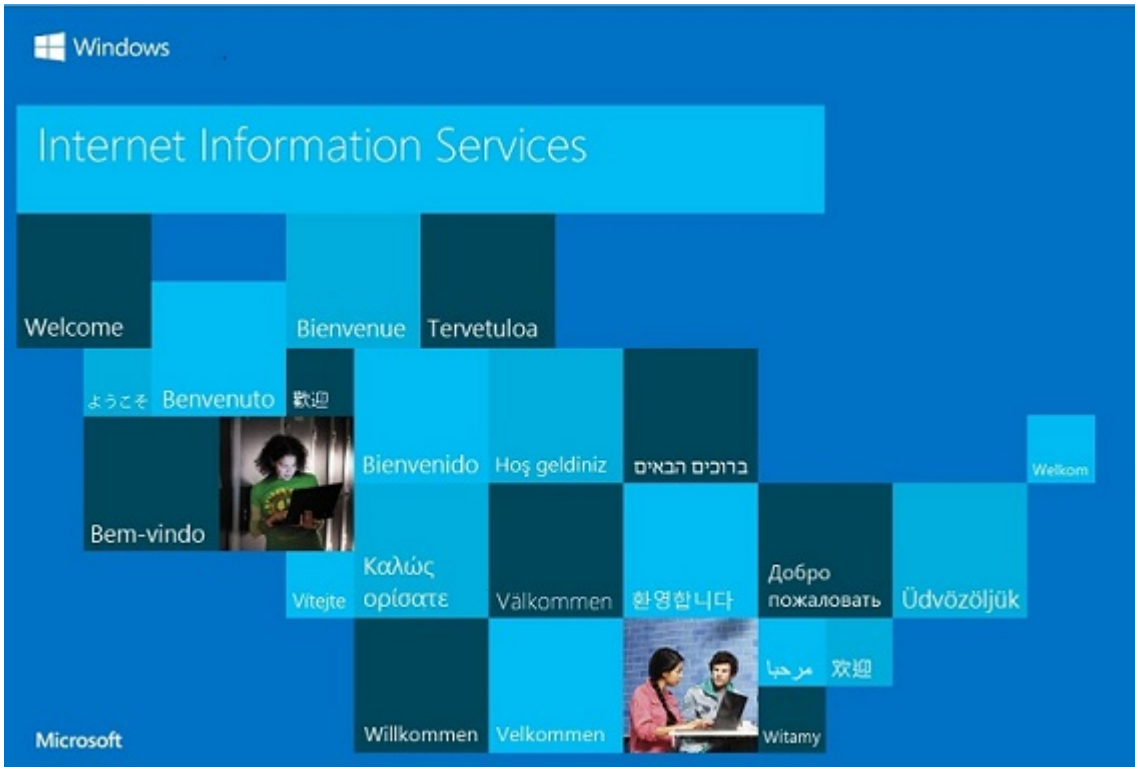


Figure 21: IIS Welcome Screen

3. Set the appropriate Windows™ Features. Go to **Control Panel** >> **Programs and Features**. Click **Turn Windows features on or off**.

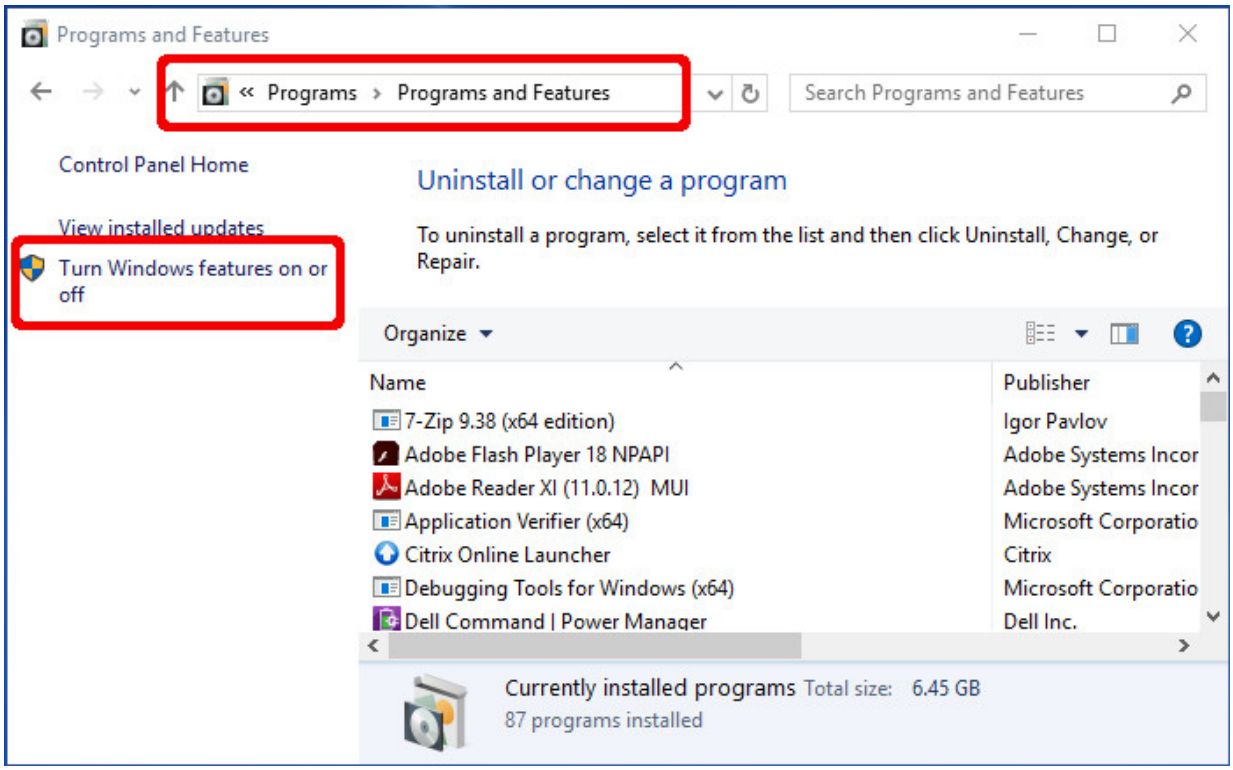


Figure 22: Turn Windows™ features on or off link

4. In the **Windows™ Features** dialog, select **IIS Management Scripts and Tools**, and verify that the ASP and the ISAPI options are also selected. When checking these options, other options may also be automatically checked. That is normal behavior, as they are combined options.

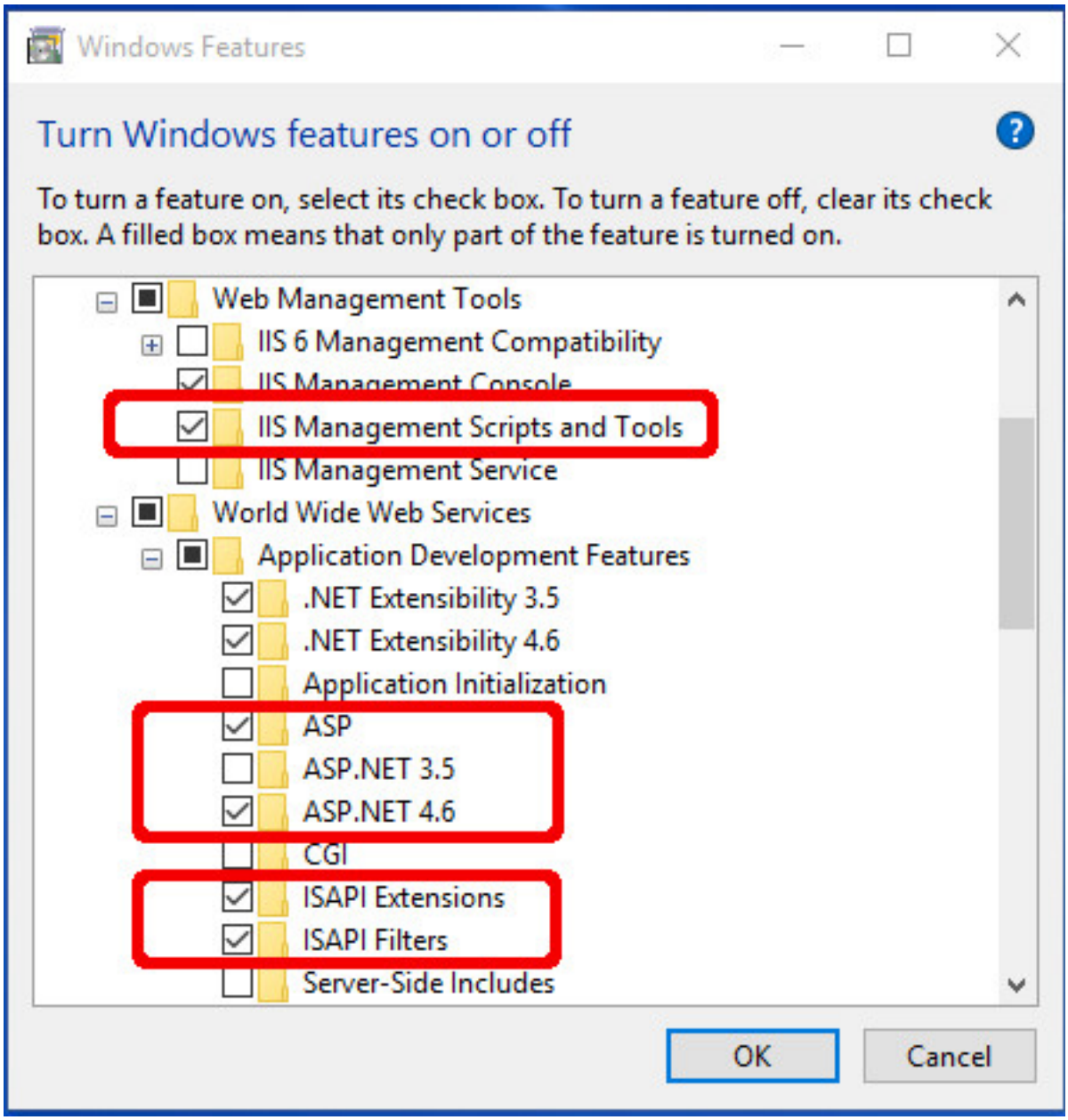


Figure 23: Turn Windows™ features on or off dialog

Configuring IIS 8.x and IIS 10.x application pools

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

To create an application pool:

1. In IIS manager, right-click **Application Pools** and click **Add Application Pool...**.

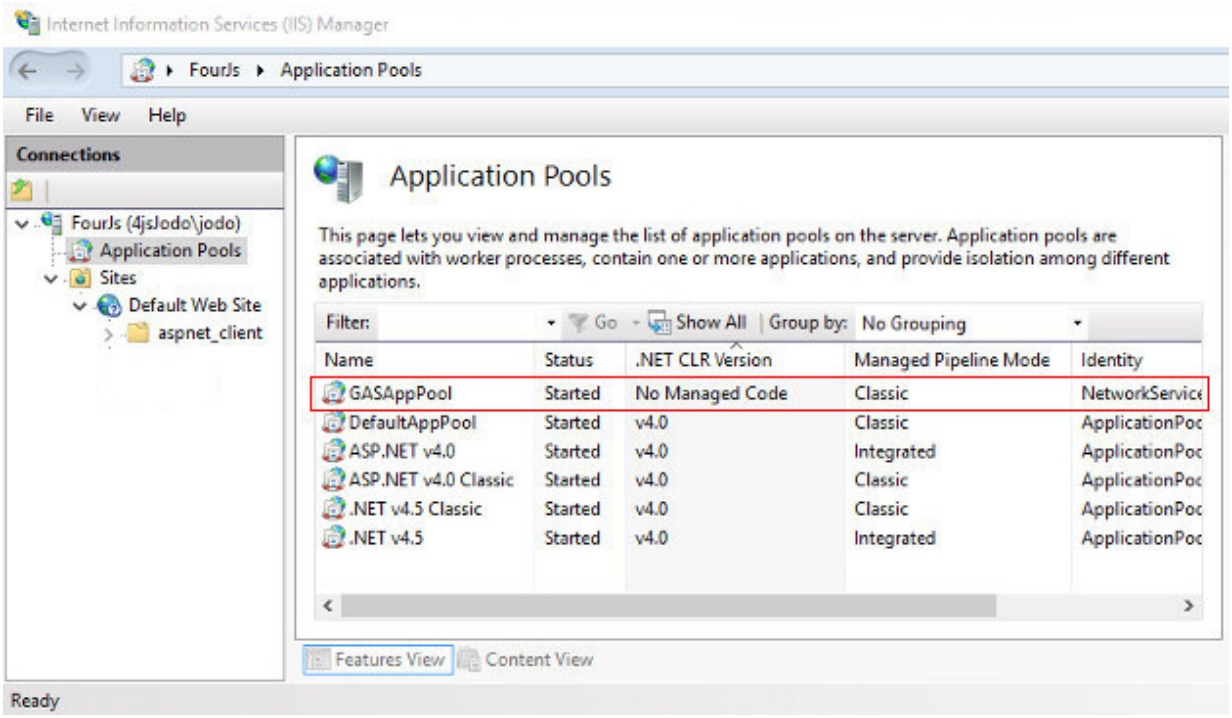


Figure 24: Internet Information Services (IIS) Manager Application Pools screen

2. In the **Add Application Pool...** dialog, enter a name for the application pool, for example "GASAppPool".
3. In the **.NET CLR version** box, select No Managed Code.
4. In the **Managed pipeline mode** box, select Classic.
5. Click **OK**.

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

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

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

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

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

Configuring an IIS 8.x and IIS 10.x application

To create an application:

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

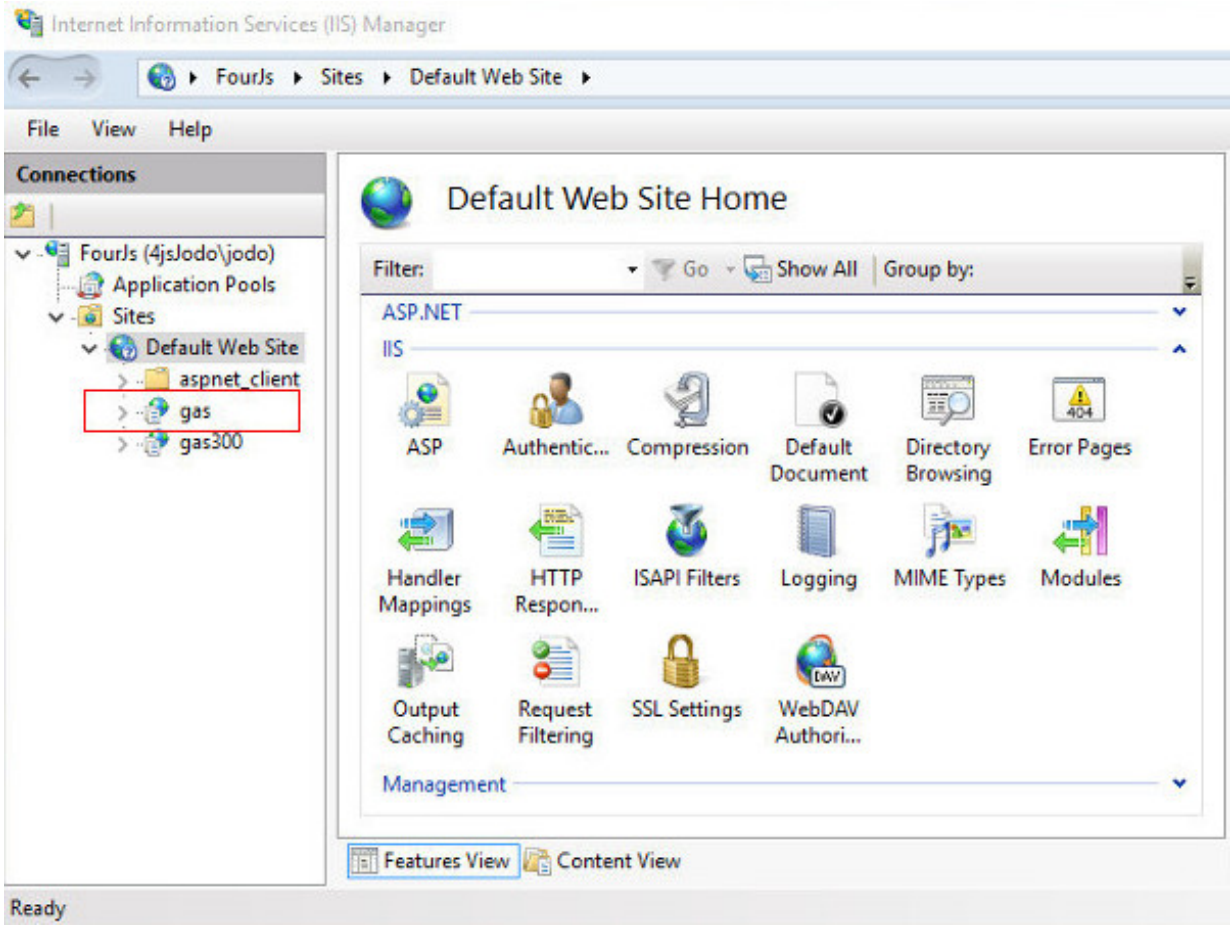


Figure 25: Internet Information Services (IIS) Manager Default Web Site Home screen

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

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

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

7. Click **OK**.

To bind the application to the GAS ISAPI Extension:

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

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

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

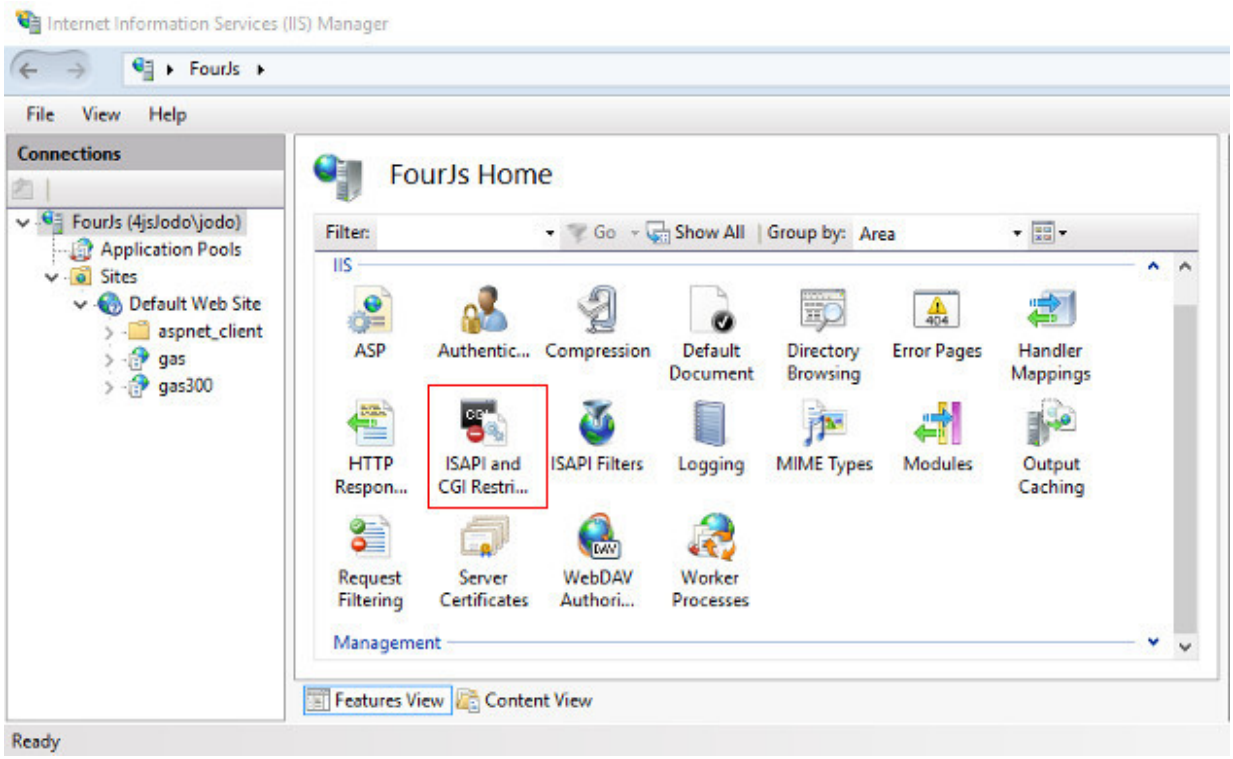


Figure 26: Internet Information Services (IIS) Manager Connections root node screen

3. In the **ISAPI and CGI Restrictions** feature, in the **Actions** area, click **Add...**
4. In the **Add ISAPI or CGI Restriction** dialog, enter the path to the GAS ISAPI Extension DLL: *FGLASDIR*`\bin\isapidispatch.dll`.

5. Enter a description, for example "GAS ISAPI Extension".
6. Ensure that the **Allow extension path to execute** checkbox is checked.
7. Click **OK**.

Post requisites

After you have finished the installation, you now need to configure the GAS ISAPI Extension configuration file, see Finishing the installation on page 83 .

Finishing the installation

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

Prerequisites

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

To finish the installation:

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

   See GAS ISAPI Extension configuration file on page 83 for details about the content of this file.
2. The GAS ISAPI Extension should be ready to use. Open a web browser and enter the URL to the "demos.html" page. The URL should look like:

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

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

   ```
   http://localhost/gas/demos.html
   ```

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

   Note:

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

GAS ISAPI Extension configuration file

The GAS ISAPI Extension configuration file is an (`ini`) file called `isapidispatch.ini`. It contains several options that the GAS ISAPI Extension uses when starting up. The `isapidispatch.ini` file is created during the installation; you must copy this file into the ISAPI Extension root directory.

This file is read on startup only. If the file is modified, the GAS ISAPI Extension has to be restarted before the changes are recognized.

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

Table 9: GAS ISAPI Extension sections and properties

Section	Property	Default value	Required	Comment
Options	as-directory	N/A	YES	The FGLASDIR directory
Options	configuration-file	FGLASDIR\etc \as.xcf	NO	The GAS configuration file

Sample `isapidispatch.ini`

```
# GAS ISAPI Extension configuration file
...
[Options]
# The GAS installation directory.
# This option is required.

as-directory=C:\FourJs\gas

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

#configuration-file=
```

Troubleshooting installation

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

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

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

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

The message displayed could help to find out what part of the start up process has failed. See GAS ISAPI Extension configuration file on page 83.

In all other cases, check the installation process.

Restarting the ISAPI dispatcher

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

Please consult your IIS documentation for details.

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

FastCGI Installation and Web Server Configuration

How to configure the FastCGI extension for various Web Servers.

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

GAS supports mod_fastcgi but not mod_fcgid.

Using the FastCGI dispatcher

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

The main advantages of FastCGI include:

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

FastCGI extension manages the GAS dispatcher process:

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

GAS FastCGI support is provided by using the `fastcgidispatch` executable, see Dispatcher: fastcgidispatch on page 265).

FastCGI GAS configuration on various Web Server

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

A typical URL would be:

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

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

Apache: mod_fastcgi

Module mod_fastcgi is the Apache module for FastCGI support.

> **Note:** Assume the Genero Application Server is installed in the following directory (`FGLASDIR`): `/opt/gas`. Make the appropriate substitution for the `FGLASDIR` when applying these examples to your own configuration.
>
> **Important:** Apache 2.4 does not support mod_fastcgi. If you are using Apache 2.4, use mod_proxy_fcgi instead, see Apache 2.4: mod_proxy_fcgi on page 87.
>
> **Note:** For more information on the Apache module mod_fastcgi, please refer to the Apache documentation.

mod_fastcgi installation

Install software package for your system or use these instructions.

mod_fastcgi configuration to manage GAS process

Add these lines to your Apache configuration file:

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

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

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

In the fastcgi configuration example:

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

Using mod_deflate for compression with mod_fastcgi

Normally on Apache web server, compression is enabled through the mod_deflate module. You should be aware that there is a known bug (see https://bugs.launchpad.net/ubuntu/+source/libapache-mod-fastcgi/+bug/381384) resulting in incorrect content-length header being returned when pages are loaded as the content-length is not updated after compression. You may begin, therefore, to notice a slow down in your applications as:

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

Moreover as GAS does compression by default (see Compression in Genero Application Server on page 146), mod_deflate's functionality is really unnecessary. It is therefore recommended that you disable mod_deflate. For more information on disabling mod_deflate, please refer to the fastcgi documentation or contact your system administrator.

Apache 2.4: mod_proxy_fcgi

With Apache 2.4, mod_proxy_fcgi is used instead of mod_fastcgi.

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

Apache 2.4 does not officially support mod_fastcgi, use mod_proxy_fcgi instead. This module requires Genero Application Server 2.50 (or later).

> **Note:** For more information on the Apache Module mod_proxy_fcgi, please refer to the Apache documentation.

1. Find the `httpd.conf` file. For example, it might be located in `/etc/apache2/`.
2. Add the following lines to the `httpd.conf` file:

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

   ```
   ...
   <IfModule mod_proxy_fcgi.c>
     #No PATH_INFO with mod_proxy_fcgi unless this is set
     SetEnvIf Request_URI . proxy-fcgi-pathinfo=1

     ProxyPass /gas/ fcgi://localhost:6394/ enablereuse=on
     Alias /gas /opt/gas/bin/fastcgidispatch
   </IfModule>
   ...
   ```

   In this excerpt:

   - localhost is where the fastcgidispatch is running
   - 6394 is the por fastcgidispatch is listening to
3. Start fastcgidispatch in standalone mode with `fastcgidispatch -s`. If this dispatcher fails, it must be restarted manually.

Fastcgi for nginx

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

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

Edit the Web site configuration file (for example, located in `/etc/nginx/sites-enabled/default`).

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

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

In this excerpt:

- 127.0.0.1 is the ip where the fastcgidispatch is running.
- 6394 is the port where fastcgidispatch listens to.

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

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

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

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

In this excerpt:

- /gas is the gas connector alias.

In the `fastcgi_params` file, find the line that reads:

```
fastcgi_param SERVER_NAME $server_name;
```

and replace with:

```
fastcgi_param SERVER_NAME $host;
```

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

If this dispatcher fails, it must be restarted manually.

Lighttpd

The Lighttpd Web server supports natively FastCGI protocol.

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

For more information, see the ModFastCGI documentation provided by Apache.

ModFastCGI configuration to manage GAS

Add these lines to your Lighttpd configuration file:

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

```
)
```

Note:

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

Sun Java™ System Web Server 7.0
Sun Web Server 7 has an integrated FastCGI support.

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

For more information, see the FastCGI Plug-in documentation provided by Sun.

Enable FastCGI Plug-in

Add this line to your magnus.conf configuration file:

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

FastCGI Plug-in configuration to manage GAS

Add these lines in your obj.conf configuration file:

```
<Object name="default">
  ...
  NameTrans fn="assign-name" from="/gas/*" name="gas.config"
  ...
</Object>

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

Note:

- The "resp-timeout" must be greater than the <REQUEST_RESULT> timeouts in the GAS configuration.

Troubleshooting

Troubleshooting tips addressing common issues.

Why does my application timeout on Apache?

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

To avoid this, you should match the GAS timeout with the Apache timeout. For more information on setting the Apache timeout, see the Apache documentation https://httpd.apache.org/docs/2.4/mod/mod_reqtimeout.html.

Why does my application not work with fastcgi?

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

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

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

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

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

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

Examine two items:

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

Applications all down at the same time

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

- GAS logs (See Table 5: appdata directories and files on page 40 for details on the location of log files)
- Web server logs

One reason could be a `cron` or a script that periodically shuts down and restarts the web server. For example, in the Apache `log_error` file you can find messages like:

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

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

In order to archive the logs, `logrotate` asks Apache to restart gracefully, so it shuts down the fastcgi, the dispatcher and all Genero applications. You can check if it is really the culprit by using this command (or an equivalent one):

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

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

Restarting the FastCGI dispatcher

To restart fastcgidispatch, use:

```
kill -9
```

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

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

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

Java™ Servlet Installation and Web Server Configuration

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

Using the GAS Java™ dispatcher

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

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

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

The Java™ servlet dispatcher, **java-j2eedispatch**, is located in `$FGLASDIR/war`. It provides the same functionality as the other GAS dispatchers, see GAS Dispatchers on page 263.

URLs used to access the Java™ dispatcher depends on the port number configured for the Java™ server and the servlet name defined in the Java™ servlet deployment file (`web.xml`), but typically will have the following form:

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

For example:

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

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

- Building the Java Web Archive (WAR) on page 91
- Deploying on a Java Web Server on page 92

Building the Java™ Web Archive (WAR)

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

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

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

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

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

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

   Caution: Do not delete, rename, or make any modifications to `web.bak`. It may only be used to make a copy.
3. Execute the `fglgar` command in the *`$FGLASDIR`*`/war/WebContent` directory replacing your options for the settings for the GAS installation (`--asdir`) and configuration (`--asxcf`) for those shown in the example:

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

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

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

When you have completed building the Web archive in the above step, your next task is to deploy the new Java Servlet. This is detailed in Deploying on a Java Web Server on page 92.

Deploying on a Java™ Web Server

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

1. Using your browser, go to the Java™ Server manager web page.
2. In the deployment section, choose the **java-j2eedispatch.war** file (created previously, see Building the Java Web Archive (WAR) on page 91) to download.
3. Click on the deploy button.
4. Check that **java-j2eedispatch** is available in the list of web applications, with a status of success.
5. In another browser tab, check that you can load the GAS demos page.

   For example:

   - http://host:port/java-j2eedispatch/demos.html
   - http://host:port/java-j2eedispatch/ua/r/gwc-demo
   - http://host:port/java-j2eedispatch/ws/r/Echo?WSDL

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

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

Restarting the J2EE dispatcher

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

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

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

1. Using your browser, go to the Java™ Server manager web page.
2. To stop the `java-j2eedispatch` dispatcher, select the option to **undeploy** the `war` file
3. To restart, select the option to **deploy** the `war` file as detailed in Deploying on a Java Web Server on page 92.
4. Check that **java-j2eedispatch** is available in the list of web applications, with a status of success.
5. In another browser tab, check that you can load the GAS demos page.

   For example:

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

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

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

Validating configuration files

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

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

What is an XML Schema Definition file?

An XML Schema Definition (XSD) describes the structure of an XML document.

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

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

Why specify the XML Schema Definition file?

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

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

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

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

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

These references to an XML schema definition file are used by XML editors, in order to provide validation and syntax hints while editing the configuration files. The XML editor either looks on the Web or in its local schemas catalog for the specified XML schema definition file.

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

Validating with the gasadmin tool

Some options that can prove useful in validating the GAS configuration file with the `gasadmin` command.

`--configuration-check`

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

`--configuration-explode`

The configuration-explode option explodes the GAS configuration file into separate files, one for each application, which are then stored in the temporary directory defined by the `res.path.tmp` resource of the GAS configuration file (see GAS configuration file on page 289). Each file lists the entire configuration for an application, expanding the inherited components.

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

`--configuration-expand-resources`

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

```
<EXECUTION>
<ENVIRONMENT_VARIABLE Id="FGLDIR">C:\4js\fgl</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="GREDIR">#!GREDIR!#</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="PATH">C:\Windows;C:\4js\gas\2.50.34\bin;#!GREDIR!#
\bin;C:\4js\fgl\bin;C:\4js\fgl\lib</ENVIRONMENT_VARIABLE>
<ENVIRONMENT_VARIABLE Id="FGLLDPATH">C:\4js\gas\2.50.34\lib;#!GREDIR!#
\lib;C:\4js\fgl\lib</ENVIRONMENT_VARIABLE>
<PATH>C:\4js\fgl/demo/WebServices/calculator/server</PATH>
...
</EXECUTION>
```

Selecting an XML editor

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

With a good XML editor, you can:

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

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

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

Configuring applications on GAS

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

> **Note:** For examples of how to get started with configuring, running, and deploying basic types of applications on the GAS, please see GAS Quick Start Guide on page 15.

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

Application Configuration Overview

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

Much of this information is common across a set of applications. Rather than have you provide all the information each time you configure an application, Genero supports the concept of *inheritance*. You define abstract applications to hold the basic information that is common across your applications, and then you configure your application to inherit the settings of the abstract application. There is no limit to the levels of inheritance: an application can inherit from another application (abstract or not) that inherits from another application, and so on. To inherit a base configuration from another application, you specify the other application as the parent.

An abstract application is typically defined first. This abstract application is not executable. It is intended to provide the baseline default configuration for other applications to inherit. You can create as many abstract applications as you require. Abstract applications can inherit a default configuration from another abstract application.

When configuring an application that is to be an executable, you can either provide the configuration details in the GAS configuration file, or you can create a separate application-specific configuration, see Create an application configuration file on page 98 (one per application).

When you add the application to the GAS configuration file, you must restart the GAS for the application to be recognized. When you create an external application configuration file and add the file into a defined GROUP directory, see Creating an application Group on page 96. The application is immediately available without having to do a GAS restart.

Creating Abstract Applications

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

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

> **Tip:** If you use this inheritance mechanism efficiently, you can configure new applications with only a few entries in the configuration file.
>
> **Important:** Abstract applications can only be defined in the GAS configuration file. They cannot be defined using an external application configuration file.

Default Genero Web Client abstract application

The default application for the Genero Web Client (GWC) is found in the Genero Application Server configuration file.

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

1. This application inherits the configuration of the `defaultwa` abstract application, also defined in the GAS configuration file.
2. To specify an abstract application, set the *Abstract* attribute to **TRUE**.
3. The *OUTPUT Rule* identifies which Front End is used to display the application, as set in the `defaultwa` abstract application. In this case the front end target is the Web client, GWC-JS.

Example for Web services abstract application

```
<APPLICATION Id="ws.default" Abstract="TRUE">
  <EXECUTION Using="cpn.ws.execution.local"/>
  <TIMEOUT Using="cpn.ws.timeout.set1"/>
</APPLICATION>
```

Creating an application Group

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

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

Syntax

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

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

Usage

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

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

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

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

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

The default group

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

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

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

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

The resource `$(res.path.app)` resolves to *appdata*`/app`, *appdata* is described in GAS directories on page 38. In this directory, you would expect to find `Edit.xcf`, the Edit application's application configuration file.

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

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

Example 2: "demo" group defined by resource

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

This example assigns the alias `demo` to the directory containing the external application configuration files for demo applications. The path is defined using the resource `$(res.path.demo.app)`. By wisely using a resource, a change to the directory structure only requires a change to a single RESOURCE element in the configuration file.

To access an application that has its configuration file stored in the group directory, enter an application URL that includes the group alias in its path: `http://server/gas/ua/r/demo/CardStep1`

Based on this URL, the GAS would expect to find the configuration file `CardStep1.xcf` within the directory specified for the `demo` group.

Create an application configuration file

An application configuration file provides the Genero Application Server with the information needed to run an application. It becomes available for use as soon as it is created and added to a recognized group directory.

Once you have created an application, you need to configure it so that it can be executed by the Genero Application Server. For this you need to create an application configuration file. Typically, the name of the file matches the name of the application and has an `xcf` suffix. For example, if the application name was "app1", create a configuration file named `app1.xcf`.

Save the file in a defined GROUP directory. By default, the directory where the GAS searches for external application configuration files is defined in the GAS configuration file (default `as.xcf`) by the tag `<GROUP Id="_default">`*`directory`*`</GROUP>`. You can specify alternate directories; see GROUP (for an application) on page 318 or GROUP (for a service) on page 319.

The configuration file defines an application environment, and starts with the `Application` element. Within this element, you can define local resources, change the execution environment, the timeout settings, the image, and output settings. You can refer to previously defined components by using the tag attribute `Using`.

The organization of the elements within the application configuration file depend on the type of application. See Application configuration files on page 289.

These examples show some well-formed external application configuration files.

Example 1 - A simple application configuration file

The simplest application configuration file specifies a parent application and the path to the compiled application files. The application inherits the configuration of the parent application. The file is named *appname*`.xcf`, where *appname* is the name of the application.

```
<APPLICATION Parent="defaultgwc">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
  </EXECUTION>
</APPLICATION>
```

In this example, the external application configuration file `Edit.xcf` provides a configuration for the Edit application. Things to note:

1. The application name is the name of the configuration file. The GAS uses the configuration file name to identify the application. The `Id` attribute of `<APPLICATION>` element is omitted; even if included, its value is not read.
2. The application configuration file is re-read at each application launch. There is no need to restart the GAS after modifying an application configuration file.
3. In this example, the *Parent* application is `defaultgwc`. A parent application is an abstract application that must be defined in the GAS configuration file. It provides default component configurations, which applications can inherit. See Creating Abstract Applications on page 96.
4. The path to the application executables is defined by the *PATH* component. See PATH (under EXECUTION) on page 332.
5. The `MODULE` element can specify the name of the `.42r` module to run. If the module name is the same as the configuration file name, the `MODULE` element is not necessary, the module name used is the name of the application. See MODULE on page 329.

Example 2 - Specifying a custom GWC-JS directory

While an application inherits its base configuration from the parent application, additional configuration elements can be added and existing configuration elements can be overwritten.

This next example configuration file is for a GWC-JS Web application.

```
<APPLICATION Parent="defaultgwc">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
  </EXECUTION>
  <!-- Override the default gwc-js with gwc-js-custom-->
  <UA_OUTPUT>
     <PROXY>$(res.uaproxy.cmd)</PROXY>
     <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
     <GWC-JS>gwc-js-custom</GWC-JS>
     <TIMEOUT> Using="cpn.wa.timeout"</TIMEOUT>
  </UA_OUTPUT>
 </APPLICATION>
```

1. The UA_OUTPUT on page 354 element provides configuration details for applications rendered by the UA proxy.
2. The GWC-JS on page 319 element provides the directory name gwc-js-custom for the customized project, for more information on customization see Configuring your Environment on page 198.

Example 3 - Specifying the use of the HTML5 theme (and other overrides)

This next example is of a hypothetical external application configuration file, tutorialStep1.xcf, for a GWC-HTML5 Web application.

> **Note:** The GWC-HTML5 Web client has been deprecated; new development should use the GWC-JS instead.

```
<APPLICATION Parent="demo-tut-abstract">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/
cfextwa.xsd">
  <!-- Define a resource to the template HTML file -->
  <RESOURCE Id="res.template.tutorial"
    Source="INTERNAL">$(res.path.demo.dem-tut)/web/tutorial/
tutorialStep1.html
  </RESOURCE>
  <EXECUTION>
    <PATH>$(res.path.demo.dem-tut)/src</PATH>
    <MODULE>tutStep1.42r</MODULE>
  </EXECUTION>
  <!-- Override default rendering template -->
  <OUTPUT>
    <MAP Id="DUA_HTML5">
      <THEME>
        <TEMPLATE Id="_default">$(res.template.tutorial)</TEMPLATE>
      </THEME>
  <OUTPUT>
 </APPLICATION>
```

1. The Parent attribute of the APPLICATION element defines the parent application as "demo-tut-abstract". The child application inherits the configuration elements defined by the parent application.
2. The RESOURCE element defines a local resource. This resource maps to a template file. See RESOURCE (for an application) on page 343.

3. The PATH element lists the path to the executable. See PATH (under EXECUTION) on page 332.
4. The MODULE elements provide the path and file name of the program executable. The MODULE element is often excluded, when the executable name matches the application name as provided in the URL. In this example, had the external application configuration file been named "tutStep1.xcf", then the MODULE element could have been excluded. See MODULE on page 329.
5. The Id attribute of the MAP element defines the output map as DUA_HTML5. This means that the application will use the deprecated GWC for HTML5 theme. See MAP on page 327.
6. The TEMPLATE element overrides the default template (Id="_default") with the template defined by the resource "$(res.template.tutorial)". Recall that this resource was defined at the start of this file using a RESOURCE element. See TEMPLATE on page 350.

Using External Application Configuration Files

To configure an application with an external application configuration file, you provide the same configuration details that you would for adding the application directly in the Genero Application Server configuration file (as.xcf), however you write the configuration XML in a separate file, where the file name matches the name of the application.

For example, to create an external application configuration file for a program named **gwc-demo**, you would:

1. Add a file named gwc-demo.xcf. You must use the xcf suffix.
2. Place the file in a GROUP directory, as defined in the Genero Application Server configuration file.

- Configuring Web client applications on page 100
- Configuring applications for Web service on page 101
- Configuring GDC applications on page 102

Configuring Web client applications

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

To add an application for GWC-JS, you need to specify:

- An application Id (a unique name for this APPLICATION element)

  Note: Applications defined in the GAS configuration file require an **Id** attribute. For external configuration files, if the application and the configuration file share the same name, there is no need to specify the **Id** attribute.
- The parent application from which to inherit configuration details (defaultgwc in this example)
- The path to the compiled application files
- The name of the application to launch
- The access control allowing access (optional)
- The customization project directory to use for the application user interface look and feel (if using a customized project)

Example: simple application for GWC-JS Web client

```
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

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

Example: gwc-demo-external.xcf

The main differences between this example and the example shown in Example: simple application for GWC-JS Web client on page 100 are the lack of the **Id** attribute and the reference to the XML schema.

```
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
        <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
  <UA_OUTPUT>
     <PROXY>$(res.uaproxy.cmd)</PROXY>
     <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
     <GWC-JS>gwc-dev</GWC-JS>
     <TIMEOUT> Using="cpn.wa.timeout"</TIMEOUT>
  </UA_OUTPUT>
</APPLICATION>
```

1. The `ALLOW_FROM` on page 300 element specifies from what hosts access is allowed, the example here is defined in a RESOURCE.
2. The `GWC-JS` on page 319 configuration element, specifies the customization project directory you used to provide the application look-and-feel. See Customization for GWC-JS applications on page 189.

Configuring applications for Web service

Create a separate application configuration (`xcf`) file for each Web services application.

What do you need to configure a Genero Web Service application?

To add an application for a Genero Web Service, you need to specify:

- Your application Id

  Note: Applications defined in the GAS configuration file require an **Id** attribute. For external configuration files, if the application and the configuration file share the same name, there is no need to specify the **Id** attribute.
- The parent application where the main configuration is set (in this example, **ws.default**)
- The path to your compiled files
- The main module to launch
- The access control allowing access (optional)
- The number of DVMs (`fglrun`) to start for this Web Service when the GAS starts, and the minimum and maximum number of DVMs allowed.

Example: simple Web service application

In the following example the configuration is for a Web service defined in the GAS configuration file. The *PATH* is a resource. The path can also be an absolute path to your application files. This configures a GWS server that any Web service client can connect to.

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

1. The application inherits the configuration settings of its parent ("`ws.default`" in this example).
2. The path used in this example references a `RESOURCE` root for demo applications; you could also use the absolute path name leading to your application files.

Example: Web Service Calculator.xcf

In the following example, if the file was named "Calculator.xcf, then this configuration file would accomplish the same task as when included in the GAS configuration file as in the example in Example: simple Web service application on page 101. The main differences are the lack of the **Id** attribute and the reference to the XML schema.

> **Note:** Because a DVM can have several services defined in it, the Web Service DVM is an application. The services defined inside are still named service. The published functions are named operations.

```
<APPLICATION Parent="ws.default"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/
cfextws.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/WebServices/calculator/server</PATH>
    <MODULE>calculatorServer</MODULE>
    <ACCESS_CONTROL>
        <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
    </ACCESS_CONTROL>
    <POOL>
        <START>0</START>
        <MIN_AVAILABLE>0</MIN_AVAILABLE>
        <MAX_AVAILABLE>1</MAX_AVAILABLE>
    </POOL>
  </EXECUTION>
</APPLICATION>
```

1. The `ALLOW_FROM on page 300` element specifies from what hosts access is allowed, the example here is defined in a RESOURCE.
2. The `POOL on page 337` element specifies the number of DVMs to start for this Web Service when the GAS starts. In this example zero DVMs at start means the Web service is not set to start with the GAS. And the maximum allowed is one DVM.

This example file can be found in *`$FGLDIR`*`/demo/WebServices`. In conjunction with the Example 2: "demo" group defined by resource on page 97 definition, to access the WSDL of this demo, you can use this kind of URL:

```
http://appserver:6394/ws/r/demo/calculator?WSDL
```

See Accessing the Web Service (Web Services URI information) on page 214.

Configuring GDC applications

What do you need to configure a Genero Desktop Client (GDC) application?

To add an application for GDC, you need to specify:

- An application Id (a unique name for this `APPLICATION` element)

  > **Note:** Applications defined in the GAS configuration file require an **Id** attribute. For external configuration files, if the application and the configuration file share the same name, there is no need to specify the **Id** attribute.
- The parent application from which to inherit configuration details ("`defaultgdc`" in this example)
- The path to the compiled application files
- The name of the application to launch
- The access control allowing access (optional)

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

```
<APPLICATION Id="my-app" Parent="defaultgdc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

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

Example: gdc-demo-external.xcf

This external configuration file would accomplish the same task as the Example: simple configuration for GDC application on page 103 that is defined in the GAS configuration file. The only differences are the lack of the **Id** attribute and the reference to the XML schema.

```
<APPLICATION Parent="defaultgdc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
    <ACCESS_CONTROL>
         <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
    </ACCESS_CONTROL>
  </EXECUTION>
</APPLICATION>
```

The `ALLOW_FROM` `on page 300` element specifies from what hosts access is allowed, the example here is defined in a RESOURCE.

Configure DVM environment variables

The application configuration file can be used to define environment variables for the DVM.

Environment variables are set with `ENVIRONMENT_VARIABLE` elements in the application configuration file.

Syntax

```
<ENVIRONMENT_VARIABLE Id="env_var">env_value</ENVIRONMENT_VARIABLE>
```

1. *env_var* is the environment variable name.
2. *env_value* is the value used to set the variable name.

Example

Example (using Informix® database client):

```
<APPLICATION Id="myapp" Parent="defaultgwc">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="DBDATE">DBMY4/</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="FGLRESOURCEPATH">/home/myapp/resources</
ENVIRONMENT_VARIABLE>
```

```
    <ENVIRONMENT_VARIABLE Id="INFORMIXDIR">/opt/informix</
ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="INFORMIXSERVER">ORION</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="LD_LIBRARY_PATH">/opt/informix/lib:...</
ENVIRONMENT_VARIABLE>
    <PATH>/home/myapp/bin</PATH>
    <MODULE>app.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

For more details, see ENVIRONMENT_VARIABLE on page 314.

Use a script to set the environment

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

Alternatively to the ENVIRONMENT_VARIABLE elements, you can use the DVM element to define a command that will set the environment and launch the application.

On UNIX platforms, use the DVM element to define a shell command that will execute a shell script defined with the MODULE element:

```
<APPLICATION Id="kiosk" Parent="defaultgwc">
  <EXECUTION>
    <PATH>/home/f4gl/gep/configfiles/officestoredemo</PATH>
    <DVM>/bin/sh</DVM>
    <MODULE>gdc-kiosk.sh</MODULE>
  </EXECUTION>
</APPLICATION>
```

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

On Windows platforms, use the DVM element to define a .BAT command file, including environment settings and program execution:

```
<APPLICATION Id="myprog" Parent="defaultgdc">
  <EXECUTION>
    <PATH>$(res.fgldir)/demo</PATH>
    <DVM>c:\myprog\launch.bat</DVM>
  </EXECUTION>
</APPLICATION>
```

1. PATH element defines where the script is stored.
2. DVM element defines the BAT command to be executed.

What if the application doesn't start?

What to do when you request an application and it does not start.

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

- Check your environment variables in *$FGLASDIR*/etc/as.xcf.
- The Genero Application Server creates separate log files for its dispatchers, proxies, and the DVMs started by those proxies. Examine the logs as they may provide you with some helpful information or error messages. For more information about accessing log files see Logging on page 156
- To troubleshoot and debug an application, you may need to run it in debug mode using the FGL debugger, i.e. fglrun -d. See Using the debugger on page 157.

Note:

- You can use the graphical debugger in Genero Studio. For more information, see the *Genero Studio User Guide*.
- The debug facility of the Genero Desktop Client includes logging and the debug console. For more information on using the GDC debug facility, see the *Genero Desktop Client User Guide*.
- For details about debugging GWC-JS applications, see Configuring development environment on page 147.

When you receive the Error: Runtime error. Try again ... page

Simply put, your application cannot start and you must check your application configuration. This error is typically the result of an incorrect path to the program executable.

Next steps

Other topics provide more details on application configuration and deployment.

See also:

- Developing Web applications on page 173
- Adding a Web Services Application
- Quick start: deploying applications on page 218
- Application List Reference

How to implement delegation

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

How delegation works

This section gives details about the delegation process.

Following steps are performed when an delegation occurs:

1. An application or web service start is requested. The type of request is defined by the `/ua/r`, `/wa/r`, `/ja/r`, or `/ws/r` path segments in the URI.
2. The Genero Application Server dispatcher passes the request to the REST service identified as the *delegation service*. The delegation service is specified in the application's configuration. The delegation service is written in Genero and managed by the Genero Application Server as a standard REST Web service. The delegation service should reside on the same GAS as the application.
3. The REST service instructs the Genero Application Server to either:
   - Refuse the start of the application or service.
   - Allow the start the application. The delegate service is able to add some environment variables to give additional information to the allowed application.
   - Allow for a service any request forwarded to it. All Web services requests are in `/ws/r` so they all go to the delegate service.

The delegation REST service refuses the start of the application or service

In this scenario, the delegation REST service refuses the start of the application or service. The REST service communicates with the user agent in HTTP using the `com.HTTPServiceRequest` object. For example, it could return an XHTML form asking for a user name and password to grant access to the application or service.

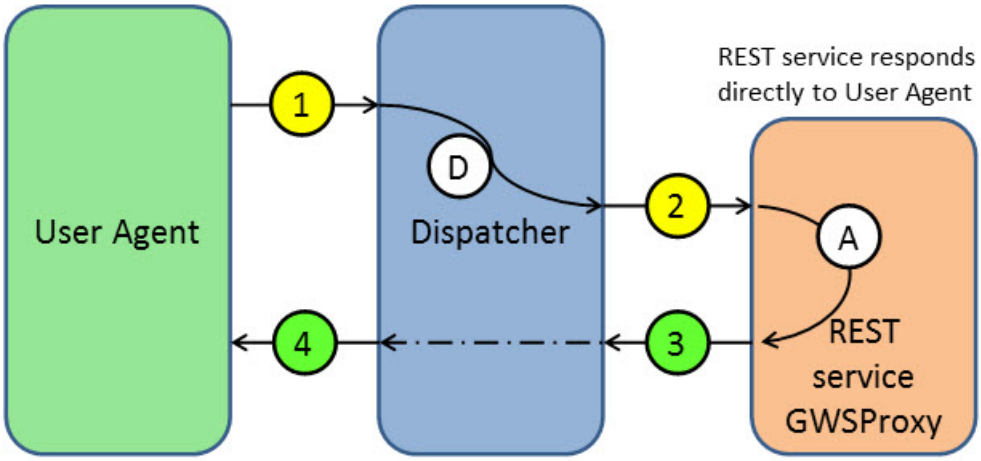


Figure 27: REST service communicates with User Agent but does not start the application or service

In this figure:

- (1) Start a new application or service of the form `/wa/r`, `/ja/r` or `/ws/r`.
- (D) Delegate application or service start to the service described in the configuration file (`xcf`).
- (2) Forward the request to REST service for delegation process
- (A) REST program responds directly to the User Agent (with a login page, for instance).
- (3) (4) Response is sent back to the User Agent.

The delegation REST service allows the start of the application or service

In this scenario, the delegation REST service allows the start the application or service as if launched directly from the user agent.

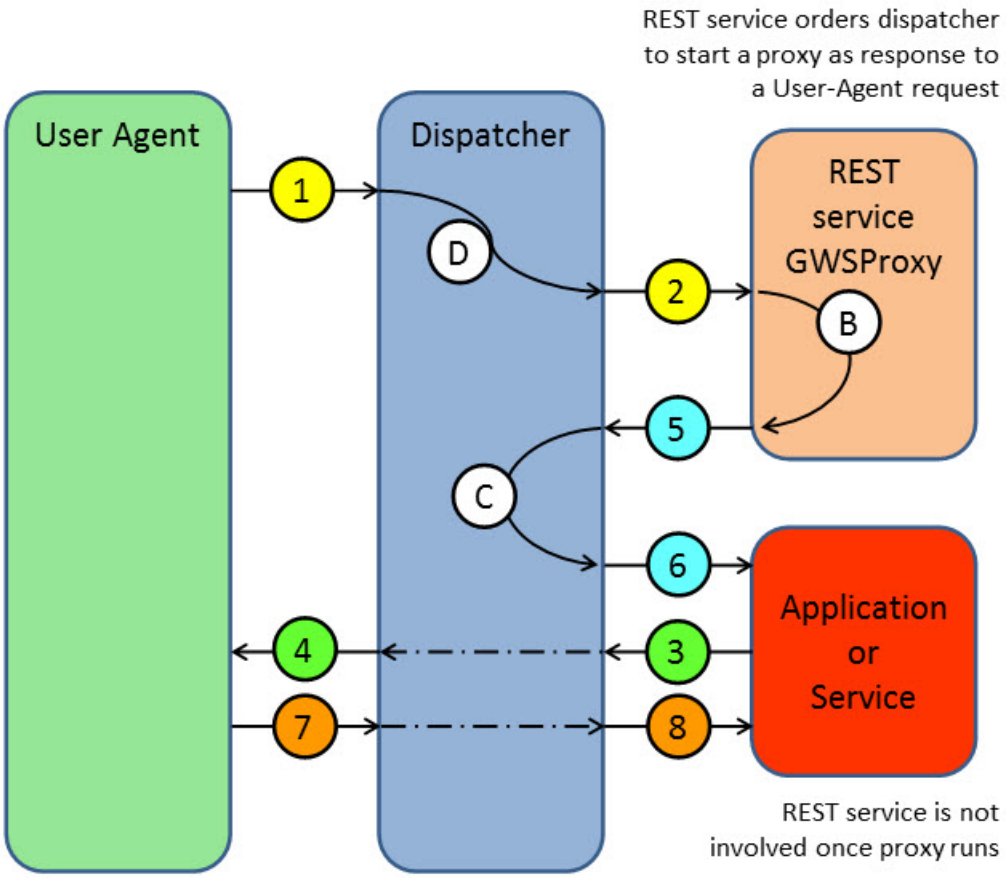


Figure 28: REST service approves application or service start

In this figure:

- (1) Start a new application or service of the form `/wa/r`, `/ja/r`, or `/ws/r`.
- (D) Delegate application or service start to the service described in the configuration file (`xcf`).
- (2) Forward request to REST service for delegation process
- (B) Genero REST program allows to start the proxy (login and password are correct, for instance).
- (3) (4) Response is sent back to the User Agent.
- (5) REST program response with HTTP code 307 and description string.
- (C) Dispatcher detects REST command and starts the proxy
- (6) Dispatcher forwards response from REST program to proxy
- (7) (8) Any request is forwarded to the proxy without going to the REST service (excepted for GWS that starts a new delegation process).

To allow the application to start, the Genero REST service returns a specific HTTP code and description to the dispatcher using the `com.HTTPServiceRequest` object. When the dispatcher gets such an HTTP response from the REST service, it starts a new proxy and forwards the request to it, as if no delegation had taken place.

Configure delegation for application or service

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

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

You can define optional parameters for the REST service to be sent each time a starting request is received. No validation is made for these optional parameters, the REST service must check them and return an error when necessary.

> **Note:** A starting request is a URL with `/r`. When you see an application URI with `/sua`, the application has been validated and the delegation REST service is no longer involved. For more information see Application URIs on page 49

Delegate configuration example

```
<EXECUTION>
  <PATH>$(res.path)</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="MyGroup/MyDelegateService">
    <AnyParameter>MyFirstParameter</AnyParameter>
    <Other>MySecondParameter</Other>
  </DELEGATE>
</EXECUTION>
```

Parameters defined in the `DELEGATE` configuration, such as `AnyParameter` and `Other` in this example, are transmitted to the REST service. See Passing parameters to the REST service on page 108 for details on receiving the parameters in the REST service.

From the user agent to the REST service

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

REST service entry point

When a `/ua/r`, `/ja/r` or `/ws/r` request is delegated to the REST service, the dispatcher appends the string `/Delegate` to the service URL in order to distinguish a dispatcher delegation from any other standard REST request. In other words, if an application has delegation, the REST service is called with a `/Delegate` appended in the URL.

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

Passing parameters to the REST service

If parameters are defined in the `DELEGATE` configuration, they will be transmitted to the Genero REST service at each `/wa/r`, `/ja/r` and `/ws/r` request as HTTP headers.

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

REST sample:

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

```
...
LET param1 = req.getRequestHeader("X-FourJs-Environment-Parameter-
AnyParameter")
DISPLAY param1 # Displays MyFirstParameter
LET param2 = req.getRequestHeader("X-FourJs-Environment-Parameter-Other")
DISPLAY param2 # Displays MySecondParameter
...
```

The sample is based on this configuration:

```
<EXECUTION>
  <PATH>$(res.path)</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="MyGroup/MyDelegateService">
    <AnyParameter>MyFirstParameter</AnyParameter>
    <Other>MySecondParameter</Other>
  </DELEGATE>
</EXECUTION>
```

Passing the user agent URL to the REST service

When a `/wa/r`, `/ja/r` or `/ws/r` request is delegated to a REST service, the original URL is transmitted to the service in the URL query string of the request.

For example, if the user types the following original URL in a browser:

```
http://host:port/ua/r/MyGrp/MyApp?P1=1&P2=2
```

The resulting URL passed to the delegation service will be:

```
http://localhost:port/ws/r/MyGroup/MyDelegateService/Delegate?
http://host:port/ua/r/MyGrp/MyApp?P1=1&P2=2
```

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

```
IMPORT com
...
DEFINE req       com.HTTPServiceRequest
DEFINE original  STRING
DEFINE url       STRING
DEFINE query     STRING
...
LET original = req.readFormEncodedRequest(false)
CALL SplitUrl(original) RETURNING url, query
DISPLAY url      # http://host:port/ua/r/MyGrp/MyApp
DISPLAY query    # P=1&P=2
...
```

If the service must also handle non-delegated requests, use the `getURL()` method to retrieve the REST operation to perform. If the REST URL contains the `"/Delegate"` string, it is a delelation request and you need to extract the original URL after the ? character. You can then check the original URL the user agent wants to access, and extract potential parameters in its query string:

```
IMPORT com
...
DEFINE req       com.HTTPServiceRequest
DEFINE url       STRING
DEFINE rest_url  STRING
DEFINE original  STRING
DEFINE orig_url  STRING
```

```
DEFINE query     STRING
...
LET url = req.getURL()
CALL SplitUrl(url) RETURNING rest_url, original
CASE
   WHEN rest_url.getIndexOf("/Delegate",1)
        CALL SplitUrl(original) RETURNING orig_url, query
        DISPLAY orig_url # http://host:port/ua/r/MyGrp/MyApp
        DISPLAY query    # P=1&P=2
   WHEN rest_url.getIndexOf("/GetCurstomerInfo",1)
        # Handle regular REST request
        ...
   OTHERWISE
        CALL req.sendTextResponse(400, NULL, "Invalid REST request")
END CASE
...
```

From the REST service to the proxy

The delegation REST service must notify the dispatcher when it approves the start of an application or service.

Approve the proxy start

To approve the start of an application or a service proxy or the service forwarding (because if it is a web service using delegation, the service may be already started), the Genero REST service must return the following HTTP code and description: The HTTP return code must be `307` and the description must be the string `_GENERO_INTERNAL_DELEGATE_`.

Returning this HTTP code and description notifies the dispatcher to start the application or service proxy as the response to the current user-agent request.

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

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

Passing parameters to the proxy

When you need to pass additional parameters to a starting proxy, you can do it via environment variables that are then set in a gwc or gdc proxy environment before the dispatcher starts it. Each parameter must be set in the HTTP header response when you specify the `307` HTTP return code. The HTTP header name must be of the form `X-FourJs-Environment-`*XXX* where *XXX* is the name of the variable to pass to the proxy and as HTTP header value, the value of the environment variable. Then, with a call to the Genero function `fgl_getenv()`, you can retrieve them in your Genero program.

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

REST sample:

```
IMPORT com
DEFINE req com.HTTPServiceRequest
...
CALL req.setResponseHeader("X-FourJs-Environment-Hello","World")
CALL req.setResponseHeader("X-FourJs-Environment-Name","Georges")
CALL req.sendResponse(307,"_GENERO_INTERNAL_DELEGATE_")
```

```
...
```

Genero program sample:

```
MAIN
...
DISPLAY fgl_getenv("Hello")  -- Displays "World"
DISPLAY fgl_getenv("Name")   -- Displays "Georges"
...
```

REST service example

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

When the GWC application is started, the parameter `ACCESS` is also set and can be retrieved in the Genero program with `fgl_getenv("ACCESS")`.

```
IMPORT COM
IMPORT XML

PRIVATE CONSTANT C_BASEURL = "/ws/r/qa-test/delegateService/"
PRIVATE CONSTANT C_X_FOURJS_ENVIRONEMENT_ = "X-FourJs-Environment-"
PRIVATE CONSTANT C_X_FOURJS_ENVIRONEMENT_PARAMETER_ =
  "X-FourJs-Environment-Parameter-"
PRIVATE CONSTANT C_GENERO_INTERNAL_DELEGATE = "_GENERO_INTERNAL_DELEGATE_"

MAIN
  DEFINE  req   com.HttpServiceRequest
  DEFINE  methd STRING
  DEFINE  url   STRING
  DEFINE  path  STRING
  DEFINE  query STRING
  DEFINE  ind   INTEGER
  DEFINE  tmp   STRING

  CALL com.WebServiceEngine.Start()

  WHILE TRUE
    TRY
      LET req = com.WebServiceEngine.GetHttpServiceRequest(-1)
      IF req IS NULL THEN
        EXIT WHILE
      ELSE
        LET url = req.getUrl()
        DISPLAY "URL : ",url
        LET methd = req.getMethod()
        CALL SplitUrl(url) RETURNING path, query
        DISPLAY "Incoming request: ",methd," path=",path," query=",query
        LET ind = path.getIndexOf(C_BASEURL,1)
        IF ind<1 THEN
          CALL req.sendResponse(400,"Invalid request")
        ELSE
          CALL DispatchService(req,path.subString(ind+C_BASEURL.getLength(),
            path.getLength()),query)
        END IF
        DISPLAY "Sent response: ",methd," path=",path," query=",query
      END IF
    CATCH
      EXIT WHILE
    END TRY
  END WHILE
```

```
END MAIN

FUNCTION DispatchService(req,path,query)
  DEFINE  req   com.HttpServiceRequest
  DEFINE  path  STRING
  DEFINE  query STRING
  DEFINE  ind   INTEGER
  LET ind = path.getIndexOf("/",1)
  IF ind>0 THEN
    CALL req.sendResponse(400,"invalid path")
  ELSE
    CASE path
      WHEN "Delegate" # Handle a dispatcher delegate service
        CALL DelegateWA(req,query)
      OTHERWISE
        CALL req.sendResponse(400,"unknown service '"||path||"'.");
    END CASE
  END IF
END FUNCTION

#
# Delegate WA service
#  If browser URL doesn't contain 'ByPass' in query string then return a 404
 error,
#  otherwise start GWC application
#
FUNCTION DelegateWA(req,query)
  DEFINE  req               com.HttpServiceRequest
  DEFINE  query             STRING
  DEFINE  url               STRING
  DEFINE  queryString       STRING
  DEFINE  parameter         STRING
  DEFINE  ind               INTEGER
  DEFINE  ByPass            BOOLEAN
  DEFINE  q                 DYNAMIC ARRAY OF RECORD
                              qname   STRING,
                              qvalue  STRING
                            END RECORD
  # Read requests
  LET query = req.readFormEncodedRequest(false)
  IF query IS NULL THEN
    CALL req.sendResponse(400,"no query string")
  ELSE
    LET ByPass = FALSE
    CALL SplitUrl(query) RETURNING url,queryString
    CALL ParseQueryString(queryString) RETURNING q
    # Check if user-agent query string has a ByPass string ?
    FOR ind=1 TO q.getLength()
      IF q[ind].qname="ByPass" THEN
        LET ByPass = TRUE
      END IF
    END FOR
    IF NOT ByPass THEN
      # return error
      CALL req.sendResponse(404,"ByPass is missing")
    ELSE
      # Set parameter for GWC application via enviroment variable: ACCESS=OK
      CALL req.setResponseHeader(C_X_FOURJS_ENVIRONEMENT_||"ACCESS","OK")
      # Start application with HTTP code 307
      CALL req.sendResponse(307,C_GENERO_INTERNAL_DELEGATE)
    END IF
  END IF

END FUNCTION
```

```
#
# Parse given string and returns a dynamic array of the element inside the
#   query string
#
FUNCTION ParseQueryString(str)
  DEFINE  str   STRING
  DEFINE  tkz   base.StringTokenizer
  DEFINE  token STRING
  DEFINE  ind   INTEGER
  DEFINE  ret   DYNAMIC ARRAY OF RECORD
                  qname   STRING,
                  qvalue  STRING
                END RECORD
  INITIALIZE ret TO NULL
  LET tkz = base.StringTokenizer.create(str,"&")
  WHILE (tkz.hasMoreTokens())
    LET token = tkz.nextToken()
    CALL ret.appendElement()
    LET ind = ind + 1
    CALL ExtractKeyValueFromQuery(token) RETURNING
 ret[ind].qname,ret[ind].qvalue
  END WHILE
  RETURN ret
END FUNCTION

#
# Extract Key and Value from query string
#
FUNCTION ExtractKeyValueFromQuery(str)
  DEFINE str  STRING
  DEFINE ind  INTEGER
  LET ind = str.getIndexOf("=",1)
  IF ind>1 THEN
    RETURN str.subString(1,ind-1),str.subString(ind+1,str.getLength())
  ELSE
    RETURN str,NULL
  END IF
END FUNCTION

#
# Splits an URL into path and query string parts
#
FUNCTION SplitUrl(url)
  DEFINE  url   STRING
  DEFINE  ind   INTEGER
  DEFINE  query STRING
  DEFINE  path  STRING
  LET ind = url.getIndexOf("?",1)
  IF ind>1 THEN
    LET query = url.subString(ind+1,url.getLength())
    LET path = url.subString(1,ind-1)
  ELSE
    LET query = NULL
    LET path = url.subString(1,url.getLength())
  END IF
  RETURN path, query
END FUNCTION
```

Delegation use cases

Three examples of possible uses for the delegation mechanism.

Simple local authentication / authorization mechanism

You can develop a simple delegation service to authenticate and authorize users to have access to one or several applications on the Genero Application Server.

The delegation service will respond to the request with an HTML form, asking for a user name and password. In this case, HTTPS is required; otherwise the login and password will be sent in clear text.

A request with user name and password as parameters will be processed by the delegation service. The service will check for the user name and password in its database. If the user name and password are correct, a digest authentication will be created, stored in the database, and sent back to the user-agent in a cookie. The delegation service will instruct the user agent to delegate on the same URL (so the user agent will use its newly set cookie).

A request with a cookie will be processed by the delegation service. The cookie will be checked in the database. The corresponding user id, as well as the user role (administrator, user, guest, and so on), will be set as application parameters and the Genero Application Server will be instructed to allow the launch of the application.

A simple local authentication / authorization mechanism is provided in the Genero Application Server tutorials located at *$FGLDIR*`/web_utilities/services/simplesso`.

Authentication / authorization Single sign-on (SSO) mechanism

You can develop a delegation service to authenticate and authorize users to access one or more applications on the Genero Application Server based on standard SSO services such as OpenID Connect (see OpenID Connect SSO on page 115) or SAML (see SAML SSO on page 128).

The delegation service responds to a simple request for delegation to the SSO service, with reference to the requested application.

A request with a cookie will be processed by the delegation service. The cookie will be checked by the SSO service (by means specific to the SSO protocol). The corresponding user id and user role (as allowed by the SSO protocol) will be set as application parameters and the Genero Application Server will be instructed to allow the launch of the application.

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

Monitoring or logging requests for a Genero Web service

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

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

How to implement Single sign-on (SSO)

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

- How to implement custom single sign-on on page 138
- Connect to the application database with SSO on page 144

OpenID Connect SSO

OpenID Connect is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under `$FGLDIR/web_utilities/services/openid-connect/`.

OpenID Connect providers include Google and Microsoft. To learn more about OpenID Connect, see the OpenID Connect web site.

The Genero OpenID Connect solution is supported on to the GAS delegation mechanism. See What is delegation? on page 37.

OpenID Connect implementation creates a circle of trust between the Genero Application Server and an OpenId Connect provider. This method may vary depending on the IdP, but typically it consists of the following:

- Getting the OAuth2 PUBLIC and SHARED SECRET ID from the IdP to be used in Genero SSO applications, see Quick start: Set up OpenID Connect in the GAS on page 115
- Providing the IdP redirect URL of the GAS to the IdP, see Add OpenID Connect SSO to Genero Web application on page 117

Quick start: Set up OpenID Connect in the GAS

Follow these steps to quickly set up OpenID Connect for your Genero Application Server and Genero Web Client applications.

Before you begin, you must Configure GAS for OpenID Connect SSO on page 116.

In this quick start, you add OpenID Connect Single sign-on (SSO) to a Genero Web Client application, then execute the application with SSO.

1. Add OpenID Connect SSO to a Genero Web Client application requiring SSO.
   a) Add the `DELEGATE` element to all Genero Web Client applications requiring SSO

      The first three parameters are mandatory:

      - IDP : the provider of the IdP account (e.g. https://accounts.google.com)
      - CLIENT_PUBLIC_ID : the OAuth2 public ID provided by the IdP
      - CLIENT_SECRET_ID : the OAuth2 shared secret ID provided by the IdP
      - SCOPE : (optional) the OpenID Connect attributes you want to get from the user at time of authentication (e.g. email, phone, address).

```
<APPLICATION Parent="defaultgwc">
      <EXECUTION>
        <PATH>$(res.path.mypath)/myapplication</PATH>
        <MODULE> myapp.42r</MODULE>
        <DELEGATE service="services/OpenIDConnectServiceProvider">
          <IDP>https://accounts.google.com</IDP>
          <SCOPE>email</SCOPE>
```

```
          <CLIENT_PUBLIC_ID>XXXXXXXX.apps.googleusercontent.com</
CLIENT_PUBLIC_ID>
          <CLIENT_SECRET_ID>XXXXXX-XXXXXX</CLIENT_SECRET_ID>
        </DELEGATE>
      </EXECUTION>
    </APPLICATION>
```

2. Execute a Genero Web Client application with SSO.
   a) Start your browser and enter the application URL.
   You are prompted to enter your OpenID Connect credentials.
   b) Click the **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 Web Client application. The application can then get OpenID Connect user information through environment variables such as OIDC_SUB.

   > **Note:** The `fglrun` process is executed in the context of the GAS operating system user. For example, when using Apache, the program process will run in the context of the Apache user.

The next time you start the same application - or any application delivered by the same Genero Application Server - you will not be prompted for your credentials. The application will start and be authenticated by the same OpenID Connect user.

> **Tip:** Read all of the OpenID Connect topics in the Genero Application Server User Guide for details on features provided by OpenID Connect SSO support in the Genero Application Server; including attributes gathering or authorization control.

Configure GAS for OpenID Connect SSO

Follow these steps to configure the Genero Application Server for OpenID Connect Single sign-on (SSO).

1. Create an account with an OpenID Connect provider that will provide authentication services for you, e.g. see Configure OpenID Connect identity on Google on page 116.

   Through this one account, the IdP provides you with an authentication services that identifies to the GAS the users that access your application. Add the public and shared secret ids obtained from the IdP to your application's configuration files, see Add OpenID Connect SSO to Genero Web application on page 117.
2. If the Genero Application Server is located behind a proxy, configure the proxy in the OpenID Connect `fglprofile` file in `$FGLDIR/web_utilities/services/openid-connect/res`.

   Remove the comment and set the correct value for the entry called `proxy.http.location` and `proxy.https.location`.
3. Start your dispatcher (if not behind a web server).

   > **Note:** Genero OpenId Connect service requires HTTPS communication with the IdP. If needed, you may have to configure SSL and CA authority in the fglprofile file. (see the *Genero Business Development Language User Guide* for details).

The Genero Application Server is ready to use OpenID Connect SSO to authenticate end users.

Configure OpenID Connect identity on Google

Follow these steps to configure an OpenID Connect Single sign-on (SSO) identity on Google.

1. Go to the Google developer console page https://console.developers.google.com/)
2. Create a new project (or use an existing one)
3. From the project page select **Credentials** under **APIs and auth**
4. In the **Credentials** page, select **Create new Client ID**

   This will open a new window where you can create a client identity and select a product.

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

The OAuth2 **PUBLIC** and **SHARED SECRET** IDs are displayed

Note: You will need to save these in your web service application configuration file.

You have now setup Google as your IdP for your web services to use OpenID Connect SSO.

Add OpenID Connect SSO to Genero Web application
Follow these steps to add OpenID Connect SSO to a Genero Web application.

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

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

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

- IDP : the IdP account (e.g. https://accounts.google.com)
- CLIENT_PUBLIC_ID : the OAuth2 public id from the IdP
- CLIENT_SECRET_ID : the OAuth2 shared secret id from the IdP
- SCOPE : (optional) the OpenID Connect attributes you want to get at authentication (e.g. email, phone, address)

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
      <DELEGATE service="services/OpenIDConnectServiceProvider" >
          <IDP>https://accounts.google.com</IDP>
          <SCOPE>email</SCOPE>
          <CLIENT_PUBLIC_ID>XXXXXXXX.apps.googleusercontent.com</
CLIENT_PUBLIC_ID>
          <CLIENT_SECRET_ID>XXXXXX-XXXXXX</CLIENT_SECRET_ID>
      <DELEGATE>
  </EXECUTION>
</APPLICATION>
```

With the above configuration and default GAS configuration, the delegation will point to the `$FGLDIR/web_utilities/services/OpenIDConnectServiceProvider.xcf` file.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 105.

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

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

To retrieve the OpenID Connect user identifier, add this code to your Genero application:

Once the user has been successfully authenticate and before starting the proxy, the OpenID Connect service sets all attributes coming from the IdP with the prefix *OIDC_* . (OIDC stands for OpenID Connect).

> **Note:** Even if there are no attributes being sent by the IdP (maybe because the user has not allowed the Google console API to send them), the *OIDC_SUB* attribute will always be available. This attribute is an opaque value representing the user subject at IdP.

For example, if you set email in the SCOPE parameter of your application configuration (see Quick start: Set up OpenID Connect in the GAS on page 115), you will have an attribute called *OIDC_email* set that is then retrievable with the instruction : `fgl_getenv("OIDC_email")` in your application.

```
LET userEmail = fgl_getenv("OIDC_email")
```

Authorization and OpenID Connect SSO

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

The GAS must be configured for OpenID Connect Single sign-on (SSO). See Configure GAS for OpenID Connect SSO on page 116.

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.

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

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

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

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

Example with a Genero authorization program:

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

The application `AccessProgram.4gl` in `$FGLDIR/web_utilities/services/openid-connect` provides an example of an authorization application written in Genero.

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

If the `AUTHORIZATION` element is not defined, any user authenticated by an OpenID Connect provider can access the Genero Web application. It is therefore recommended that you add an authorization program to filter access to your applications.

> **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`.

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"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDConnectServiceProvider">
      <AUTHORIZATION>fglrun AccessProgram</AUTHORIZATION>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

The authorization program will be called before access to the Web application is granted. If the authorization program exits with an error code of zero (0), then access is granted for the user. Any exit code other than zero indicates access for the user is denied. In the last case, the end user will be warned with a error page in the web browser, generated by the OpenID Connect service.

Genero OpenID Connect FGLPROFILE
Genero OpenID Connect implementation uses its own FGLPROFILE file.

The Genero OpenID Connect Single sign-on (SSO) implementation uses its own FGLPROFILE file in `$FGLDIR/web_utilities/services/openid-connect/res`.

This file can be modified to define the following features:

- ODI database driver definition.
- HTTP and HTTPS proxy configuration.
- X509 and SSL keys for handling HTTPS connection (if needed).

When to modify this file:

- If you want a database engine other than SQLite.
- If your Genero Application Server installation requires proxy configuration to connect to an OpenID Connect provider.

Genero OpenID Connect log file
The Genero OpenID Connect implementation produces a log file that helps to identify issues.

The log file of the Genero OpenID Connect Single sign-on (SSO) implementation is called `OIDC.log` and is located in `$(res.appdata.path)/log`. This log file contains all incoming and outgoing requests. It can help to debug OpenID Connect issues.

You can specify the level of detail recorded to the log with the `-debug` *`category`* option of the OpenID Connect server program. There are two categories that can be logged individually or together:

- `MSG` - Standard information regarding access and errors. By default, only access and error information are logged.
- `DEBUG` - Traces the entire process of single sign-on (SSO).

To add debugging information to the OIDC.log, modify `OpenIDConnectServiceProvider.xcf` to include the `-debug DEBUG` option as first argument in the command defined by the `MODULE` element:

```
<APPLICATION Parent="ws.default"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="FGLPROFILE">
```

```
      $(res.path.fgldir.services)/openid/res/fglprofile</
ENVIRONMENT_VARIABLE>
    <PATH>$(res.path.fgldir.services)/openid-connect/bin</PATH>
    <MODULE>OpenIDConnectServer -logfile "$(res.appdata.path)" -debug
 DEBUG</MODULE>
    <POOL>
      <START>1</START>
      <MIN_AVAILABLE>0</MIN_AVAILABLE>
      <MAX_AVAILABLE>10</MAX_AVAILABLE>
    </POOL>
  </EXECUTION>
</APPLICATION>
```

Note: Logging is based on the Genero `ERRORLOG()` function. As several instances of the same OpenID Connect server can write into the log file, the PID of the server is written to the log file as well.

OpenID SSO

OpenID is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under `$FGLDIR/web_utilities/services/openid`.

OpenID providers include Google and Yahoo. To learn more about OpenID, see the OpenID web site.

Note: Genero implements only version 2.0 of the OpenID specification and is only intended for Genero Web Client applications.

The Genero OpenID solution is supported on the GAS delegation mechanism. See What is delegation? on page 37.

Important: Genero OpenID service uses an SQLite database to store authentication data. If you do not configure another database, Genero OpenID service will use the `openid.db` SQLite database located in `$FGLDIR/web_utilities/services/openid/bin`. The OpenID REST service will execute in the context of the Genero Application Server. This user must have write permission on the `openid.db` file and the parent `bin` directory, otherwise the service will fail to insert data into the SQLite database.

Quick start: Set up OpenID in the GAS

Follow these steps to quickly set up OpenID Single sign-on (SSO) for your Genero Application Server and Genero Web Client applications.

Before you begin, you must Configure GAS for OpenID SSO on page 121.

In this quick start, you add OpenID Single sign-on (SSO) to a Genero Web Client application, then execute the application with SSO.

1. Add OpenID SSO to a Genero Web Client application.

a) Add the `DELEGATE` element to all Genero Web Client applications requiring SSO.

```
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.mypath)/myapplication</PATH>
    <MODULE>myapp.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider" />
  </EXECUTION>
</APPLICATION>
```

b) Add a `PROVIDER` element to indicate which identity provider to use for the application.

```
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.mypath)/myapplication</PATH>
    <MODULE>myapp.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider">
      <PROVIDER>google.com</PROVIDER>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

If a provider is not defined, a page with the list of available ID provides is displayed.

2. Execute a Genero Web Client application with SSO.

a) Start your browser and enter the application URL.
You are prompted to enter your openid.

b) Click the **signin** button.
Your browser is redirected to the OpenID provider.

c) Enter your credentials.

If your credentials are valid, your browser is redirected to the Genero Web Client application. The application can then get OpenID user information through environment variables such as OPENID_CLAIMED_ID.

> **Note:** The fglrun process is executed in the context of the GAS operating system user. For example, when using apache, the program process with run in the context of the apache user.

The next time you start the same application - or any application delivered by the same Genero Application Server - you will not be prompted for your credentials. The application will start and get the same OpenID user information.

> **Tip:** Read all of the OpenID topics in the Genero Application Server User Guide for details on features provided by OpenID SSO support in the Genero Application Server, to include attributes gathering or authorization control.

Configure GAS for OpenID SSO

Follow these steps to configure the Genero Application Server for OpenID Single sign-on (SSO).

1. Create one or more OpenID users on an OpenID provider.

2. If the Genero Application Server is located behind a proxy, configure the proxy in the OpenID `fglprofile` in `$FGLDIR/web_utilities/services/openid/res`.

Remove the comment and set the correct value for the entry called `proxy.http.location` and `proxy.https.location`.

3. Start your dispatcher (if not behind a web server).

The Genero Application Server is ready to use OpenID SSO to authenticate end users.

Add OpenID SSO to a Genero Web application

Follow these steps to add OpenID Single sign-on (SSO) to a Genero Web application.

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

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

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider"/>
  </EXECUTION>
</APPLICATION>
```

With the above configuration and default GAS configuration, the delegation will point to the `$FGLDIR/web_utilities/services/OpenIDServiceProvider.xcf` file.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 105.

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

Retrieve the OpenID user identifier

Follow these steps to retrieve the OpenID user identifier in your Genero application.

To retrieve your OpenID Single sign-on (SSO) user identifier, add this code to your Genero application:

```
LET id = fgl_getenv("OPENID_CLAIMED_ID")
```

More OpenID user information can be retrieved, for details see Retrieve identity attributes with OpenID SSO on page 122.

Retrieve identity attributes with OpenID SSO

Follow these steps to retrieve additional attributes about user identity when authenticating to an OpenID provider.

As a prerequisite, identify which identity attributes are supported by the OpenID provider (i.e. the identity provider - IDP). The Genero OpenID Single sign-on (SSO) implementation automatically detects which attribute exchange protocol is supported.

There are two kinds of attribute exchange protocols:

- OpenID Simple Registration Extension (default for Genero)
  - Specifications
  - List of attributes
- OpenID Attribute Exchange (used if default is not available)
  - Specifications
  - List of attributes are specific to an OpenID provider and must be mapped to a single name in the configuration file of the Genero OpenID implementation.

Complete this procedure to retrieve additional attributes about your identity when authenticating to an OpenID provider. For example, you can retrieve the email, full name, or country of the user.

1. Add an `ATTRIBUTES` element as a child of the OpenID `DELEGATE` element in the application configuration (`xcf`) file. Provide a comma-separated list of OpenID attributes within the `ATTRIBUTES` element.

   In this example, the `email`, `fullname`, and `country` openid attributes are specified.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider">
      <ATTRIBUTES>email,fullname,country</ATTRIBUTES>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

2. To retrieve the OpenID attributes in your Genero application, add a `fgl_getenv()` call for each attribute specified in the XCF file with a prefix of `OPENID_`.

```
LET email = fgl_getenv("OPENID_email")
LET fullname = fgl_getenv("OPENID_fullname")
LET country = fgl_getenv("OPENID_country")
```

The Genero application retrieves the requested identity attributes.

Authorization and OpenID SSO

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

The GAS must be configured for OpenID Single sign-on (SSO). See Configure GAS for OpenID SSO on page 121.

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

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

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

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

- The first argument is the OpenID identifier.
- The second argument is the application path.
- Next arguments are optional and define OpenID attributes/value pairs.

Example with a Genero authorization program:

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

The application `AccessProgram.4gl` in `$FGLDIR/web_utilities/services/openid` provides an example of an authorization application written in Genero.

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

If the AUTHORIZATION element is not defined, any user authenticated by an OpenID provider can access the Genero Web application. It is recommended that you add an authorization program to filter the access to your applications.

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

1. Add an AUTHORIZATION element as a child of the OpenID DELEGATE element in the application configuration (xcf) file.
2. Within the AUTHORIZATION element, specify the command to execute the external authorization program.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/OpenIDServiceProvider">
      <ATTRIBUTES>email,fullname,country</ATTRIBUTES>
      <AUTHORIZATION>fglrun AccessProgram</AUTHORIZATION>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

The authorization program will be called before access to the Web application is granted. If the authorization program exits with an error code of zero (0), then access is granted for the user. Any exit code other than zero indicates access for the user is denied. In the last case, the end user will be warned with a error page in the web browser, generated by the OpenID service.

Execute an application with OpenID SSO

Follow these steps to execute a Genero Web Client application and authenticate user with OpenID Single sign-on (SSO).

1. Open a browser and enter the application URL.
   You will be prompted to enter your openid in the form **xxx.openid.com**.
2. Click the **signin** button.
   Your browser redirects to the OpenID provider.
3. Enter your credentials.

If the credentials are valid, your browser redirects to your Genero Web Client application that starts and runs as the entered OpenID user.

When you next start the same application, or any application served from the same Genero Application Server, you will not be prompted for your credentials. The Genero Web Client application will start and authenticate with the same OpenID user.

Distinct user authentication per application

Use the realm parameter to define specific domains for a set of applications.

By default, the OpenID Connect Single sign-on (SSO) service is launched with a realm parameter set to auto. All applications using this OpenID service are authenticated by the OpenID provider as coming from the same domain, and thus only require one authentication process for all of them.

The `realm` parameter can be changed from `auto` to an exact domain name (such as `www.4js.com:6394/gas`):

```
<EXECUTION>
    ...
    <PATH>$(res.path.fgldir.services)/openid/bin</PATH>
    <MODULE>OpenIDServer -realm www.mycompany.com:6394/gas -logPath
 "$(res.appdata.path)"</MODULE>
    ...
</EXECUTION>
```

When using an exact domain name, ensure that all URLs for accessing your Genero Web applications have that form, otherwise you will get an OpenID error message.

To force the OpenID authentication login for any application executed by the GAS, remove the `realm` parameter from the `$FGLDIR/web_utilities/services/OpenIDServiceProvider.xcf` file. The end user will then be requested for credentials for any single application, even if they use the same OpenID service.

Genero OpenID configuration file
Specify OpenID provider constraints and mappings in the configuration file.

The Genero OpenID Single sign-on (SSO) implementation provides a configuration file named `configuration.xml` in `$FGLDIR/web_utilities/services/openid/res`.

The OpenID configuration file allows you to:

- Specify OpenID provider constraints about unsupported features (encryption and signature methods)
- Map the OpenID attributes URL to a single name if OpenID Attribute Exchange protocol is required.

Server constraints

Some OpenID providers do not support all encryption and signature methods recommended in the specification. To bypass this issue, you can specify the supported method for each individual OpenID provider. By default, Genero OpenID implementation supports the strongest one.

The *encryption method* defines how the Genero OpenID implementation and the OpenID provider will exchange the signature key. Supported methods are:

no-encryption	Requires HTTPS as communication channel between the OpenID provider and the Genero implementation. SSL certificate and keys must be specified in the fglprofile of the OpenID service (`$FGLDIR/web_utilities/services/openid/res/fglprofile`).
DH-SHA	Default. No specific communication channel required. The signature key is exchanged using a public/private Diffie-Hellman key agreement method fully secured.

The *signature method* defines how the Genero OpenID implementation validates what comes from the OpenID provider. Supported methods are:

direct	Requires HTTPS as communication channel between the OpenID provider and the Genero implementation in order to validate an authentication. Each authentication process requires an additional connection to the OpenID provider.

HMAC-SHA1	Default. No specific communication channel required. The Genero OpenID implementation can validate the authentication without any additional request to the OpenID provider.

Each server has a *secured attribute* that ensures that if a combination of encryption and signature method is not fully secured, the authentication process fails with a specific message in the log file.

If you set this attribute to false, the authentication process would not be fully secured as keys are exchanged between the OpenID provider and the Genero implementation on unsecured channels and thus vulnerable to malicious attacks.

OpenID Attribute Exchange mapping

If an OpenID provider uses the OpenID Attribute Exchange protocol, the configuration file must define an URL for each kind of attributes the OpenID provider supports.

This example illustrates how this is completed for the Google OpenID provider.

```
<Server secured="true" provider="google.com">
  <URL>www.google.com/accounts/o8/ud</URL>
  <Encryption>no-encryption</Encryption>
  <Signature>HMAC-SHA1</Signature>
  <AttributeProfile>http://openid.net/srv/ax/1.0</AttributeProfile>
  <Attribute name="email">http://axschema.org/contact/email</Attribute>
  <Attribute name="country">http://axschema.org/contact/country/home</
Attribute>
  <Attribute name="firstname">http://axschema.org/namePerson/first</
Attribute>
  <Attribute name="lastname">http://axschema.org/namePerson/last</Attribute>
  <Attribute name="language">http://axschema.org/pref/language</Attribute>
</Server>
```

The identity provider given in the application configuration file must match an identify provider configured in `$FGLDIR/services/openid/res/configuration.xml`.

The response from the ID provider should contain the provider identity ("google.com" in the above example).

The `AttributeProfile` element indicates the method to retrieve the attributes.

Specify a database to store OpenID data

Follow these steps to specify a database different from the default for the Genero OpenID implementation.

The implementation of Genero OpenID Single sign-on (SSO) requires a database to store OpenID data related to the protocol. By default, the database engine is SQLite and the database file is *`$FGLDIR/`*`web_utilities/services/openid/bin/openid.db`. This database is fully functional after installing the Genero Application Server.

1. Create a new or use a existing database, eventually on a dedicated machine, if several GAS servers are configured for load balancing. There must be a unique database, to centralize all OpenID authentication data.
2. In the file `server.4gl`, modify the functions `BDConnect()` and `DBDisconnect()` to handle and customize the database connection. Recompile the `server.4gl` source.

   `server.4gl` is found in *`$FGLDIR`*`/web_utilities/services/openid/src`
3. Modify the FGLPROFILE file in `$FGLDIR/web_utilities/services/openid/res` to include the connection information for the database.
4. Create OpenID tables with the `CreateDatabase.4gl` program. Define the database permissions required to let the Genero Application Server modification the OpenID tables in the new database.

   `CreateDatabase.4gl` is found in *`$FGLDIR`*`/web_utilities/services/openid/src`

5. If needed, define the PATH (Windows) or LD_LIBRARY_PATH (UNIX) environment variables in `$FGLDIR/web_utilities/services/OpenIDServiceProvider.xcf` with `ENVIRONMENT_VARIABLE` elements, in order to find the database client libraries required by Genero OpenID service.

   Note: If you use SQLite (by default), you do not need to add the path to the library since it is integrated in the ODI driver on most systems.

The alternate database is now used for the Genero OpenID implementation.

Genero OpenID FGLPROFILE
Genero OpenID implementation uses its own FGLPROFILE file.

The Genero OpenID Single sign-on (SSO) implementation uses its own FGLPROFILE file in `$FGLDIR/web_utilities/services/openid/res/fglprofile`.

This file can be modified to define the following features:

- ODI database driver definition.
- HTTP and HTTPS proxy configuration.
- X509 and SSL keys for handling a HTTPS connection.

When to modify this file:

- If you want a database engine other than SQLite.
- If your Genero Application Server installation requires proxy configuration to connect to an OpenID provider.
- If you need HTTPS specific settings to communicate with your OpenID provider.

Genero OpenID log file
The Genero OpenID implementation produces a log file that helps to identify issues.

The log file of the Genero OpenID implementation is called `OpenID.log` and is located in `$FGLDIR/web_utilities/services/openid/bin`. This log file contains all incoming and outgoing requests. It can help to debug OpenID Single sign-on (SSO) issues.

You can specify the level of detail recorded to the log with the `-debug` *`category`* option of the OpenID server program. There are two categories that can be logged individually or together:

- `MSG` - Standard information regarding access and errors. By default, only access and error information are logged.
- `DEBUG` - Traces the entire SSO process.

To add debugging information to the log, modify `OpenIDServiceProvider.xcf` to include the `-debug DEBUG` option as first argument in the command defined by the `MODULE` element:

```
<APPLICATION Parent="ws.default"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE Id="FGLPROFILE">
      $(res.path.fgldir.services)/openid/res/fglprofile</
ENVIRONMENT_VARIABLE>
    <PATH>$(res.path.fgldir.services)/openid/bin</PATH>
    <MODULE>OpenIDServer -debug DEBUG</MODULE>
    <POOL>
      <START>1</START>
      <MIN_AVAILABLE>0</MIN_AVAILABLE>
      <MAX_AVAILABLE>10</MAX_AVAILABLE>
    </POOL>
  </EXECUTION>
</APPLICATION>
```

Note: Logging is based on the Genero `ERRORLOG( )` function. As several instances of the same OpenID server can write into the log file, the PID of the server is written to the log file as well.

SAML SSO

Security Assertion Markup Language (SAML) is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under `$FGLDIR/web_utilities/services/saml`.

Genero SAML will establish a circle of trust between the service provider (the Genero Application Server) and the SAML identity provider (the entity in charge of managing and authenticating the users).

If you don't configure another database, Genero SAML service will by default use the saml.db SQLite database located in `$FGLDIR/web_utilities/services/saml/bin`. If Genero is installed with a different user than the user who runs the web server, you must (as a minimum) set write permissions for that user on the `openid.db` file and its parent `bin` directory, otherwise the service will fail to insert data into the SQLite database.

Note: Genero implements only version 2.0 of the SAML specification and supports only the HTTP-POST bindings. It is only intended for Genero Web Client applications.

Quick start: Set up SAML in the GAS

Follow these steps to quickly set up SAML for your Genero Application Server and Genero Web Client applications.

In this quick start, you add SAML Single sign-on (SSO) to a Genero Web Client application, then execute the application with SSO.

1. Configure the GAS for SAML SSO:
   a) If your GAS is located behind a proxy, configure the proxy in the SAML `fglprofile`, located in *`$FGLDIR`*`/web_utilities/services/saml/res`. Uncomment and set values for the entries `proxy.http.location` and `proxy.https.location`.
   b) SAML requires digital signatures. Create a X509 Certificate and its private key (see the *Genero Business Development Language User Guide* for details), then modify the SAML configuration file located in *`$FGLDIR`*`/web_utilities/services/saml/res`:
      - Uncomment and set values for the entries `xml.saml_signature.x509` and `xml.saml_signature.key`.
      - If your Genero Web Client applications must be accessible by HTTP, to be fully secured you must use that key and certificate for XML-Encryption. Uncomment and set the same value for the entries `xml.saml_encryption.x509` and `xml.saml_encryption.key`.
   c) Create a circle of trust between the Genero Application Server and a SAML provider.

- Go to `$FGLDIR/web_utilities/services/saml`.
- Set the SAML environment using the scripts `envsaml.bat` or `envsaml.sh`.
- Launch the ImportIdP application with the SAML Provider URL.
  - Example: `fglrun ImportIdp http://host:port/openam_954/saml2/jsp/exportmetadata.jsp`
  - See SAML provider documentation about how to retrieve the Metadata.
- If needed, retrieve the SAML provider Certificate and add it as trusted certificate in the SAML configuration file.
  - Uncomment and set values for the entry `xml.keystore.calist`; see the *Genero Business Development Language User Guide* for more details.
  - See SAML provider documentation about how to retrieve its X509 certificate.

d) Create a circle of trust between the SAML provider and the Genero Application Server.

- Start the dispatcher (if needed).
- Log into your SAML provider and create a circle of trust based on the Genero Application Server SAML metadata available at this URL: `http[s]://host:port/[gas/]ws/r/services/SAMLServiceProvider/Metadata`
  - See SAML provider documentation for information on creating the circle of trust.
  - Genero Application Server default SAML identity name is "urn:genero". If needed, you can change the identifier by modifying the `saml.entityID` entry in the fglprofile file.

2. Add SAML SSO to a Genero Web Client application:

a) Add the `DELEGATE` tag to all Genero Web Client applications requiring SSO.

```
<DELEGATE service="services/SAMLServiceProvider" />
```

For example:

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

3. Execute a Genero Web Client application with SSO:

a) Start your browser and enter the application URL.
You are redirected to the SAML provider and prompted to enter your credentials.

b) Enter your credentials and click the **signin** button.

If your credentials are valid, your browser is redirected to the Genero Web Client application. The application starts and runs as the entered SAML user.

The next time you start the same application - or any application delivered by the same Genero Application Server - you will not be prompted for your credentials. The application will start (and be authenticated for) the same SAML user.

Configure GAS to support SAML SSO

Follow these steps to setup Genero SAML service.

Before you can use SAML Single sign-on (SSO) with the Genero Application Server, a circle of trust must be established between the service providers (the Genero Application Servers) and one or more SAML identity providers (an entity in charge of managing and authenticating the users). This is established via SAML metadata exchange, where each party imports the metadata from the other party. Each party's metadata defines how to communicate with it.

Note: An X509 certificate authority file can also be exchanged in order to validate SAML signatures. See Certificate authority on page 137.

1. If the Genero Application Server is located behind a proxy, configure the proxy in the SAML FGLPROFILE.

   Uncomment and set correct values for the entries `proxy.http.location` and `proxy.https.location`.
2. Create an X509 Certificate and its private key.

   SAML requires digital signatures. See the *Genero Business Development Language User Guide* for information on creating the certificate and its private key.
3. Modify the SAML configuration file and enter the X509 certificate and private key information.

   The SAML configuration file is located in `$FGLDIR/web_utilities/services/saml/res`.

   Remove the comment and set correct values for the entries `xml.saml_signature.x509` and `xml.saml_signature.key`.

   If your Genero Web application must be accessible in HTTP, you must also use that key and certificate for XML-Encryption to be fully secure. Uncomment and set the same values for the entries `xml.saml_encryption.x509` and `xml.saml_encryption.key`.
4. Create a circle of trust between the Genero Application Server and a SAML provider. Import the IdP metadata file into the Genero Application Server SAML service provider.
   a) Go to `$FGLDIR/web_utilities/services/saml`.
   b) Set SAML environment via `envsaml.bat` or `envsaml.sh`.
   c) Launch the `ImportIdP` application using the SAML Provider URL.

      Refer to the IdP documentation for information on generating the metadata file (or the url) from the SAML identity provider.

      ```
      $fglrun ImportIdP http://host:port/openam_954/saml2/jsp/exportmetadata.jsp
      ```
   d) Retrieve the SAML provider Certificate and add it as a trusted certificate in the SAML configuration file (if needed).

      Uncomment and set the correct values for the entry `xml.keystore.calist`. Refer to the *Genero Business Development Language User Guide* for more information.

      Refer to the SAML Identity Provider (IdP) documentation for information about retrieving its X509 certificate.
5. Create a circle of trust between the SAML provider and the Genero Application Server.
   a) Start the dispatcher (if needed).
   b) Log in to your SAML provider and create a circle of trust based on the Genero Application Server SAML metadata.

      Generate the metadata from this URL: `http[s]://host:port/[gas/]ws/r/services/SAMLServiceProvider/Metadata`

      Refer to the SAML Identity Provider (IdP) documentation for information about importing the Genero Application Server SAML metadata.

The Genero Application Server is ready to support SAML SSO.

The ImportIdP program

Use the ImportIdP program to register a SAML identity provider.

With the Genero ImportIdP program, you can:

- Register a new SAML identity provider (IdP) into the Genero Application Server for SAML Single sign-on (SSO).
- Lists all registered IdPs
- Remove the Idp identified by its URI.

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

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

Syntax

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

Note:

1. *options* are described in Table 10: ImportIdP options on page 131.
2. *url* is the url of a SAML identity provider.
3. *file* is the metadata file of a SAML identity provider.

Table 10: ImportIdP options

Option	Description
`-import`	Import the IdP specified by the URL or metadata file.
`-list`	List all registered IdPs.
`-remove`	Remove the registered IdPs.

Add SAML SSO to a Genero Web application

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

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

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

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider"/>
  </EXECUTION>
</APPLICATION>
```

With the above configuration and default GAS configuration, the delegation will point to the `$FGLDIR/web_utilities/services/SAMLServiceProvider.xcf` file.

For more information about the `DELEGATE` configuration element, see How to implement delegation on page 105.

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

Select the SAML server (Identity Provider)

Follow these steps to specify the SAML server a Genero application must use as its Identity Provider (IdP).

Before you begin, determine the EntityID name for the IdP server you wish to specify. Use the ImportIdP program with the `-list` option to identify the EntityID name.

Complete this procedure to specify which SAML server a Genero application must use as its Identity Provider (IdP).

If the IDP element is not set in the DELEGATE element of the application configuration file, the Genero Application Server will retrieve the unique registered IdP. It will raise an error if more than one IdP is registered.

Add an IDP element as a child of the SAML DELEGATE element in the application configuration (xcf) file. Enter the EntityID name in the IDP tag.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider">
      <IDP>http://idp.4js.com</IDP>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

The Genero Application Server will use the specified IdP as its Single sign-on (SSO) identity provider.

Define the SAML ID format

Follow these steps to define the ID format to receive from the SAML IdP.

The SAML Single sign-on (SSO) protocol allows federation of identities. This means that a single user can have different identities on different SAML IdPs. To federate a same user across several IdPs, the notion of ID format was introduced.

The default ID format is transient, meaning that the returned ID is only valid for the current session and has only a meaning for the IdP the GAS is connected to. Other formats exist such as email or persistent, but you must be sure that your IdP supports them, otherwise you will get an error. The IdP decides which format they support. See SAML core specification for more details about the supported ID format.

The ID format allows you to specify how the user is represented to a Service Provider. For Genero Application Server, it defines what piece of data is sent from the IdP to the Genero Application Server to represent the user.

To define the ID format you want to receive from your IdP:

Add an IDFORMAT element with a valid SAML URN as a child of the SAML DELEGATE element in the application configuration (xcf) file.

In this example, the IdP will return the email of the authenticated user to the Genero Application Server as SAML_ID environment variables

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider">
      <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.

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:

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

Set the authentication context
At the Genero Application Server level, you can specify how the Identity Provider must authenticate a user that wants to access a Genero Web application via a browser.

As a prerequisite, see the SAML core specification for the list of supported URNs. There are several methods -- password protected, X509 certificate, PGP -- but not all work for Web-based Single sign-on (SSO).

Note: For most Web Single sign-on, the default authentication method is password protected.

SAML provides a mechanism that allows a service provider (Genero Application Server) to define how a user must be authenticated by the Identity Provider (IdP). The Genero Application Server supports an optional element (AUTHCONTEXT) that allows you to specify which authentication method to use.

If the AUTHCONTEXT element is not defined, the default mechanism set in the IdP is used.

Important: Do not specify this tag unless you require a specific authentication method.

Add an AUTHCONTEXT element as a child of the SAML DELEGATE element in the application configuration (xcf) file. Enter a valid authentication method in the text of the AUTHCONTEXT element.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider">
      <AUTHCONTEXT>urn:oasis:names:tc:SAML:2.0:ac:classes:X509</
AUTHCONTEXT>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

When set, the authentication context method is defined. If the IdP does not support the specified method, or if it uses another mechanism, the Genero Application Server will return an access denied page.

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

As a prerequisite, SAML Single sign-on (SSO) protocol does not provide a mechanism to request specific attributes to be returned when authenticated. You must configure that list at the IdP level. As SAML supports identity federation, it provides a mechanism to map user-specific attributes between different IdPs - an attribute called with one name in one IdP can be called a different name in another IdP. If federation is in use, map them according to other IdPs if needed. Refer to your IdP documentation for more information on how to map and define the list of attributes to pass to the GAS during authentication setup.

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

```
LET email = fgl_getenv(“SAML_email”)
LET fullname = fgl_getenv(“SAML_fullname”)
LET country = fgl_getenv(“SAML_country”)
```

The Genero application retrieves the requested identity attributes.

Authorization and SAML SSO

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

The GAS must be configured for SAML SSO. See Configure GAS to support SAML SSO on page 129.

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

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

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

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

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

Example with a Genero authorization program:

```
fglrun AccessProgram
       "AZEd3R4" \
       "qa-test/application" \
       "fullname" "genero test" \
       "email" "genero@4js.com" \
       "country" "France"
```

The application `AccessProgram.4gl` in `$FGLDIR/web_utilities/services/openid` provides an example of an authorization application written in Genero.

The external program is specified in the application configuration element by adding a `AUTHORIZATION` element in the `DELEGATE` element.

If the `AUTHORIZATION` element is not defined, any user registered in the SAML IdP can access the Genero Web application. It is recommended that you add an authorization program to filter the access to your application.

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

1. Add an `AUTHORIZATION` element as a child of the SAML `DELEGATE` element in the application configuration (`xcf`) file.
2. Within the `AUTHORIZATION` tag, specify the external authorization program.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
```

```
  <EXECUTION>
    <PATH>$(res.path.qa)/applications/myapp</PATH>
    <MODULE>App.42r</MODULE>
    <DELEGATE service="services/SAMLServiceProvider">
      <AUTHORIZATION>fglrun AccessProgram</AUTHORIZATION>
    </DELEGATE>
  </EXECUTION>
</APPLICATION>
```

The authorization program will be called before access to the Web application is granted. If the authorization program exits with an error code of zero (0), then access is granted for the user. Any exit code other than zero indicates access for the user is denied. In the last case, the end user will be warned with a error page in the web browser, generated by the SAML service.

Execute an application with SAML SSO

Follow these steps to execute a Genero Web Client application and authenticate user with SAML Single sign-on (SSO).

1. Open a browser and enter the application URL.
   You are redirected to the SAML provider and prompted to enter your credentials.
2. Enter your credentials.
3. Click the **signin** button.

If the credentials are valid, your browser redirects to your Genero Web Client application, which starts and runs as the entered SAML user.

When you next start the same application, or any application served from the same Genero Application Server, you will not be prompted for your credentials. The Genero Web Client application will start and authenticate with the same SAML user.

Genero SAML configuration

Specify `FGLPROFILE` entries to configure the Genero SAML service provider.

The Genero SAML implementation provides a list of `FGLPROFILE` entries to configure the Genero SAML service provider. The configuration file is located in `$FGLDIR/web_utilities/services/saml/res`.

Table 11: SAML-related FGLPROFILE entries

FGLPROFILE entry	Description
`saml.entityID`	Defines the SAML entity name for the Genero Application Server, which is how the Genero Application Server is represented to other SAML partners. Mandatory. Default is `urn:genero`.
`saml.allowUnsecure`	Defines whether the GAS accepts unsecured authentication mechanisms. Default is `false` (recommended). A SAML authentication mechanism is unsecured if communication between the Identity Provider (IdP) and the Genero Application Server 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 136.
`saml.wantAssertionsSigned`	Defines whether SAML assertions coming from Identity Providers (IdPs) must be signed. Default is

FGLPROFILE entry	Description
	`true` (recommended). It is recommended to have either (or both) saml.wantAssertionsSigned and saml.wantResponseSigned set to true, to ensure the request was not altered. If not signed and entry is set to true, the Genero Application Server returns an access denied HTML page. This entry also adds the wantAssertionsSigned attribute to the SAML metadata describing the SAML needs of the Genero Application Server.
`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 Genero Application Server 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 Genero Application Server sends to the Identity Provider (IdP), you must add an X509 certificate and its RSA private key to handle XML-Signature using the Genero Web Services xml key mapping. There are two entries to be set:

- `xml.saml_signature.x509`: path to the X509 certificate
- `xml.saml_signature.key`: path to the RSA private key

You can use the same X509 certificate and RSA private key for signature, encryption and metadata signature.

Metadata signature

To sign the generated SAML metadata, add an X509 certificate and its RSA private key in charge of XML-Signature using the Genero Web Services xml key mapping. There are two entries to be set:

- `xml.saml_metadata_signature.x509`: path to the X509 certificate
- `xml.saml_metadata_signature.key`: path to the RSA private key

You can use the same X509 certificate and RSA private key for signature, encryption and metadata signature.

Certificate authority

As XML-Signature and XML-Encryption are in use to secure SAML communication, you must specify the list of trusted certificate authorities. This is done via the Genero Web Services key mapping mechanism, where this entry must be added, containing the list of trusted X509 certificates (coming from the Identity Provider (IdP)).

- `xml.keystore.calist`: path of colon-separated certificate authorities the Genero SAML service provider trusts.

Specify a database to store SAML data

Follow these steps to specify a database different from the default database for the Genero SAML implementation.

The implementation of Genero SAML Single sign-on (SSO) requires a database, to store SAML data related to the protocol. By default, the database engine is SQLite and the database file is *`$FGLDIR/`*`web_utilities/services/saml/bin/saml.db`. This database is fully functional after installing the Genero Application Server.

1. Create a new or use a existing database, eventually on a dedicated machine, if several GAS servers are configured for load balancing. There must be a unique database, to centralize all SAML authentication data.
2. In the file `DBase.4gl`, modify the functions `BDConnect()` and `DBDisconnect()` to handle and customize the database connection. Recompile the `DBase.4gl` source.

   `DBase.4gl` is found in *`$FGLDIR`*`/web_utilities/services/saml/src`.
3. Modify `fglprofile` in `$FGLDIR/web_utilities/services/saml/res` to include the connection information for the database.
4. Create SAML tables with the `CreateDatabase.4gl` program. Define the database permissions required to let the Genero Application Server modification the SAML tables in the new database.

   `CreateDatabase.4gl` is found in *`$FGLDIR`*`/web_utilities/services/saml/src`.
5. If needed, define the PATH (Windows) or LD_LIBRARY_PATH (UNIX) environment variables in `$FGLDIR/web_utilities/services/SAMLServiceProvider.xcf` with `ENVIRONMENT_VARIABLE` elements, in order to find the database client libraries required by Genero SAML service. Note that if you use SQLite (by default), you do not need to add the path to the library since it is integrated in the ODI driver on most systems.
   The alternate database is now used for the Genero SAML implementation.

Genero SAML FGLPROFILE

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

The file is located in `$FGLDIR/web_utilities/services/saml/res/fglprofile`.

This file can be modified to define the following features:

- ODI database driver definition.
- HTTP and HTTPS proxy configuration. This is needed only when the ImportFGL tool is used.

When to modify this file:

- If you want a database engine other than SQLite.
- If your Genero Application Server installation requires proxy configuration to connect to an SAML provider.

Genero SAML log file

The Genero SAML Single sign-on (SSO) implementation produces a log file that helps to identify issues.

The log file of the Genero SAML implementation is called `SAML.log` and is located in `$FGLDIR/web_utilities/saml/bin`. This log file contains all incoming and outgoing requests. It can help to debug SAML issues.

You can specify the level of detail recorded to the log with the `-debug` *`category`* option of the SAML server program. There are two categories that can be logged individually or together:

- `MSG` - Standard information regarding access and errors. By default, only access and error information are logged.
- `DEBUG` - Traces the entire process of single sign-on (SSO).

To add debugging information to `SAML.log`, modify `SAMLServiceProvider.xcf` to include the `-debug DEBUG` option in the command defined by the `MODULE` element:

```
<APPLICATION Parent="ws.default"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextws.xsd">
  <RESOURCE Id="res.saml.db" Source="INTERNAL"/>
  <EXECUTION>
   <!-- ENVIRONMENT_VARIABLE entries removed for this example -->

   <PATH>$(res.path.fgldir.services)/saml/bin</PATH>
   <MODULE>SAMLServer -logPath $(res.appdata.path) -debug DEBUG -debug MSG</
MODULE>
   <POOL>
     <START>0</START>
     <MIN_AVAILABLE>0</MIN_AVAILABLE>
     <MAX_AVAILABLE>10</MAX_AVAILABLE>
   </POOL>
  </EXECUTION>
</APPLICATION>
```

> **Note:** Logging is based on the FGL `ERRORLOG()` function. As several instances of the same SAML server can write to a single log file, the PID of the server is written to the log file as well.

How to implement custom single sign-on

The aim of this tutorial is to show the basics for delegating the start of a Genero application to another service, in order to handle the authentication via a REST service.

Tutorial overview

Genero Application Server Single sign-on (SSO) solution is based on the "delegate" mechanism. For more details, see How to implement delegation on page 105.

The purpose of this tutorial is to:

- Prevent direct access to the application and force the end user to enter a login and password via an XHTML page.
- Add a cookie mechanism allowing the password to be kept by the browser, in order to allow future connections without requiring another login.
- Add a simple method for logging out.

This example can be adapted for your own application needs.

> **Important:** This tutorial will help you to understand how to handle authentication using a REST service, using a basic authentication mechanism with a username/password and cookie management. On production sites, the security must be improved. Please consider the recommendations mentioned in the last step of this tutorial.

Workflow

The end user types the URL of the application in his web browser (step 1 in the workflow). Instead of starting the application program directly, the GAS forwards the HTTP request to the SSOService program.

This program checks the user credentials or cookie validity, and returns the appropriate HTTP response. The response is one of the following:

- indicates the user credential are not valid;
- indicates the user credentials are valid, and creates and sets a cookie in the browser; or
- if an existing cookie is received, allows the execution of the application.

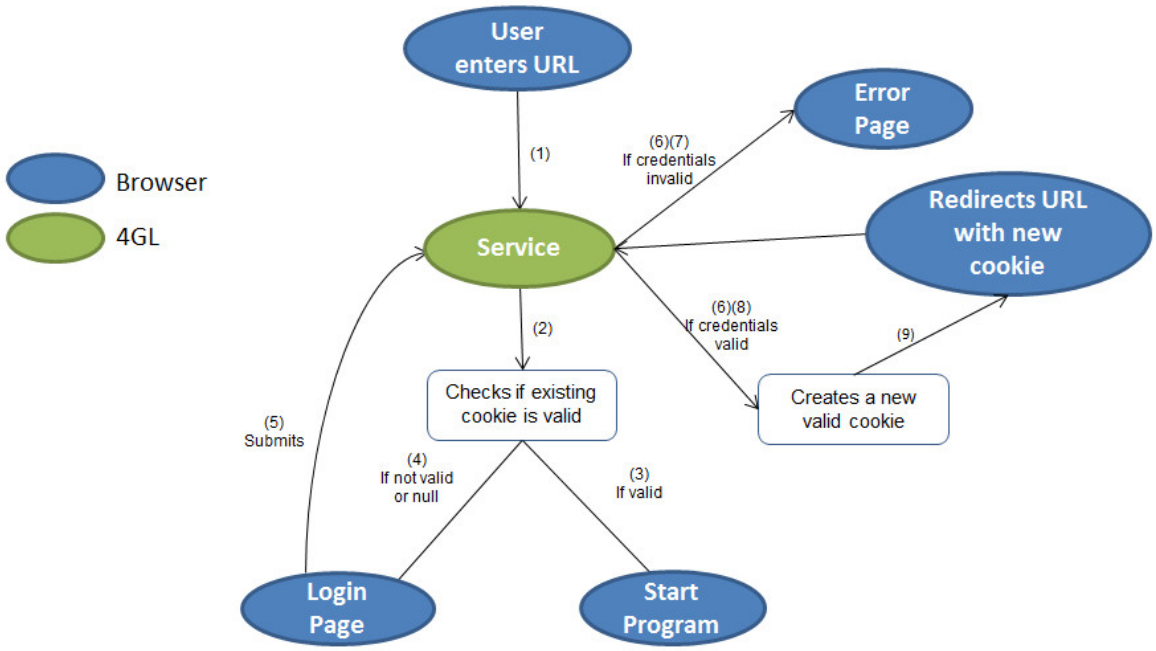


Figure 29: Single sign-on tutorial workflow

As shown in the diagram, the following steps are performed:

1. The user enters a URL in the web browser.

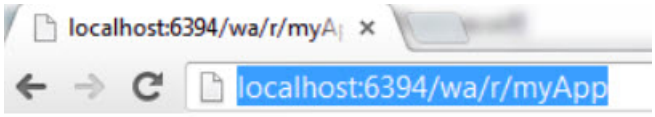


Figure 30: Application URL

2. The Genero REST service receives the HTTP request and checks for a valid cookie.
3. If the cookie is valid, the REST service returns an HTTP response with code 307, and the description `_GENERO_INTERNAL_DELEGATE_`, to notify the GAS dispatcher to start the application.
4. If the cookie is not valid, the REST service returns a login page to the browser, with HTTP code 200.
5. The end user enters a name and password, and submits the HTML form to the GAS.
6. The Genero REST service receives the HTTP request and checks the user credentials.
7. If the user is invalid, the REST service returns an HTTP request with an error page, with HTTP code 200.
8. If the user is authenticated, the REST service creates a new cookie, and returns and HTTP response with code 302, to redirect the browser to the application URL.
9. The browser receives the HTTP response, and redirects to the application URL, with the new cookie returned by the REST service. Then return to step 2 (above).

Initial workflow path

The first time the user enters the application URL, it is redirected to the login page.

Figure 31: Login page

Based on XHTML standards, when the login button is clicked, all data entered in fields of the form, as well the checking of the box “Would you like to keep password?”, are set as parameters in the URL query string.

> **Important:** Use HTTPS in production. This example uses a POST method to submit the login form. With the POST method, the username and password are not visible in the browser URL field, however they are passed in clear text in the body of the POST request. You should use HTTPS in order to encrypt and secure on any production site, and avoid clear data being sent through the network.

In step 2 of the workflow, the SSOService program checks the existence of a valid cookie, and analyzes the URL query string parameters in order to find values for the user login and password. Since this is the first start, there is no existing valid cookie (step 3b of the workflow) and the query string is null because the HTML login form was not yet submitted (step 4 on the workflow). As a result, the connection is refused and the user is directed by the service to a login XHTML page.

GAS configuration for delegation

Delegate the start of an application to a REST service program.

Configure the application for Simple SSO provider

Modify your customization to delegate the execution of the application to a REST web service.

Add a `DELEGATE` element under the `EXECUTION` element in the customization file (`xcf`), with a reference to the `SimpleSSOServiceProvider` service:

```
<EXECUTION>
  <PATH>$(res.path)</PATH>
  <MODULE>myApp.42r</MODULE>
  <DELEGATE service="services/SimpleSSOServiceProvider">
  </DELEGATE>
```

```
</EXECUTION>
```

Configure the SSO service program

The name `SimpleSSOServiceProvider` maps to the `SimpleSSOServiceProvider.xcf` configuration file, located in `$FGLDIR/web_utilities/services`. This configuration file contains these execution settings:

```
<EXECUTION>
  <PATH>$(res.path.fgldir.services)/simplesso/bin</PATH>
  <MODULE>SSOService.42r</MODULE>
</EXECUTION>
```

In this example, the SSO service program is `SSOService.42r`, and the fglrun execution context in `$FGLDIR/web_utilities/services/simplesso/bin`.

Handle login and password input

Prevent direct access to the application and force the end user to enter a login and password.

URL handling in the Simple SSO service program

Once the login page has been submitted (step 4 in the workflow), the incoming URL is split in 2 parts:

- the base URL
- the query string

The query string is parsed by the `SSOService.4gl` code, to retrieve values for the user name, the password, the state of the “keeping password” checkbox, and any potential query string parameters the user may have manually entered in the URL:

```
CALL parseQueryString(query) RETURNING user, pwd, userQueryString, isCookie
```

> **Important:** Consider reviewing the functions included in the sample. They are intended to be simple examples. For a production site, you would want to avoid data transmitted in clear.

Once the user and password have been retrieved from the query string, the service program checks whether the user name and password are valid.

If the user name and password are valid, the application can be started by creating the cookie for the user, as explained in Cookie handling on page 141.

If the user could not be authenticated, the Genero REST service returns an HTTP request with an error page, with HTTP code 200:

```
CALL req.setResponseHeader("Content-Type", "text/html")
CALL req.sendTextResponse(200,"Access Forbidden", html-error-page-string)
```

Cookie handling

The cookie mechanism allows the browser to permanently keep the password, eliminating the need to login when the user revisits the application.

How does the cookie mechanism work?

If the user name and password are valid, the REST service program creates a new cookie to be sent to the browser in the HTTP response. An instantaneous redirection is done, this valid cookie is checked by the service (it returns to step 2 of the workflow) and the connection is accepted (step 3a of the workflow). The next time the user starts the application, the SSO service program will check the validity of this cookie. According to the cookie's expiration, the application may start automatically without requiring the user to login again or returning a new login page.

The cookie is set via the HTTP header “Set-Cookie” and requires a name-value pair that can be sent to the browser by a request on the form.

```
CALL req.setResponseHeader("Set-Cookie", myCookie)
```

Where *myCookie* is a string containing a name-value pair, with optional parameters, for example:

```
LET myCookie = COOKIE_NAME, "=",
               CookieEncrypt(user,pwd,expiredDate,key), "; ",
              "Path=", getCookiePath(path)
```

Note that the following information should also be set:

- A expiration date for the cookie (cookie attributes "max-age" or "expires").
- A path for limiting the validity of the cookie to the current application (cookie attribute “path”).

Cookie expiration date handling

The date of expiration is usually defined by the cookie attribute “max-age”. It represents a value in seconds relative to the current date and time. For instance, `max-age=3600` means that the cookie will expire in 1 hour.

The max-age attribute is not supported by Internet Explorer. The service could use the attribute “expires”, however it requires an absolute GMT hour and Genero currently doesn't handle this format.

To solve this problem, the expiration date is directly included (and encrypted) inside the cookie value.

The cookie structure

To ease the understanding of this example, the cookie has a simple structure:

`'`*`name`*`=`*`value`*`;Path=`*`path_value`*`'`

- The name of the cookie is hard-coded in the `SSOUserFunctions.4gl` code with the `COOKIE_NAME` constant.
- *value* of the cookie is an encrypted combination of username, password and expiration date, separated by a given separator. Once decrypted, value of the cookie can be something like `'userName="myuser";password="mypassword";expirationDate="12/12/2018"'`.

  The separator and the name of attributes can be changed by the constants `C_USERNAME` (for the username attribute), `C_PASSWORD` (for the password attribute), `C_EXPIRATION` (for the expiration date attribute) and `C_SEP` (for the separator) in the`SSOUserFunctions.4gl` module.
- Path remains the normal “Path” cookie attribute. *path_value* should correspond to what is after “http://host:port” in the URL.

How the cookie is handled in the Simple SSO service

The cookie needs to be checked initially by the Simple SSO service (step 2 of the workflow) in order to know if the application can be executed directly or if the end user needs to login again.

There are three options.

1. The cookie is valid (step 3a of the workflow).

   The connection is accepted immediately and the application executed. For checking if the cookie is valid, the service needs to retrieve the content of the “Cookie” HTTP header. Once this content has been retrieved, the service decrypts the value of the expiration date. If the expiration date is later than the current date (`CURRENT` value), the connection is accepted using HTTP code 307 and the description `_GENERO_INTERNAL_DELEGATE_`.
2. The cookie is no longer valid as the cookie expiration date has expired (step 3b of the workflow).

A cookie cannot be valid if the date has expired. The cookie is set to a new value (in our example, the user and password values are set to -1 before being encrypted in this new cookie value) and a redirection is done on the same URL. After redirection, the cookie is decrypted and values “-1” are found for login and password. They are considered as invalid and the user is redirected again to the login page. Specifically, in our sample, it's redirected to an XHTML login page indicating “the session has expired”.

3. The cookie is NULL (step 3b of the workflow).

   For the initial connection, the cookie is NULL. It can be redirected to a simple “Welcome” page rather than the login page.

Expiration date of the cookie

When the cookie is created, it is handled like this:

- If the “keeping password” box has been checked (password kept), the expiration date is set by default to one year later.
- If “keeping password” box has not been checked (password not kept), the expiration date is almost instantaneous (10 seconds by default, so that the cookie remains only valid one time for redirection)

It can be easily changed by the constants `C_EXPIRATION_COOKIE_CHECKED` (value in years) and `C_EXPIRATION_COOKIE_UNCHECKED` (value in seconds) of the `SSOUserFunctions.4gl` module.

Disconnect (log off)
How does the user disconnect after a permanent cookie is set?

If the end user wants to re-connect when a permanent cookie has been set, the `disconnect=true` parameter must be added to the URL:

```
http://host:6394/ua/r/myapp?disconnect=true
```

The user will be automatically redirected to the login page.

When the cookie has expired, the user name and password are set to -1 before being encrypted in the cookie value. In such instances, redirection is also done to the login page.

Production recommendations
It is important to improve security beyond the tutorial.

The tutorial is designed to convey single sign-on basics. Consider these recommendations when preparing for your production system.

- For easing the understanding of this sample, user, password and expiration date have been encrypted directly in the cookie. This should not be done on a production site. If somebody found the decryption algorithm, he would be able to read user and password values in clear. We recommend you review the encryption mechanism and provide better security by encrypting a hash of the login + password, instead of the login + password.
- Function shown in this sample are “fake” functions adapted especially for this example. They may contain some dummy code. Review these functions in detail before adapting them to your production environment.
- A single unique user name and password are hard-coded in the sample source code. A production site requires a complete user management solution.
- Production sites requires the use of the HTTPS protocol rather than HTTP in order to avoid the transmission of clear data through the network.

Connect to the application database with SSO

There are several solutions for automatically connecting to the database server, after starting an application program with a Single sign-on (SSO) delegation.

Overview

The goal of a complete SSO solution is to let the end-user enter credentials once in a login form on the front-end, authenticate that user with an GAS/SSO mechanism, then start the application program and connect to the database without having the end user input other credentials for the database server.

Depending on the features of the target database server, you can implement different techniques to connect to the database automatically, without having to provide more credentials.

The goal of this topic is not to provide a complete example. There are different authentication methods available, and your SSO solution must be adapted to the type of database and operating system. Consider learning about database user security within the database engine of your application.

> **Note:** Because Kerberos SSO support is deprecated by the Genero Application Server, this type of SSO mechanism is not covered in this topic. However, database engines like IBM® Informix® IDS support Kerberos SSO with the Generic Security Services CSM (GSSCM) feature.

User handling depends on the type of Web Application

Regarding application users, we can distinguish the following type of web applications:

- Typical *public web applications*, for an undefined number of end users, who can register themselves to the application.
- Typical *enterprise web applications*, for a defined number of known end users, with strong data access control, managed by application administrators in an enterprise directory.

The SSO solution implemented will depend on the type of web application.

Connecting to the database from the application program

Once the end user is authenticated with one of the SSO mechanisms supported by the GAS, the application is started in the context of the GAS operating system user. For example, when on an Apache server, the application program will execute as the apache user.

Most application programs then connect to a database server, to store and query application data. Connecting to a database server requires the application to identify and authenticate the end user as a database user.

> **Note:** A database user is typically created by a `CREATE USER` SQL statement.

The most common way to authenticate a user in a database connection is to provide the login name and the associated password of the user object existing in the database:

```
MAIN
   DEFINE uname, upswd VARCHAR(50)
   ...
   CONNECT TO "dbsource" USER uname USING upswd
   ...
END MAIN
```

For more details about the `CONNECT TO` instruction, see "Database Connections" chapter in the *Genero Business Development Language Reference Guide*.

> **Note:** With IBM® Informix® IDS servers, database users are traditionally authenticated at the OS level; there must be an OS login from group "informix", created for each DB user. However, IDS 11.7 introduces the concept of *internal users*, to integrate with external authentication mechanisms or to define pure database users based on logins and passwords.

Database user creation

In order to use database engine features to control privileges and audit activity, end users must be identified in the database server, as db user objects in the database system. This is typically done with the `CREATE USER` SQL instruction.

When creating a database user object, you must specify the authentication method.

The basic default authentication method is to specify a password be provided each time the SQL session is created. For example, to create a database user with password authentication in an Oracle® database:

```
CREATE USER username IDENTIFIED BY password
```

It is also possible to define database users with an authentication method based on credentials issued from a trusted part. For example, in Oracle, you can create a database user that will be authenticated with the Oracle Internet Directory®:

```
CREATE USER username IDENTIFIED GLOBALLY AS distinguished name (LDAP DN)
```

Connecting to the DB with predefined database users

This technique can be implemented for a web application where the number of end users is unknown, and where users can register themselves to the application with requiring the database admministrators to create database users. End users enter an application login and password, that will be checked and stored by the application in a dedicated table of the database.

In this solution, use an SSO technology where end users can create credentials from an open identity provider (IdP), as with the GAS OpenID implementation. Since anyone can freely register with the web application, there is no application administrator task regarding the creation of an application user.

To access the application data, a set of predefined database users must be exist in the database, with a fixed name and password, that are hidden to the end users. Each predefined database user will be assigned to several real physical end users. For example, you can create four types of database users, each with specific application permissions and database privileges:

- An *application administrator* can manage application users, can read/write all application data.
- A *read/write access user* can read/write all data of the application.
- A *read access user* can only read all application data.
- A *guest user* can only read a limited set of application data.

Application users are managed and controlled at the application level, and stored in a database table, or in an external resource file (with passwords encrypted).

> **Important:** Because application programs will implicitly connect to the database with predefined database users, no security holes can exist in programs that would allow an end user to connect as a database user to attack the database, for example by using SQL injection.

Once the web application is allowed by the GAS SSO mechanism to start and needs to connect to the database, the program must get application user information (login name, password and type of user), from the GAS SSO procedure. For example, when using OpenID, the web application must get the end user login, password and user type from OpenID attributes through the corresponding environment variables:

```
LET user_name = FGL_GETENV("OPENID_user_name")
LET user_pswd = FGL_GETENV("OPENID_user_pswd")
LET user_type = FGL_GETENV("OPENID_user_type")
```

The user type identifies the predefined database user that will be used to connect to the database, and in turn determines the privileges allowed for the end user.

In order to connect to the database server, the predefined database credentials must be found according to the user type got from the SSO attributes. For example, the program can get the database user name and password from a encrypted configuration file:

```
CALL get_db_login("config_file", user_type) RETURNING db_user, db_pswd
CONNECT TO dbname USER db_user USING db_pswd
```

Once connected to the database, the application program can issue SQL queries as the predefined database user.

> **Note:** Because physical/end users are mapped to predefined/anonymous database users, db server auditing services will not be able to trace end user activity. If needed, this feature must be implemented at the application level.

At this level, the application user must be validated with simple SQL queries. The application user definition can be stored in an application table, where the password should be encrypted:

```
LET user_pswd_encrypted = my_encode(user_pswd)
SELECT last_login INTO ts FROM app_users
   WHERE app_users.u_name = user_name
     AND app_users.u_pswd = user_pswd_encrypted
IF SQLCA.SQLCODE == 100 THEN
   -- application user does not exist: ask for registration, or deny access
   ...
END IF
```

Connecting to the DB with custom SSO implementation

This technique can be implemented for a public web application (where end users can register themselves), or for an enterprise web application (where end users are known and where creation is controlled).

The principle is similar to the Connecting to the DB with predefined database users solution, but instead of using an standard SSO protocol, the GAS SSO delegation feature is used to implement a custom single sign-on procedure.

Application users (SSO login and password) are handled by the delegation program, and associated database user credentials can be stored in a file or light-weight database, which can then be passed through environment variables to the application program. The application program then issues a regular `CONNECT TO` instruction with `USER` *`db_username`* `USING` *`db_password`* option.

Compression in Genero Application Server

Compression is enabled by default for the Genero Application Server. You can disable compression on selected resources or for the entire Genero Application Server.

Files managed by the Genero Application Server can be compressed to reduce the size of files sent to the User Agent. The files can be compressed by hand and deployed in the GAS, but it is also possible to configure the GAS to compress application files on the fly.

Compression is configured in the `$FGLASDIR/etc/imt.cfg` file. This file lists the type of resources that can be compressed. GAS automatically compresses the files with the file extensions listed in `imt.cfg`.

When does compression take place?

Compression takes place at:

- Installation time
- At application deployment with Genero Archive

- At runtime, when communication files, i.e. HTTP request/response type files that do not need to be saved on disk, are exchanged between GAS and applications. These are compressed on the fly.
- At runtime when files on disk are requested by applications.

  Note: At runtime, the GAS checks the `imt.cfg` to see if the requested file is expected to be a compressed file. If the compressed file is not up to date or is missing, the GAS performs the following:

  - It compresses the requested resource on the fly and sends a compressed result.
  - A warning is displayed in the GAS log (see Logging on page 156) if the compressed file is out of date or is missing. (To prevent these warnings, you will need to update compressed files. You can manually generate the compress files (.gz) with the `gasadmin -z` command or using another appropriate compression utility command).
- You can compress static files as required using the `gasadmin -z` command, see gasadmin tool.

Enabling and disabling GAS compression
Both the `gasadmin -z` compression command and the GAS use the `imt.cfg` file to identify the resources that can be compressed.

imt.cfg sample:

```
# [-] Internet media type           Extension
# The optional '-' sign at the beginning can be used to explicitly
# disable compression when sending resources
# Example: see below application/java-archive
#
# Uncomment the following line to completely disable compression
# gas:disable-compression
application/andrew-inset ez
- application/java-archive  jar
application/font-woff woff
application/mac-binhex40 hqx
application/mac-compactpro cpt
application/msword  doc
...
```

To disable compression for a specific resource, place a '-' sign at the beginning of that resource in the listing. For example:

```
- application/java-archive  jar
```

To disable all compression by the Genero Application Server, uncomment the `gas:disable-compression` entry in the file.

Compression enabled (entry is a comment; default):

```
# gas:disable-compression
```

Compression disabled (entry is not a comment):

```
gas:disable-compression
```

Configuring development environment

Configure a development environment to troubleshoot an application.

To troubleshoot and debug an application, you may need to view the application log files and the AUI tree. For this you will need to set up a development environment. For example, to configure a development environment for applications using `uaproxy` protocol, perform the following steps:

1. In the GAS configuration file (default *%FGLASDIR%*/etc/as.xcf), change:

   ```
   <RESOURCE Id="res.uaproxy.param" Source="INTERNAL"></RESOURCE>
   ```

   to:

   ```
   <RESOURCE Id="res.uaproxy.param" Source="INTERNAL">--development</
   RESOURCE>
   ```

   Tip:

   Or alternatively:

   You can also run the dispatcher from the command line and override the settings in the GAS configuration file for `res.uaproxy.param`:

   ```
   httpdispatch.exe -E res.uaproxy.param=--development
   ```

2. Restart the dispatcher.

   The dispatcher must be restarted whenever you modify the application server configuration file in order for the changes to take effect.

3. Enter the application URL in your browser.

   You should now see the **Debug Tools** icon (next to the close application icon) on the right-hand side of your web application window.

4. Click on the **Debug Tools** icon to open the **GWC-JS Debug tools** page in a new browser tab.

   An AUI tree is displayed showing the node and properties of the user interface screen you are currently working on.

Configuring Multiple Dispatchers

If you need to configure multiple dispatchers, you must configure log, temporary and session directories in order to distinguish the entries for each dispatcher.

Multiple dispatchers are typically not needed; in fact they are rarely used. You would need to start multiple dispatchers if you needed to have different environments using the same version of the Genero Application Server on the same host. For example, you may wish to co-locate your production, training, and development environments.

To start multiple dispatchers on a single host, you create a copy of the application server configuration file (default `as.xcf`) for each dispatcher you will start. At a minimum, each dispatcher will have its own uniquely-named configuration file, and the file will have different values for the listening port.

It is important to also ensure that your log, temporary and session directories remain independent. In the default application server configuration file, these directories are located under the directory specified by the resource `res.appdata.path`. These examples are extracted from the default application server configuration file:

```
-- res.appdata.path specifies the directory for application data
-- for the dispatcher

<RESOURCE Id="res.appdata.path"
 Source="INTERNAL">C:\ProgramData\FourJs\gas\2.50.07-130789</RESOURCE>

-- res.log.output.path specifies the log directory,
-- as specified by the LOG element

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

```
<LOG>
  <OUTPUT Type="$(res.log.output.type)">$(res.log.output.path)</OUTPUT>
  <FORMAT Type="TEXT">$(res.log.format)</FORMAT>
  <CATEGORIES_FILTER>$(res.log.categories_filter)</CATEGORIES_FILTER>
  <RAW_DATA MaxLength="$(res.log.raw_data.maxlength)" />
</LOG>

-- res.path.tmp specifies the temporary directory,
-- as specified by the TEMPORARY_DIRECTORY element

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

-- SESSION_DIRECTORY specifies the session directory,
-- as specified by the SESSION_DIRECTORY element

<SESSION_DIRECTORY>$(res.appdata.path)/session</SESSION_DIRECTORY>
```

The quickest (and recommended) way to ensure that a dispatcher's information is not mixing with other dispatcher information is to change the value of the `res.appdata.path` resource to a directory value unique for the dispatcher. This is the recommended approach.

Alternatively, you can change the values for the `OUTPUT` element of the `LOG` element, `TEMPORARY_DIRECTORY` and `SESSION_DIRECTORY` directly.

Administering the Genero Application Server

Understand the options available for the administration of the Genero Application Server.

Monitoring

The GAS Monitor displays information on the GAS dispatcher and on **active** applications. This information is available via an URL, as an XML document.

Statistics about the GAS are provided as an XML document through the `/monitor` URL. You can modify the Statistics presentation by customizing the `monitor.xslt` file located in `$FGLASDIR/web/fjs`.

Usage

Enter a URL to access the monitor.

Standard URL with direct connection:

```
http://appserver:port/monitor
```

For more details on monitor configuration, see Monitor - Configuration Reference in the Configuration Reference section.

If you are using a web server, you will enter in a different URL. Refer to The Application Web Address for more information.

Statistics

The statistics or information provided by the GAS Monitor can be viewed in two steps.

Step 1: Viewing information about the dispatcher

When you first access the GAS Monitoring page, information is provided about the Dispatcher.

Example: http://localhost:6394/monitor provides the dispatcher details.

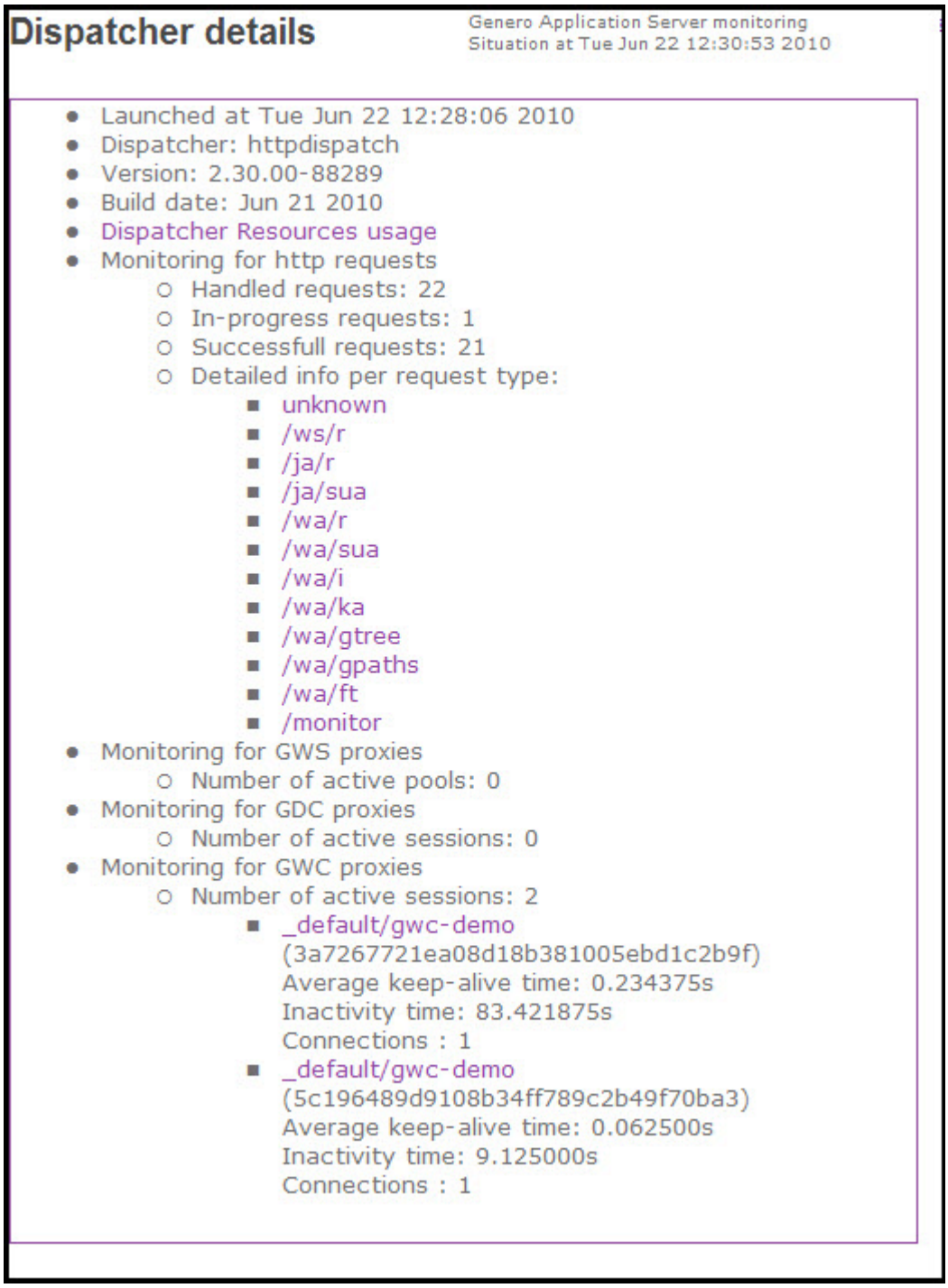


Figure 32: Monitoring: Dispatcher details

Example: http://localhost/gas/monitor provides the dispatcher details.

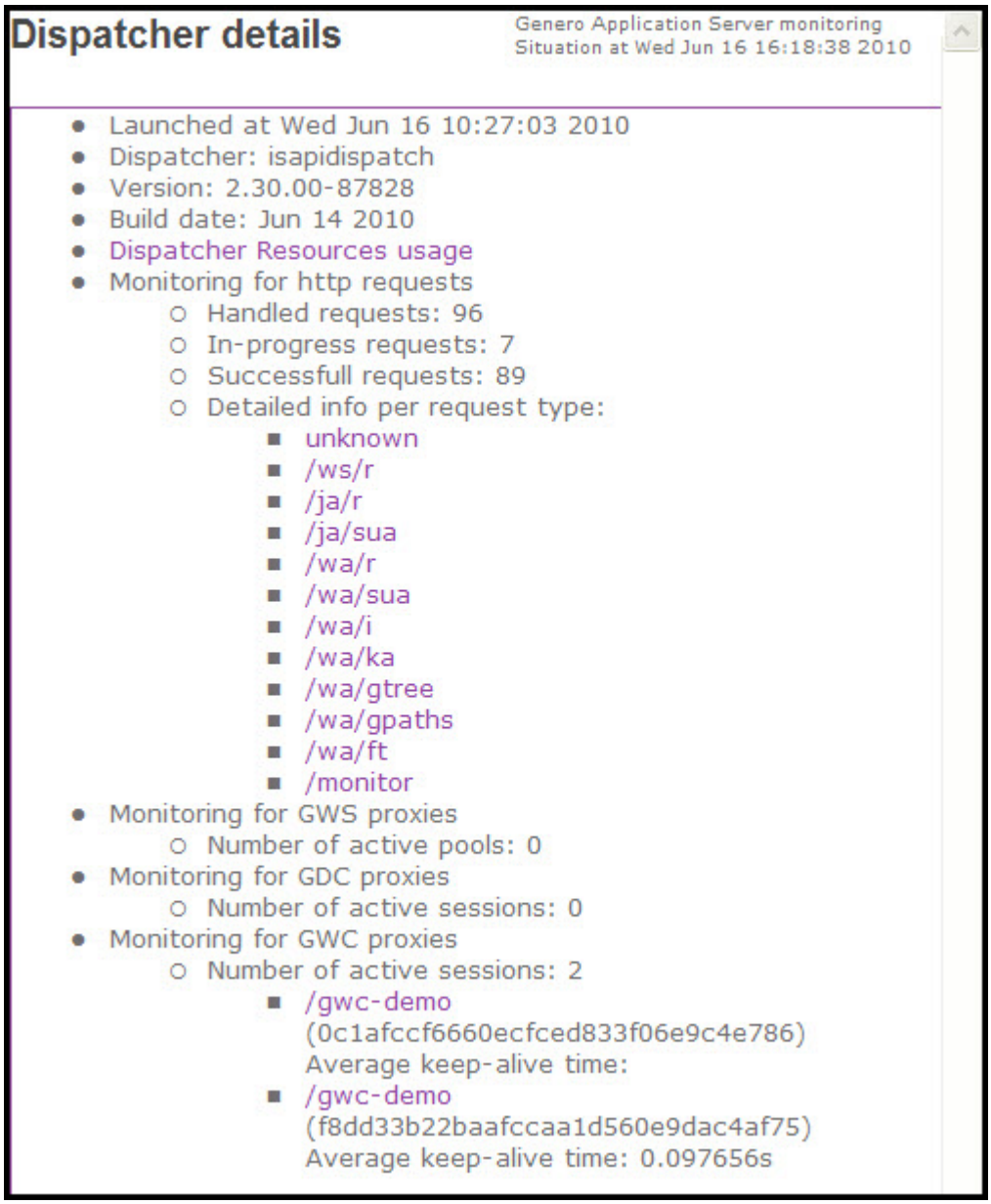


Figure 33: Monitoring: Dispatcher details

From this page, you can view Dispatcher Resources usage information by clicking on the Dispatcher Resources usage link.

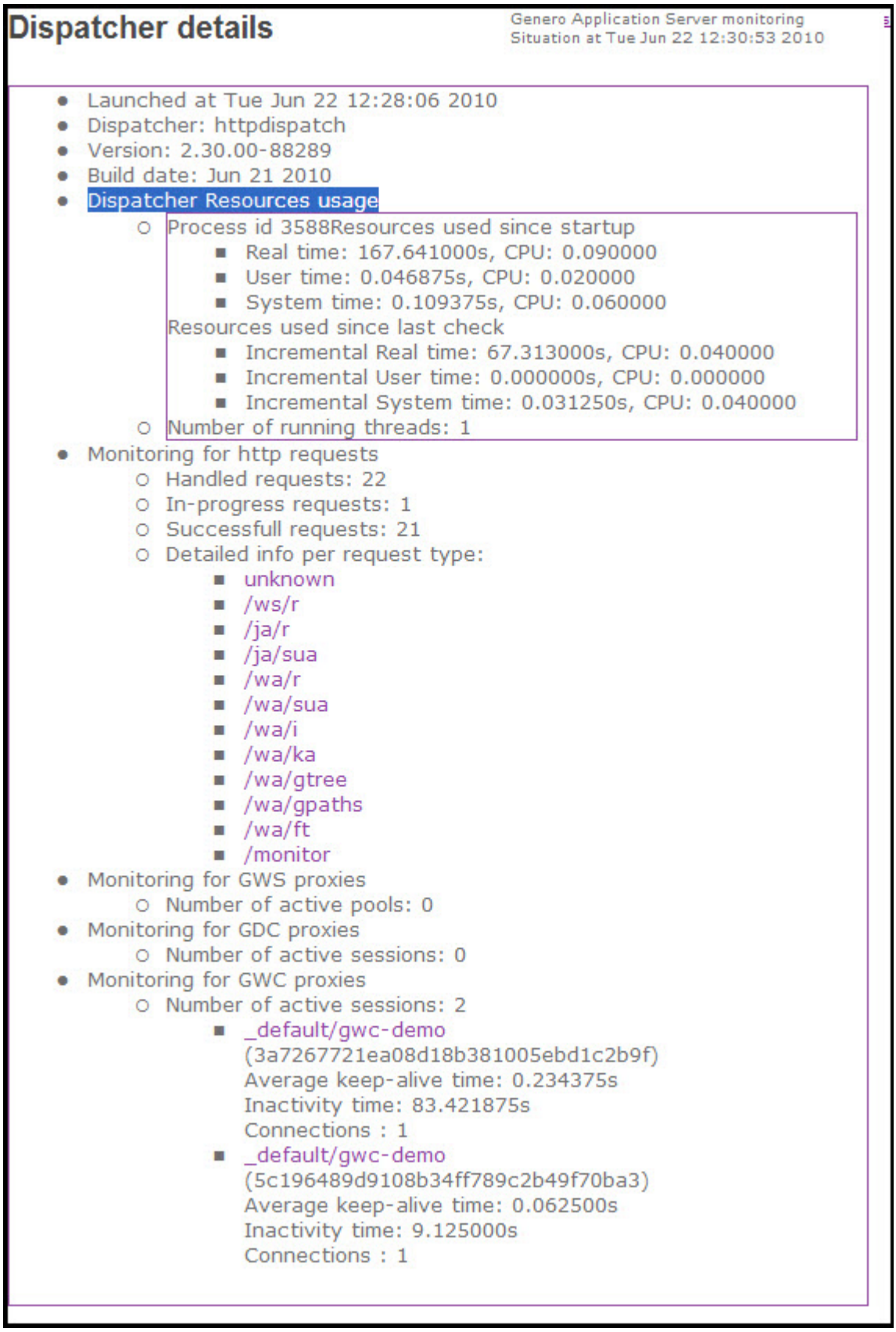


Figure 34: Monitoring: Dispatcher Resources usage

You can monitor detailed information per request type by clicking one of the request type links. For more information on what each of the request types are, see URIs Acknowledged by the GAS.

For example, to see statistics relating to the launching of a GWC application, you could select to view the statistics for the request type /wa/r.

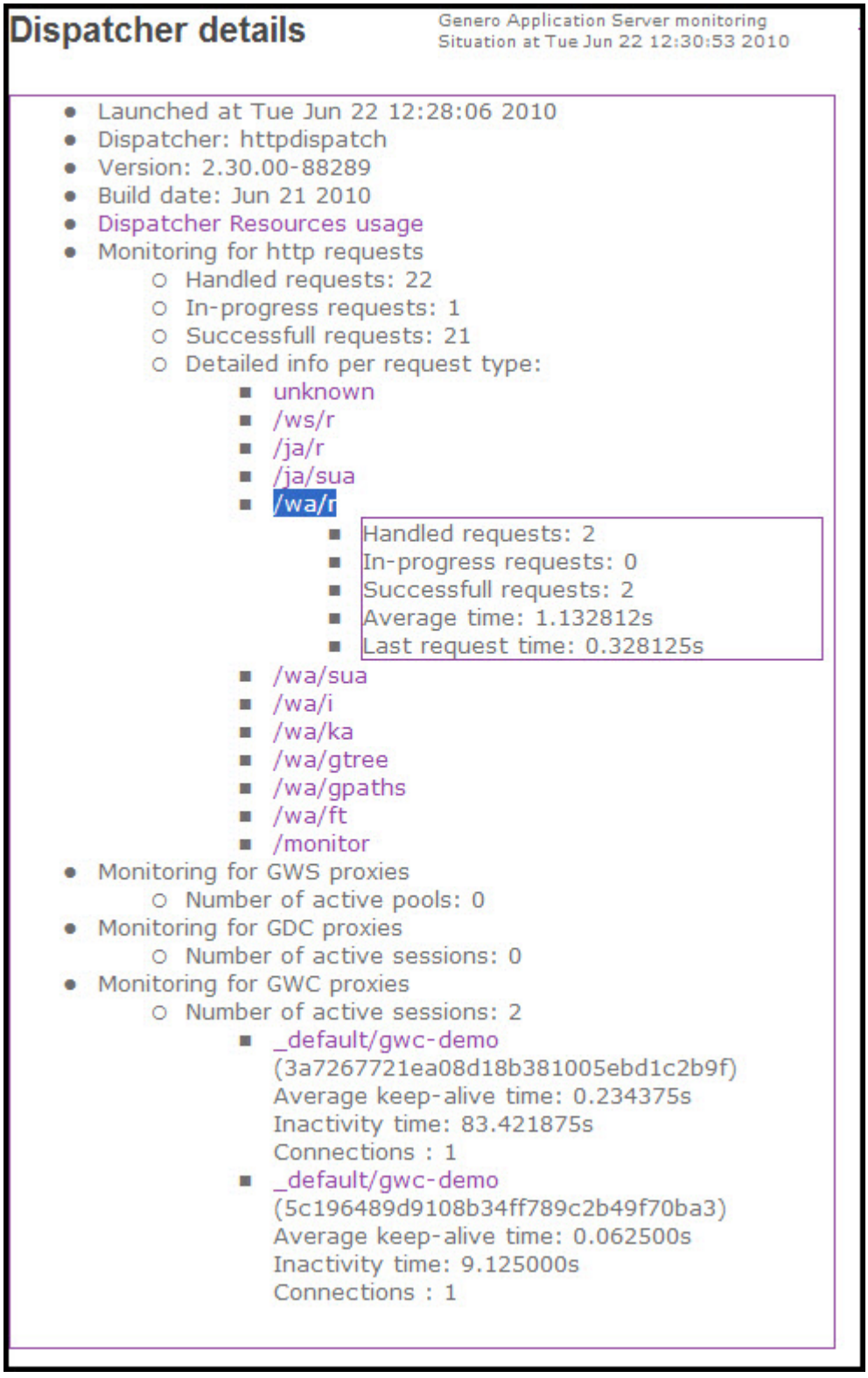


Figure 35: Monitor: Statistics for /wa/r

Step 2: Viewing information about the VMProxies

In addition to viewing statistics about the dispatcher, you can also view statistics about a session by clicking on session links under the Monitoring for GWS / GDC / GWC proxies sections. The statistics and information for the session - for the proxy - appear in a separate window. To return to the Dispatcher window, you must close the Session window.

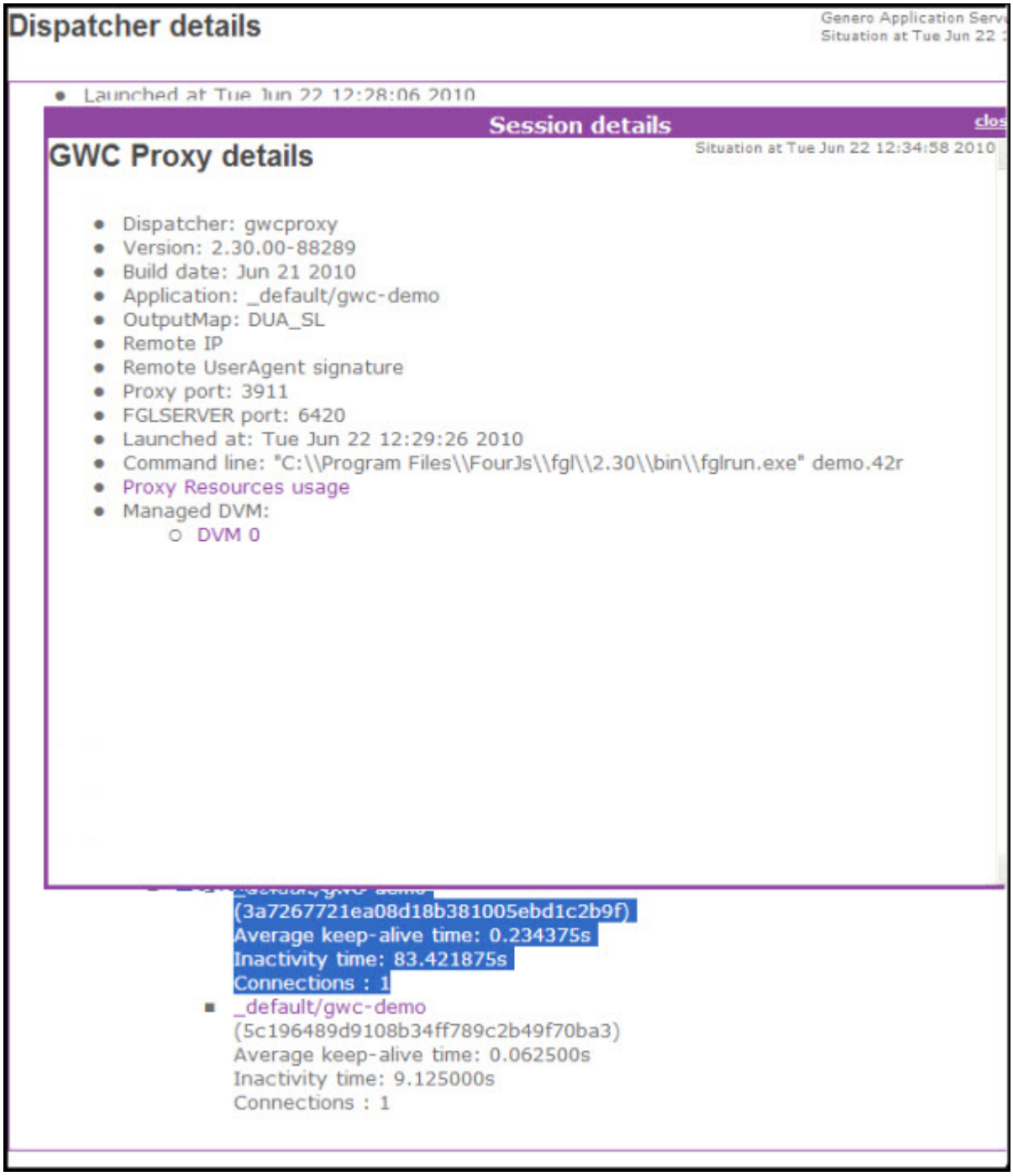


Figure 36: Viewing proxy statistics

Logging

The Genero Application Server creates separate log files for its dispatchers, proxies, and the DVMs started by those proxies.

- A log file is generated for each dispatcher. This log file captures incoming requests, the starting of proxies, responses sent, and system error messages.
- A log file is generated for each proxy started. A separate log file is generated for each proxy started.
- A log file is generated for each DVM started. DVM standard error and standard output are sent to the dedicated DVM log files.

When in development mode, the ending page for a GWC-JS Web application contains a link to the DVM log file.

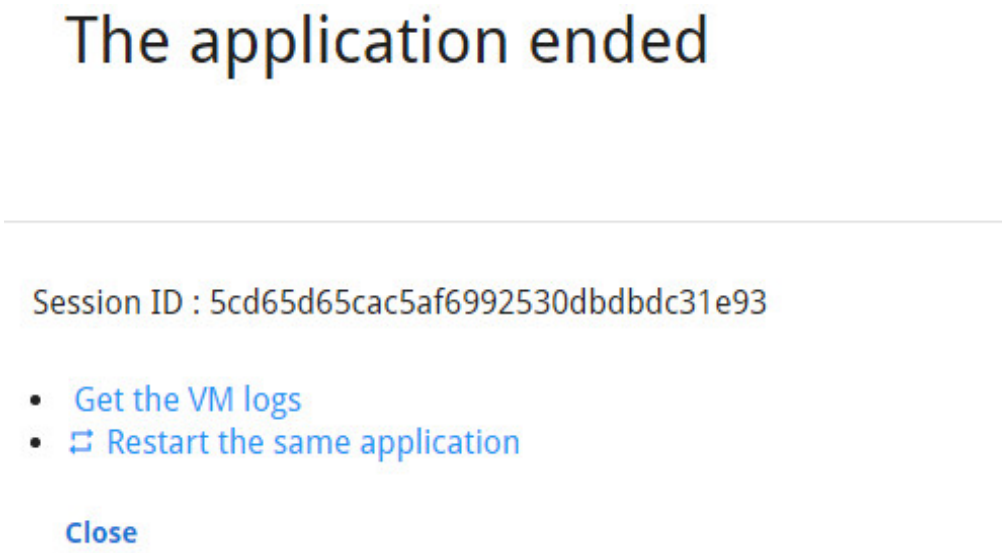


Figure 37: Example GWC-JS application ended page

Log files are also accessible from the Genero Application Server monitor via the (LOG) link next to the proxy name or PID display.

Note: See Table 5: appdata directories and files on page 40 for details on the location of log files.

Log file names

For the dispatcher log, the name specifies the type of dispatcher. Example:

- `httpdispatch.log`

For the proxy log, the name indicates the type of proxy. Examples:

- `uaproxy-<session-id>.log`
- `gwsproxy-<group>-<app>.log`

For the DVM log, if the DVM is started by the `gdcproxy` or `gwcproxy`, the name includes the *session-id*:

- `vm-<session-id>.log`

When working with Web services, a GWS proxy can spawn multiple DVMs. Each DVM gets its own log file. The log file is suffixed with a number from 0 to MAX_AVAILABLE-1. A log file is reused for new DVM logs if the previous DVM has finished, to avoid the accumulation of log files on disk.

- vm-*<group>*-*<app>*-*<number>*.log

Examples:

- `vm-demo-Calculator-0.log`
- `vm-demo-Calculator-1.log`

Manage the Genero Application Server log files

The GAS creates a log for each application session. As a result, you can end up with a lot of log files. You should have some plan for archiving and removing log files. For UNIX-based platforms, you can use utilities such as logrotate to compress and move log files. For Windows™, any program that can compress and archive log files can be used.

> **Note:** If using `logrotate` on Apache web server logs, lograte will start and stop the Apache server. When Apache restarts, it also starts a new `fastcgidispatch` process (see Apache: mod_fastcgi on page 86) while it may not stop the existing process. If you observe this behavior, you can set logrotate's `prerotate` script to get the pid of the running `fastcgidispatch` process, it should then be possible to stop the old `fastcgidispatch` process in the `postrotate` script, for more information see Logrotate.

Using the debugger

This section provides instructions for using the debugger for the httpdispatch.

In addition to using the instructions below, you can use the graphical debugger in Genero Studio. For more information on the graphical debugger, refer to the *Genero Studio User Guide*.

Using the Debugger for the GAS on the Windows™ platform

To run the FGL debugger, the dispatcher must open a DOS command or a xterm window and then run "fglrun -d".

1. In the GAS configuration file (default *%FGLASDIR%*`/etc/as.xcf`), change:

```
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">
 $(res.fgldir)\bin\fglrun.exe</RESOURCE>
```

   to:

```
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">
 c:\fjs\gas\debug.bat</RESOURCE>
```

   (Windows™) for example, where debug.bat contains `cmd /K start cmd`

2. In the application configuration file, change the DVM availability timeout value to allow you time to type your debug commands.

   For example, change:

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

   to:

```
<DVM_AVAILABLE>60</DVM_AVAILABLE>
```

   This change allows you 60 seconds in which to type your debug commands.

3. Restart the dispatcher.

   (The dispatcher must be restarted whenever you modify the application server configuration file in order for the changes to take effect.)

4. Enter the application URL in your browser.

   This opens a shell window.
5. Type the commands to run the application:

   `fglrun -d test.42r` <<< Opens the debugger tool and sets it on program test.42r.

   `(fgldb)b test:20` <<< Sets a break point at line 20.

   `(fgldb)run` <<< Runs the application.

   This refreshes the browser, like the FGL debugger does with the GDC.

   Tip:

   You can also run the dispatcher from the command line and override some of the settings for `res.dvm.wa`:

   `httpdispatch -E res.dvm.wa="cmd /K start cmd"` (Windows™)

Using the Debugger for the GAS on UNIX™

These instructions assume that you are operating within a graphical environment. If you are not operating within a graphical environment, simply enter the commands you want to process in the script.

To run the dispatcher, enter the following:

```
httpdispatch -E res.dvm.wa="/home/test/xterm.sh"
```

In the **xterm.sh** shell, you have: `/usr/bin/xterm` (the complete path to xterm).

This removes all of the options given by the dispatcher along with all error messages. A new xterm is opened. At this point, proceed as you would if you were running your applications from a Windows™ platform.

Performance tuning

These topics cover various performance tuning considerations for configuring your Web server, your Genero Application Server, and your Genero applications.

- Web server configuration: Keep Alive on page 158
- SPDY on page 158

Web server configuration: Keep Alive

Recommendations for the keep alive settings on the Web server.

To improve performance, we recommend that you turn on the HTTP Keep-Alive feature on the Web server.

Prior to version 2.50, if you were connecting to the GDC through the GAS (via the GDCProxy), it was recommended that the connection timeout be longer than 120 seconds. The recommendation is removed starting with version 2.50.

SPDY

SPDY (pronounced "SPeeDY") is an experimental network protocol created to transport Web content. The Genero Application Server is compatible with the SPDY protocol.

The main goal of SPDY is to reduce web page load time by:

- Multiplexing unlimited concurrent file transfers over a single connection. In other words, it allows many concurrent HTTP requests across a single TCP connection.
- Ordering file transfers by priority, preventing the channel from being congested with non-critical resources.

- Reduce bandwidth by compressing headers and eliminating unnecessary headers. For example, User-Agent, Host, Accept* are typically static and do not need to be resent.
- To enable the server to initiate communications with the client and push data to the client whenever possible.
- SSL is the underlying transport protocol:
  - better security
  - compatibility with existing network infrastructure
  - ensure communication across existing proxies is not broken

The SPDY protocol is built-in with Firefox (version 13 and greater) and Google Chrome.

It is the responsibility of the system administrator to install SDPY for their Web servers (where available). It is transparent to the GAS.

> **Important:** GAS 2.50 provides a built-in compression feature. If SPDY is used and compression is active in SPDY, you should disable the built-in GAS compression. SPDY will do a better job of compression as it can compress HTTP headers as well as HTTP request/response bodies.

For more information:

- SPDY home page
- SPDY white paper
- SPDY protocol
- Apache module for SPDY / installation instructions

Load balancing

One way to increase the capacity of the Genero Application Server (GAS) is to scale it out by deploying multiple instances of the GAS on different servers.

As the GAS is fully integrated with existing Web Servers, third-party tools can be used to implement load balancing. GAS applications can be load balanced using standard load balancing techniques, including software load balancers such as Windows™ Network Load Balancing or HAProxy, as well as hardware-based load balancing appliances.

> **Note:** There is no reason to have a Web server, its dispatcher, and the VMProxy on different machines. While the load balancer will balance load across different machines, on each machine you will find one Web server, one dispatcher, all attached proxies, and all attached DVMs. In production, you should think of the GAS as a group comprised of the web server, dispatcher and proxies, and DVMs.

This section discuss considerations for load balancing GAS applications on multiple instances. The provided examples illustrate simple configurations for Internet Information Services and Apache Web Servers.

- GAS requests on page 159
- Sessionless request processing on page 160
- Session-bound request processing on page 161
- Load Balancing Configuration Examples on page 165

GAS requests

The secret to load balancing GAS requests involves equally dispatching the requests between the different GAS instances.

However, all requests are not the same. Some requests must go to a specific GAS instance. These requests can be recognized through the session identifier that is part of their URL. The session identifier represents a GAS session, and because a GAS session is known by a single GAS instance only, all requests that are relative to that session must be driven to that specific GAS instance.

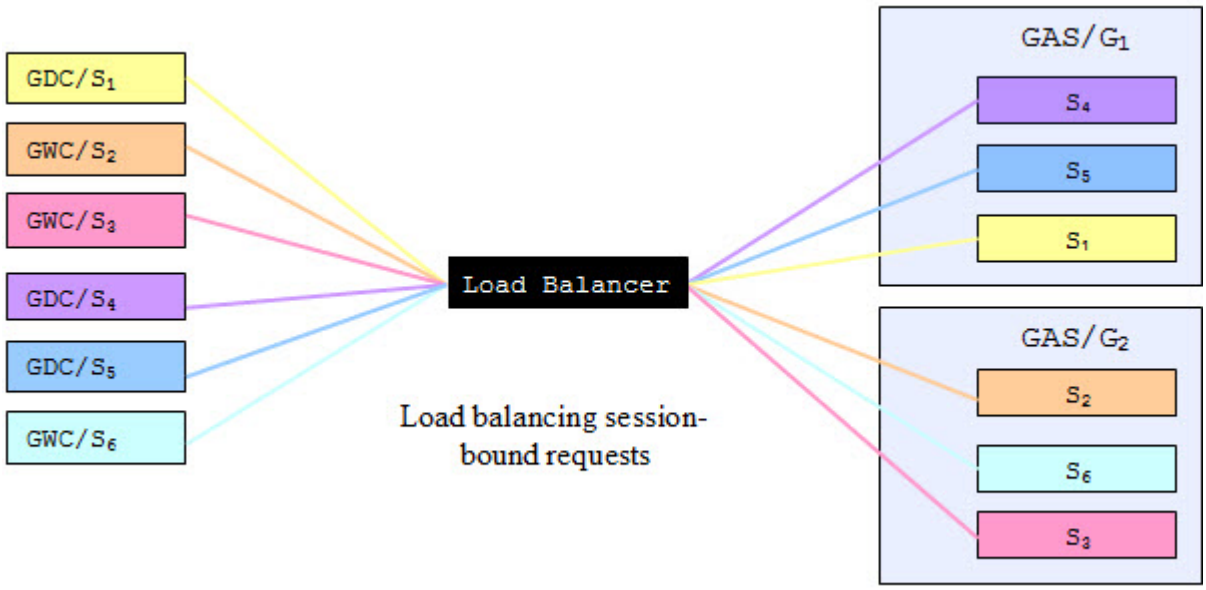


Figure 38: Load Balancing GAS Requests

Session-bound requests that share the same session identifier are part of a specific GAS session. Of the various kinds of GAS applications, only two create a session: GDC and GWC applications. Since GWS applications don't create sessions, GWS requests can always be routed to any of the available GAS instances. Sites that serve only GWS applications do not have to deal with session-bound request issues. However, if a site serves all kind of applications, it must be configured to serve session-bound requests only.

Managing the load balancing means handling the two types of requests:

- Load balancing sessionless requests among several GAS instances.
- Driving session-bound requests to the GAS instance that handles the session.

Sessionless request processing

Load balancing sessionless requests among several GAS instances does not require any particular knowledge of the incoming request.

Generally, the load balancing tool offers rules to do sessionless load balancing by itself. Depending on the tool, those rules may go from a simple round-robin algorithm to more sophisticated ones like choosing a server according to the available memory or to the time that it takes to respond to a network request.

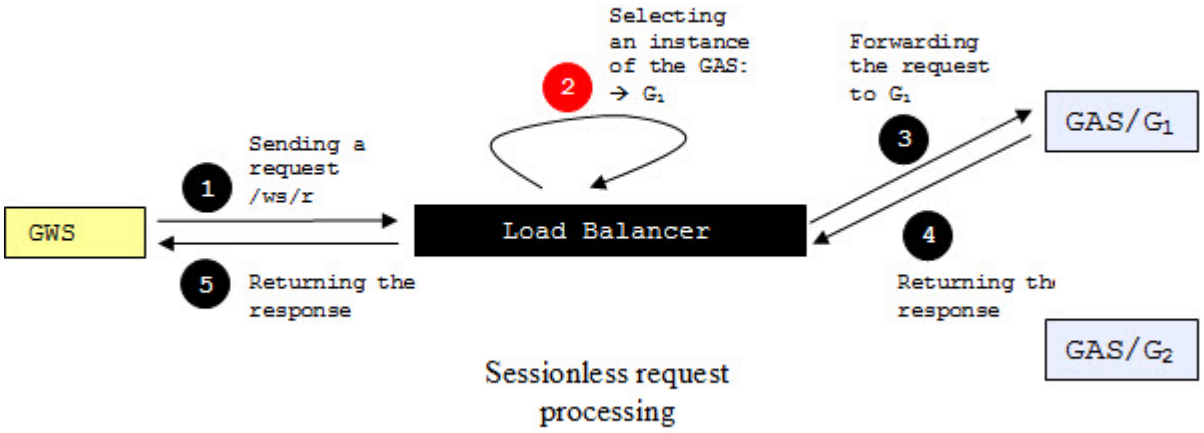


Figure 39: Sessionless Request Processing

Figure 39: Sessionless Request Processing on page 161shows the processing of a sessionless request. It takes the following steps:

1. A Web Service client tool sends a request to the load balancer server.
2. The load balancer server is configured to dispatch the requests over two GAS instances. It does not matter if those instances are installed on separate servers, or if they are on the same server configured to listen at different ports. The routing really depends only on the capabilities of the load balancing tool. In this example, it happens to choose the instance G1.
3. The request is forwarded to G1.
4. The response is returned to the load balancer server.
5. The response is returned to the client.

Session-bound request processing

With session-bound requests, more work has to be done in order to route session-bound requests to the GAS (G_1) instance that handles the session.

For them, G_1 must be retrieved from the data of the request. The load balancing tool does not record the currently active GAS sessions. Actually, it even doesn't know anything about GAS sessions at all. It offers other means to retrieve G_1: either through a cookie sent along with the request that will contain the identity of G_1 or through a part of the URL of the request that will link to G_1. This piece of information is added to the request that will create the session. This starting-session request is sessionless, and can therefore be processed by any of the available GAS instances. However, once that instance has been chosen (G_1 in our example) , the piece of information to identify G_1 will be added to the request and/or to its response in order to route following requests to G_1.

The following paragraphs discuss the two kinds of configuration for the load balancing of session-bound requests: cookie-based and path-based. For each method, first the processing of the starting-session request is shown, and then the processing of the session-bound requests is shown.

Cookie-based starting-session request processing

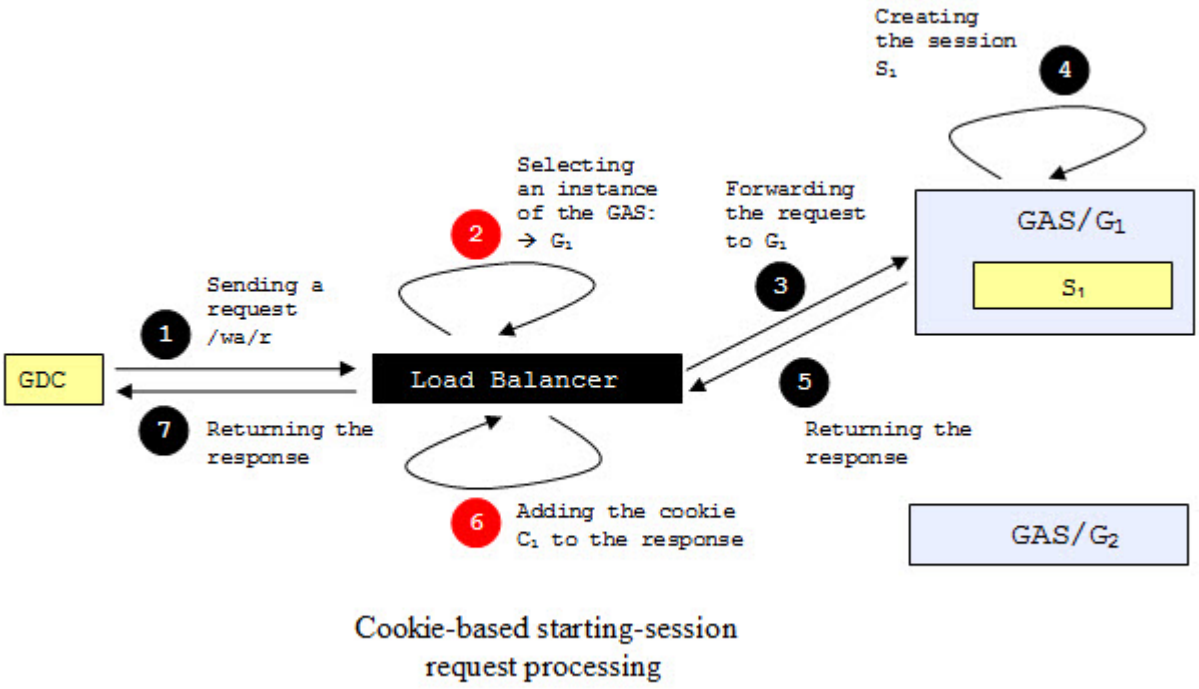


Figure 40: Cookie-based starting-session request processing

Figure 40: Cookie-based starting-session request processing on page 162shows the processing of a cookie-based starting-session request. It takes the following steps:

1. GDC (or a user agent on the behalf of GWC) sends a request to the load balancer server.
2. The load balancer server is configured to dispatch the requests over two GAS instances. In this example, it happens to choose the instance G_1.
3. The request is forwarded to G_1.
4. G_1 creates the session S_1.
5. The response is returned to the load balancer server.
6. A session cookie that holds the identity of G_1 is added to the response so that following requests will also contain this cookie.
7. The response is returned to the client.

Cookie-based session-bound request processing

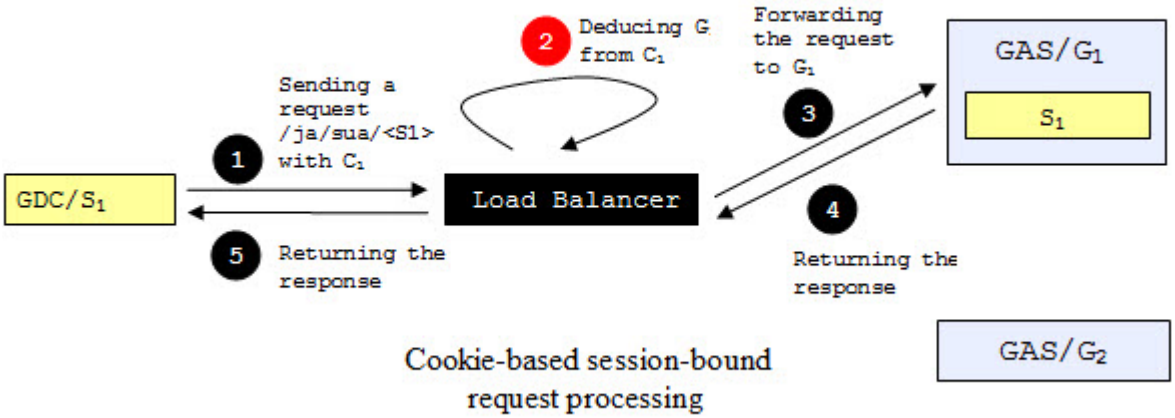


Figure 41: Cookie-based session-bound request processing

Figure 41: Cookie-based session-bound request processing on page 163shows the processing of a cookie-based session-bound request. It takes the following steps:

1. GDC (or a user agent in behalf of GWC) sends a request that contains the cookie C_1 to the load balancer server.
2. The load balancer server, thanks to C_1, recognizes that the request must be routed to G_1.
3. The request is forwarded to G_1.
4. The response is returned to the load balancer server.
5. The response is returned to the client.

> **Note:** Every request received by the load balancer server that contain C1 will be routed to G_1, whether it is a session-bound or a sessionless request.

Path-based starting-session request processing

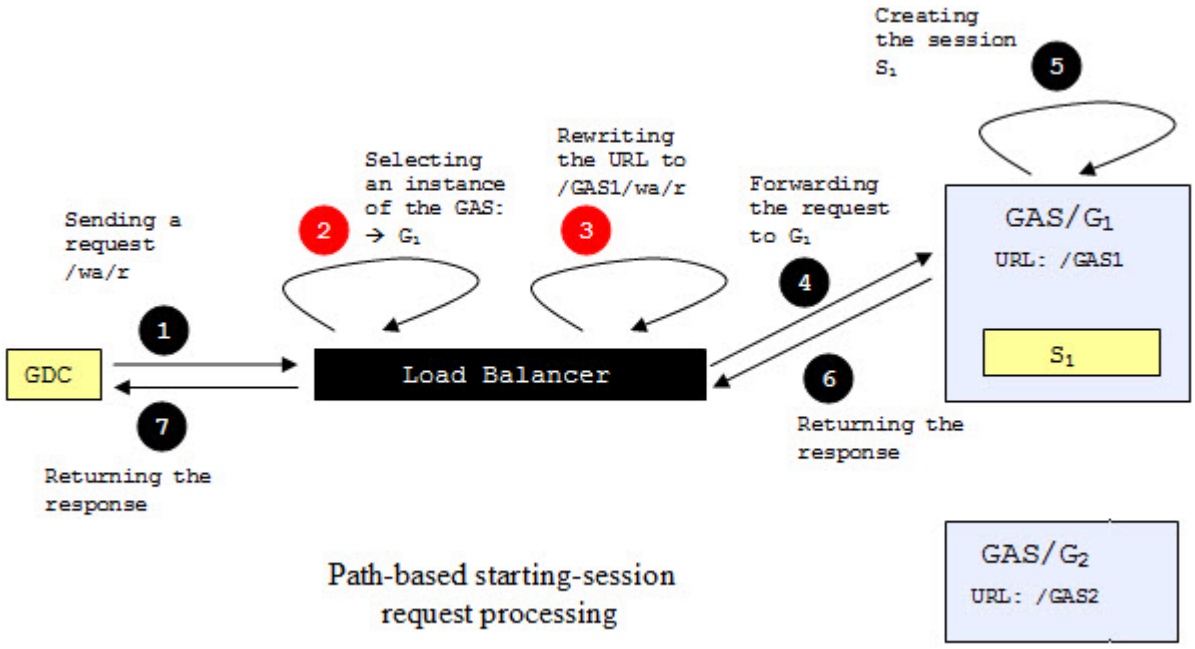


Figure 42: Path-based starting-session request processing

Figure 42: Path-based starting-session request processing on page 164shows the processing of a path-based starting-session request. It takes the following steps:

1. GDC (or a user agent in behalf of GWC) sends a request to the load balancer server.
2. The load balancer server is configured to dispatch the requests over two GAS instances. Each of these instances must be configured so that URLs that it handles begin with a part that is unique between all GAS instances, whether the GAS instances are installed on the same machine or different ones. In this example, G_1 handles requests whose URLs begin with /GAS1, and G2 handles requests whose URLs begin with /GAS2. In this example, it happens to choose the instance G_1.
3. Once G_1 has been chosen, the URL of the request is rewritten so that it begins with /GAS1. This prefix will be recognized by the GAS as the connector URI part of the URL. The GAS adds the connector URI to every URL that is part of the request responses.
4. The request is forwarded to G_1.
5. G_1 creates the session S_1.
6. The response is returned to the load balancer server.
7. The response is returned to the client.

Path-based session-bound request processing

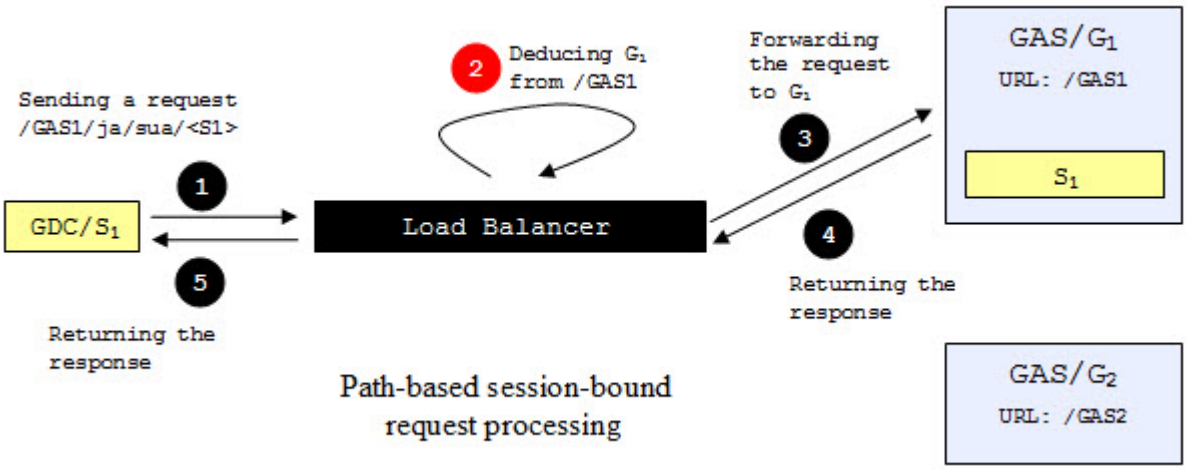


Figure 43: Path-based session-bound request processing

Figure 43: Path-based session-bound request processing on page 165shows the processing of a path-based session-bound request. It takes the following steps:

1. GDC (or a user agent in behalf of GWC) sends a request who's URL begins with /GAS1 to the load balancer server.
2. The load balancer server, thanks to the /GAS1 prefix, recognizes that the request must be routed to G_1.
3. The request is forwarded to G_1.
4. The response is returned to the load balancer server.
5. The response is returned to the client.

> **Note:** Every request received by the load balancer server that begins with /GAS1, respectively /GAS2, will be routed to G_1, respectively G2, whether it is a session-bound or a sessionless request.

Load Balancing Configuration Examples

Configuration examples depend on the tools available on the Web Server.

The following configuration examples are based on two sample architectures. The first architecture is configured as follows:

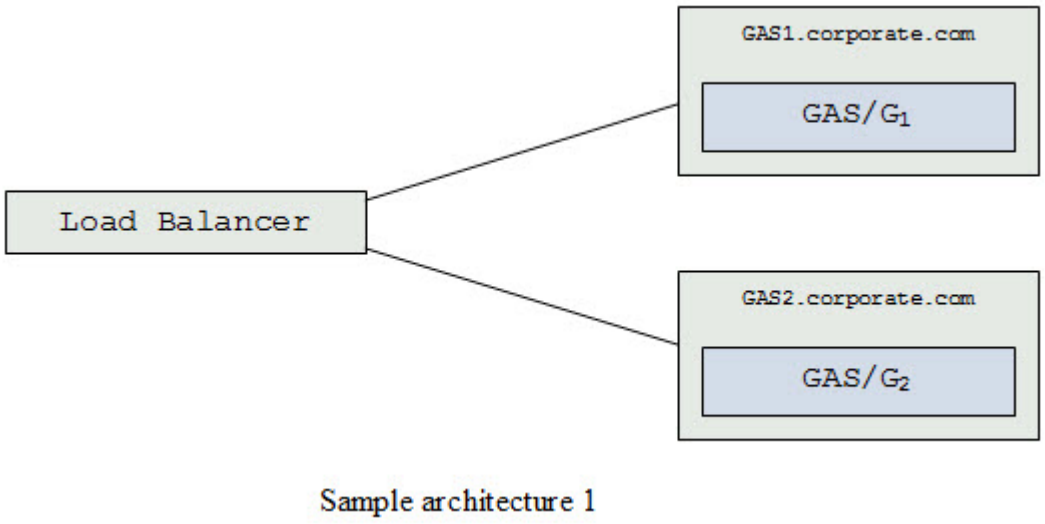


Figure 44: Load Balancing Sample Architecture 1

- Three machines.
- The load balancer machine contains all the tools to do the load balancing.
- Two machines each contain a GAS instance. Each GAS instance listens to the default HTTP port and are configured on the same virtual directory.
- The host names of the machines that contain the GAS instances are GAS1.corporate.com and GAS2.corporate.com.

The second architecture is configured as follows:

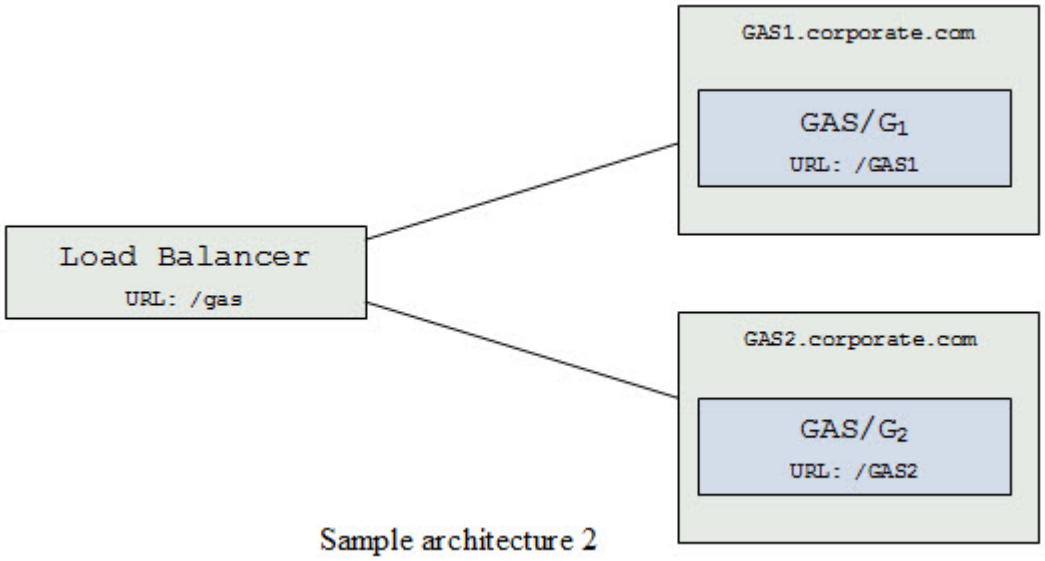


Figure 45: Load Balancing Sample Architecture 2

- It is the same configuration as Figure 44: Load Balancing Sample Architecture 1 on page 166, but the load balancer server is configured on the gas virtual directory and the GAS instances are configured on distinct virtual directories.

- Internet Information Services 5.x and 6.0 on page 167
- Internet Information Services 7.x on page 168
- Apache 1.3.x and 2.0.x on page 170
- Apache 2.2.x on page 171

Internet Information Services 5.x and 6.0

Internet Information Services (IIS) 5.x and 6.0 have no built-in tools to do load balancing. However, third party tools do exist.

One third-party tool is ISAPI_Rewrite, which aims to enable Apache mod_rewrite on IIS.

Sessionless requests

The following example shows how to configure the ISAPI_Rewrite filter to do load balancing for sessionless requests. It assumes the architecture illustrated by the sample architecture 1.

```
# Helicon ISAPI_Rewrite configuration file
# Version 3.1.0.66

RewriteEngine on

RewriteMap servers rnd:hosts.txt

RewriteRule ^/(.*) http://${servers:host}/$1 [P,L]
```

- These configuration entries can be put at the server level in the `httpd.conf` file.
- The `RewriteMap` instruction uses a round-robin algorithm to select a host based on the content of the `hosts.txt` file, which has following content:

  ```
  host GAS1.corporate.com|GAS2.corporate.com
  ```

- The `RewriteRule` instruction replaces the server host of incoming requests by the content of the `servers:host` variable, then forwards the request to the chosen machine.

For example, the `http://localhost/ws/r/echo` URL will be rewritten to `http://GAS1.corporate.com/ws/r/echo`, assuming that the GAS1.corporate.com server has been chosen.

Session-bound requests

The following example shows how to configure the ISAPI_Rewrite filter to do load balancing for session-bound requests. It assumes the architecture illustrated by the sample architecture 2. ISAPI_Rewrite doesn't support cookie-based request routing. The example illustrates path-based session-bound request routing.

```
# Helicon ISAPI_Rewrite configuration file
# Version 3.1.0.66

RewriteEngine on

RewriteMap servers rnd:vdirs.txt

RewriteRule ^/gas/(.*) /${servers:vdir}/$1
RewriteRule ^/GAS1/(.*) http://GAS1.corporate.com/GAS1/$1 [P,L]
RewriteRule ^/GAS2/(.*) http://GAS2.corporate.com/GAS2/$1 [P,L]
```

- These configuration entries can be put at the server level in the `httpd.conf` file.
- The `RewriteMap` instruction uses a round-robin algorithm to set the value of the `servers:vdir` variable based on the content of the `vdirs.txt` file, which has following content:

  ```
  vdir GAS1|GAS2
  ```

- For starting-session requests, the `RewriteRule` instruction replaces the `gas` part of the URL by the content of the `servers:vdir` variable, which is either `GAS1` or `GAS2`; then other rewrite rules will be applied to set the host. For session-bound requests, the rule will not match so the rule doesn't apply.
- The host name of requests who's URL begin with `/GAS1/` will be set to GAS1.corporate.com.

- The host name of requests who's URL begin with `/GAS2/` will be set to GAS2.corporate.com.

For example, the `http://localhost/gas/ua/r/gwc-demo` URL will be rewritten to `http://localhost/GAS2/ua/r/gwc-demo`, assuming that the `servers:vdir` variable contains `GAS2` at this time, then will be rewritten to `http://GAS2.corporate.com/GAS2/wa/r/gwc-demo`. Likewise, the `http://localhost/GAS2/wa/sua/93837374/1` URL will be rewritten to `http://GAS2.corporate.com/GAS2/wa/sua/93837374/1`.

Internet Information Services 7.x
Internet Information Services (IIS) 7.x provides load balancing tools through IIS extensions.

The IIS extensions are:

- Application Request Routing
- IIS URL Rewrite

Both tools are required to do the job.

The following examples show the resulting configuration entries. It can be done either by editing the configuration files manually or by using the IIS manager user interface. For more information on how to use the IIS manager user interface, see HTTP Load Balancing using Application Request Routing.

Sessionless requests

The following example shows how to configure IIS 7.x to do load balancing for sessionless requests. It assumes the architecture illustrated by the sample architecture 1.

```
<configuration>
  ...
  <system.webServer>
    ...
    <rewrite>
      <globalRules>
        ...
        <rule name="ARR_GASFarm_loadbalance" enabled="true"
        patternSyntax="Wildcard" stopProcessing="true">
          <match url="*" />
          <conditions />
          <action type="Rewrite" url="http://GASFarm/{R:0}" />
        </rule>
      </globalRules>
    </rewrite>

    <proxy enabled="true" />

  </system.webServer>
  ...
  <webFarms>
    ...
    <webFarm name="GASFarm" enabled="true">
      <server address="GAS1.corporate.com" enabled="true">
        <applicationRequestRouting weight="100" />
      </server>
      <server address="GAS2.corporate.com" enabled="true">
        <applicationRequestRouting weight="100" />
      </server>
      <applicationRequestRouting>
        <loadBalancing algorithm="WeightedRoundRobin" />
        <affinity useCookie="false" />
      </applicationRequestRouting>
    </webFarm>
  </webFarms>
  ...
```

```
</configuration>
```

- These configuration entries can be put at the server level in the `applicationHost.config` file.
- In the globalRules section of the rewrite section, the rule named `ARR_GASFarm_loadbalance` tells the module to rewrite all incoming requests so that they are routed to the `GASFarm` web farm.
- The proxy is enabled in order to forward the requests to the two GAS servers.
- The web farm named `GASFarm` is declared with the two GAS servers.

For example, the `http://localhost/ws/r/echo` URL will be rewritten to `http://GAS1.corporate.com/ws/r/echo`, assuming that the GAS1.corporate.com server has been chosen.

Session-bound requests

The following example shows how to configure IIS 7.x to do load balancing for session-bound requests. It assumes the architecture illustrated by the sample architecture 1. The example illustrates cookie-based request routing.

```
<configuration>
  ...
  <system.webServer>
    ...
    <rewrite>
      <globalRules>
        ...
        <rule name="ARR_GASFarm_loadbalance" enabled="true"
        patternSyntax="Wildcard" stopProcessing="true">
          <match url="*" />
          <conditions />
          <action type="Rewrite" url="http://GASFarm/{R:0}" />
        </rule>
      </globalRules>
    </rewrite>

    <proxy enabled="true" />

  </system.webServer>
  ...
  <webFarms>
    ...
    <webFarm name="GASFarm" enabled="true">
      <server address="GAS1.corporate.com" enabled="true">
        <applicationRequestRouting weight="100" />
      </server>
      <server address="GAS2.corporate.com" enabled="true">
        <applicationRequestRouting weight="100" />
      </server>
      <applicationRequestRouting>
        <loadBalancing algorithm="WeightedRoundRobin" />
        <affinity useCookie="true" />
      </applicationRequestRouting>
    </webFarm>
  </webFarms>
  ...
</configuration>
```

- These configuration entries can be put at the server level in the `applicationHost.config` file.
- In the globalRules section of the rewrite section, the rule named `ARR_GASFarm_loadbalance` tells the module to rewrite all incoming requests so that they are routed to the `GASFarm` web farm.
- The proxy is enabled in order to forward the requests to the two GAS servers.
- The web farm named `GASFarm` is declared with the two GAS servers.

- The `useCookie` attribute of the `affinity` element is set to true; actually, this is the only difference compared to the sessionless request routing.

For example, the `http://localhost/ua/r/gwc-demo` URL will be rewritten to `http://GAS1.corporate.com/ua/r/gwc-demo`, assuming that the GAS1.corporate.com server has been chosen.

Apache 1.3.x and 2.0.x
Apache HTTP Server versions 1.3.x and 2.0.x provide load balancing by using mod_rewrite Apache module.

Refer to the following:

- Apache 1.3: Module mod_rewrite: URL Rewriting Engine
- Apache 2.0: Module mod_rewrite

All configurations examples explained here work on both Apache version 1.3.x and 2.0.x

Sessionless requests

The following example shows how to configure the Apache mod_proxy module to do load balancing for sessionless requests. It assumes the architecture illustrated by the sample architecture 1.

```
# Apache configuration file
<IfModule mod_rewrite.c>

RewriteEngine on

RewriteMap servers rnd:hosts.txt

RewriteRule ^/(.*) http://${servers:host}/$1 [P,L]

</IfModule>
```

- These configuration entries can be put at the server level in the `httpd.conf` file.
- The `RewriteMap` instruction uses a round-robin algorithm to select a host based on the content of the hosts.txt file, which has following content:

  ```
  host GAS1.corporate.com|GAS2.corporate.com
  ```

- The `RewriteRule` instruction replaces the server host of incoming requests by the content of the `servers:host` variable, then forwards the request to the chosen machine.

For example, the `http://localhost/ws/r/echo` URL will be rewritten to `http://GAS1.corporate.com/ws/r/echo`, assuming that the GAS1.corporate.com server has been chosen.

Session-bound requests

The following example shows how to configure the Apache mod_proxy module to do load balancing for session-bound requests. It assumes the architecture illustrated by the sample architecture 2. The example illustrates path-based session-bound request routing.

```
# Apache configuration file
<IfModule mod_rewrite.c>

RewriteEngine on

RewriteMap servers rnd:vdirs.txt

RewriteRule ^/gas/(.*) /${servers:vdir}/$1
RewriteRule ^/GAS1/(.*) http://GAS1.corporate.com/GAS1/$1 [P,L]
RewriteRule ^/GAS2/(.*) http://GAS2.corporate.com/GAS2/$1 [P,L]
```

```
</IfModule>
```

- These configuration entries can be put at the server level in the `httpd.conf` file.
- The `RewriteMap` instruction uses a round-robin algorithm to set the value of the `servers:vdir` variable based on the content of the vdirs.txt file, which has following content:

  ```
  vdir GAS1|GAS2
  ```

- For starting-session requests, the `RewriteRule` instruction replaces the `gas` part of the URL by the content of the `servers:vdir` variable, which is either `GAS1` or `GAS2`; then other rewrite rules will be applied to set the host. For session-bound requests, the rule will not match so the rule doesn't apply.
- The host name of requests who's URL begin with `/GAS1/` will be set to GAS1.corporate.com.
- The host name of requests who's URL begin with `/GAS2/` will be set to GAS2.corporate.com.

For example, the `http://localhost/gas/ua/r/gwc-demo` URL will be rewritten to `http://localhost/GAS2/ua/r/gwc-demo`, assuming that the `servers:vdir` variable contains `GAS2` at this time, then will be rewritten to `http://GAS2.corporate.com/GAS2/ua/r/gwc-demo`. Likewise, the `http://localhost/GAS2/ua/sua/93837374/1` URL will be rewritten to `http://GAS2.corporate.com/GAS2/ua/sua/93837374/1`.

Apache 2.2.x
Apache HTTP Server versions 2.2.x provides load balancing tools through two Apache modules

The two modules are:

- Apache 2.2: Module mod_proxy_balancer
- Apache 2.2: Apache Module mod_headers

The mod_proxy_balancer module provides support of HTTP requests load balancing. The mod_headers module provides HTTP cookies management needed by the Cookie-based request processing. For Sessionless request processing, the mod_headers module is not required.

Sessionless requests

The following example shows how to configure Apache 2.2.x to do load balancing for sessionless requests. It assumes the architecture illustrated by Session-bound request processing on page 161.

```
<IfModule mod_proxy.c>
  ProxyPass / balancer://GASFarm
  <Proxy balancer://GASFarm>
    BalancerMember http://GAS1.corporate.com
    BalancerMember http://GAS2.corporate.com
  </Proxy>
</IfModule>
```

- These configuration entries can be put at the server level in the `httpd.conf` file.
- The `ProxyPass` instruction tells the module to rewrite all incoming requests so that they are routed to the `GASFarm` web farm.
- The `<Proxy balancer://GASFarm>...</Proxy>` block defines which are the members of `GASFarm` web farm.
- The `BalancerMember` instructions declare two HTTP web server members of `GASFarm` web farm: `GAS1.corporate.com` and `GAS2.corporate.com`.

For example, the `http://localhost/ws/r/echo` URL will be rewritten to `http://GAS1.corporate.com/ws/r/echo`, assuming that the GAS1.corporate.com server has been chosen.

Session-bound requests

The following example shows how to configure Apache 2.2.x to do load balancing for session-bound requests. It assumes the architecture illustrated by Session-bound request processing on page 161. The example illustrates cookie-based request routing.

```
<IfModule mod_proxy.c>
  ProxyPass / balancer://GASFarm stickysession=GAS_AFFINITY
  <Proxy balancer://GASFarm>
    BalancerMember http://GAS1.corporate.com route=GAS1
    BalancerMember http://GAS2.corporate.com route=GAS2
  </Proxy>
</IfModule>

<IfModule mod_headers.c>
  Header add Set-Cookie "GAS_AFFINITY=balancer.%{BALANCER_WORKER_ROUTE}e;
  path=/; domain=.corporate.com" env=BALANCER_WORKER_ROUTE
</IfModule>
```

- These configuration entries can be put at the server level in the `httpd.conf` file.
- The `ProxyPass` instruction tells the module to rewrite all incoming requests so that they are routed to the `GASFarm` web farm. The `stickysession` parameter gives the name of the cookie to use to retrieve the GAS that holds the session.
- The `<Proxy balancer://GASFarm>...</Proxy>` block declares which are the members of `GASFarm` web farm.
- The `BalancerMember` instructions declare two HTTP web server members of the `GASFarm` web farm: `GAS1.corporate.com` and `GAS2.corporate.com`. The `route` parameter gives the name of the route associated with that member. If that route name is found in the request cookie, then that member will be chosen.
- The `Header` instruction will set, in the HTTP response, the cookie named GAS_AFFINITY to value `balancer.`*`BALANCER_WORKER_ROUTE`*. The *`BALANCER_WORKER_ROUTE`* variable is assigned to the route of the `GASFarm` member that has be chosen for the current request. Lastly, the `env` parameter tells the module to set that cookie only if the *`BALANCER_WORKER_ROUTE`* variable is defined.

For example, the `http://localhost/ua/r/gwc-demo` URL will be rewritten to `http://GAS1.corporate.com/ua/r/gwc-demo`, assuming that the GAS1.corporate.com server has been chosen.

Developing Web applications

The Genero Application Server allows you to deliver Web applications across a variety of clients, as well as Web services.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Genero Web Client for JavaScript (GWC-JS)

These topics provide information about the Genero Web Client for JavaScript (GWC-JS) client.

Developing and deploying Web applications can be as simple as configuring the Genero Application Server to launch the application, or it may require some slight modifications to the existing applications to work properly given the limitations of what an application can do from a browser. A general knowledge of how the Genero Web Client for JavaScript operates can be helpful in the planning and deploying of Web applications.

What is GWC-JS?

The Genero Web Client for JavaScript is a Web client that delivers your applications over the Web using the latest technologies for web application development.

The Genero Web Client for JavaScript (GWC-JS) is a Web client that runs in a browser. It is a JavaScript client that works with the well-known and widely-used frameworks like node.js and sass. Its UI is adapted from concepts of material design inspired by the material design recommendations from Google. This innovative approach opens up limitless possibilities for you to customize the GWC-JS features and styles according to your requirements.

The GWC-JS allows you to deliver true Web applications developed in the Genero Business Development Language (BDL). Having the underlying source written in Genero BDL means that the GWC-JS is flexible enough to let you build a full range of Web applications, from simple to corporate applications, with few limitations on what you can achieve.

GWC-JS brings Genero applications to the Internet world with the capacity of integration in a Web site. It provides you with the opportunity of working with state of the art web technologies.

Why deliver an application as a Web application?

- Web application deployment is easier and cheaper than desktop application deployment.

- The end user requires a browser; no software needs to be installed on the client by the end user.

GWC-JS principles

GWC-JS has evolved from the earlier GWC for HTML5 client, but unlike the earlier version, which used themes comprised of *templates* and *snippet sets* to create dynamic web pages that were rendered on the Application Server for delivery to the client browser, the GWC-JS is a JavaScript client that works with `Node` technology. The GWC-JS client supports Genero real-time Web applications based on the following principles:

- Nothing is computed or rendered on the Application Server side.
- Genero applications are rendered based on DVM instructions (not HTML).
- GWC-JS interprets the same protocol as Genero Desktop Client (GDC) to build web interfaces.
- GWC-JS project sources are provided in the GAS package. You can modify and adapt these sources (JavaScript, HTML and CSS) to customize your application's features and styles to meet your needs, see Configuring your Environment on page 198.

GWC-JS can deliver an application to any device as long as your browser supports HTML5.

Key Players

When working with applications deployed in a Web environment, you will need to identify or add team members who are proficient in various Web technologies. Many of these technologies will be unfamiliar to your traditional Genero BDL application developer.

The key players involved in developing Web-based applications are listed by area:

Table 12: Web application development key players

Area	Player Responsibility
Application Design	Responsible for the customization of aspects of the application within GWC-JS by adding and modifying CSS using Sass to influence the look-and-feel of the application. Members of this team should be proficient with HTML, CSS, Sass and Grunt.
Application Development	Responsible for the development of the Genero application, concentrating on the business logic. Members of this team should be proficient with Genero BDL.
Advanced Production	. Responsible for the additional functionality and navigation added to an application through the use of the CSS to link BDL form objects and JavaScript™ and to define the behavior. Members of this team should be proficient with the non-Genero languages and tools that can be involved in customization, such as HTML, CSS, Node.js, Sass, Grunt, and JavaScript™.
Deployment and Infrastructure	Responsible for the complete GWC-JS solution from a component perspective: the installation and configuration of the application server and Web server; the communication between the user agent, Web server, application server, DVM, and database server. Members of this team should be proficient in working on the different platforms and operating systems where the application will reside and proficient in administration of the Web server.

It is rare that a single person fulfills the requirements demanded in each of these areas.

If you are working with the Genero Web Client for JavaScript, you must have some understanding of Web technologies like HTML, XML, style sheets, Node.js, Sass, Grunt, and JavaScript™. You can find Web standards at the World Wide Web Consortium (W3C) site.

> **Tip:** For basic tutorials about Web standards, visit http://www.w3schools.com. While the w3schools site provides basic, free tutorials, it is a private venture and not affiliated with the W3C. It provides a starting point for learning new Web technologies.

How GWC-JS works

The GWC-JS Web client uses AUI tree, controllers and widgets to create and manage Document Object Model (DOM) elements. JavaScript uses the DOM to build and manage the UI interface dynamically.

How GWC-JS works

The GWC-JS Web client interface is built dynamically using JavaScript. When the GWC-JS is launched, it receives an AUI tree from the GAS. Effectively each node of the AUI tree is managed by a GWC-JS controller and widget.

The GWC-JS uses its *controller* and *widget* elements to create the HTML DOM. Through DOM elements, JavaScript is able to dynamically implement the elements of the HTML pages of the UI by responding to changes to their properties and listening for events that change their behavior. See this illustrated in Figure 46: GWC-JS Controller and Widget Function on page 175.

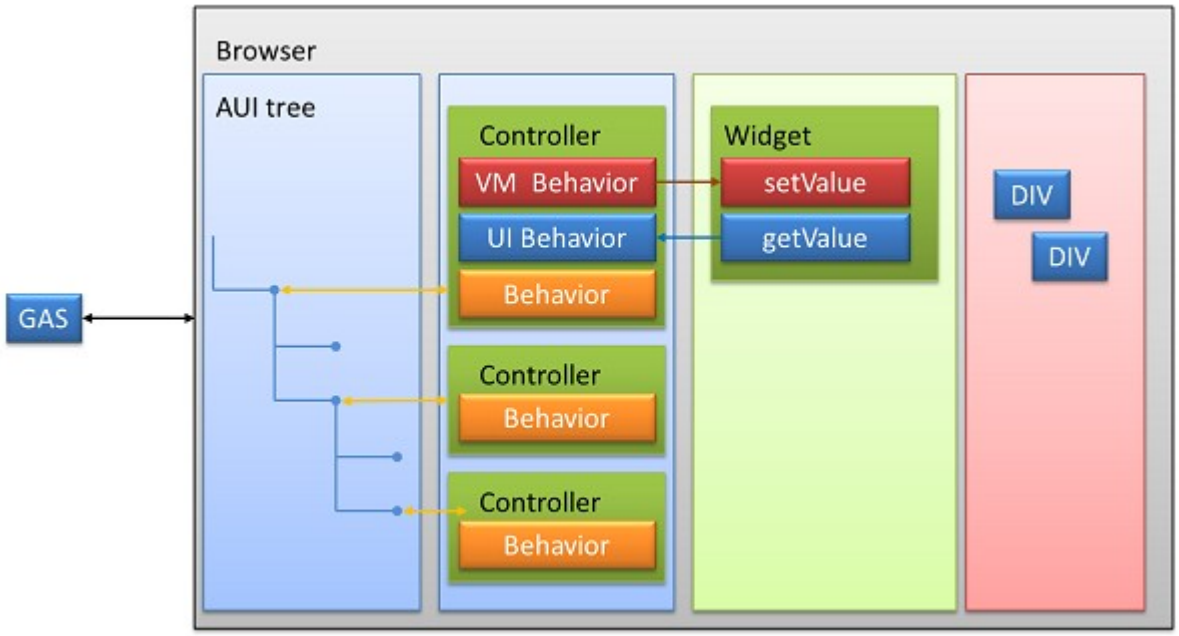


Figure 46: GWC-JS Controller and Widget Function

GWC-JS Controller

The GWC-JS controller and widget elements are interdependent but have specific and distinct roles. The JavaScript controller (`_Controller.js`) plays a role in:

- Creating the widget.
- Maintaining its behaviors.

Behaviors involves mapping one or more attributes of the AUI tree to an aspect of the widget. For example, the `color` and `reverse` attributes control a widget's background color. To reflect the two-way interaction

between the Web client and the application, there are two types of behavior that a controller needs to maintain:

- VM behavior: the controller is listening to the DVM and applying the changes received from DVM instructions.
- UI behavior: the controller is also listening to the UI and sending UI modification to the DVM.

Note: These built-in behaviors are internal and not meant to be modified by users.

GWC-JS Widget

The JavaScript widget (`_Widget.js`) plays a role in updating the DOM tree to reflect the application state. It has:

- APIs to manipulate the DOM.
- Listeners to react to DOM events.
- Understanding GWC-JS widgets on page 194

Features and limitations

While a Genero application largely reacts the same across the different Front End clients (GWC-JS, GDC, and so on), there are limitations for applications.

These tables list the features and limitations for Genero Front Ends. This is to help you determine which Front End is the most suitable for your applications.

Note: Some features of the new Genero Web Client for JavaScript (GWC-JS) are currently not yet available. As development work continues, we will update this page as new versions of the GWC-JS are released.

Legend

- X : available
- X* : available with limitations
- NA : not available

Table 13: Rendering: Features and limitations for Genero Front Ends

	GDC	GWC-HTML5	GWC-JS
Look	windows manager theme	custom theme	custom theme
Layout	X	X*	X*
MDI windows	X	NA	NA
Stacked window	X	X*	X
Styles: .4st colors, fonts (common styles)	X	X	X
Styles: .4st positioning	X	X	X
Widgets Customization	NA	X	X
Web Components	X	X	X
Traditional GUI Mode	X	X	NA

Table 14: Behavior: Features and limitations for Genero Front Ends

	GDC	GWC-HTML5	GWC-JS
Synchronous triggers	X	X	X
Drag and Drop	X	X	X
Keying: Dialogtouched	X	X	X
Keying: Type ahead	X	X	NA
Keying: Accelerator keys	X	X	X
Keying: Local actions	X	X*	X*
Natural accelerator	X	X*	X*
Cursor: fgl_dialog_getselectionend	X	NA	X
Cursor: fgl_dialog_setselection	X	NA	X
Cursor: fgl_dialog_getcursor	X	NA	X
Cursor: fgl_dialog_setcursor	X	NA	X
Cursor: fgl_getcursor	X	NA	X
Widgets	X	X*	X*
RIP widgets	X	NA	NA
Table: Resizing columns	X	X	X
Table: Display/Hide columns	X	X	X
Table: Frozen columns	X	X	X
Dockable toolbars/menus	X	NA	NA
GRV	X	NA	NA
CANVAS	X	X	X
Richtext	X	X	NA
PictureFlow	X	X	NA

Table 15: Interaction (with third party): Features and limitations for Genero Front Ends

	GDC	GWC-HTML5	GWC-JS
File transfer	X	X	X
Session variables/Cookies	NA	X	X
Stored settings	X	X	NA
Front End Call	X	X*	X*

Table 16: Deployment: Features and limitations for Genero Front Ends

	GDC	GWC-HTML5	GWC-JS
Front End System	All / Internet Explorer	HTML5 Browsers	HTML5 Browsers

	GDC	GWC-HTML5	GWC-JS
Connections: HTTP / HTTPS	X	X	X
Connections: rlogin / telnet / SSH / SSH2	X*	NA	NA
Single Sign On	X	X	X

- Non-supported hot keys on page 178

Non-supported hot keys
Some hot key combinations are not supported by specific browsers for the Genero Web Client for JavaScript.

The table below identifies which hot keys are NOT supported by browser type, for applications delivered by the GWC-JS.

> **Important:** An '**NS**' in a browser's column indicates that the hot key combination is **NOT supported** for that browser.
>
> **Important:** Details about hot key support for Microsoft™ Edge and Safari browsers will be added in a future update to this topic.

Table 17: Hot keys NOT supported by GWC-JS by browser

Hot key combination	Microsoft™ Internet Explorer	Microsoft™ Edge	Google Chrome	Safari
Menu	NS		NS	
Windows	NS		NS	
Shift - Menu	NS			
Shft - Ctrl	NS			
Ctrl - F4	NS			
Ctrl - Menu	NS			
Ctrl - 0 (zero)	NS			
Ctrl - + (plus sign)	NS			
Ctrl - + (plus sign on the Numeric pad)	NS			
Ctrl - - (minus sign on the Numeric pad)	NS			
Ctrl - N			NS	
Ctrl - O (letter O)	NS			
Ctrl - P	NS			
Ctrl - T			NS	
Ctrl - W			NS	
Alt - F4	NS			
Alt - Enter	NS			
Alt - Space	NS		NS	
Alt - A	NS			

Hot key combination	Microsoft™ Internet Explorer	Microsoft™ Edge	Google Chrome	Safari
Alt - C	NS			
Alt - E	NS			
Alt - F	NS			
Alt - H	NS			
Alt - T	NS			
Alt -V	NS			
Alt - X	NS			
Alt - Z	NS			
Ctrl - Shift - Menu	NS			
Ctrl - Shift - + (plus sign)	NS			
Ctrl - Shift - + (plus sign on the Numeric pad)	NS			
Ctrl - Shift - - (minus sign on the Numeric pad)	NS			
Ctrl - Shift - N			NS	
Ctrl - Shift - T			NS	
Ctrl - Shift - W			NS	
Ctrl - Alt - Menu	NS			
Ctrl - Alt - Delete	NS		NS	
Ctrl - Alt - Delete (on the Numeric pad)	NS		NS	
Ctrl - Alt - Arrows (up/down/left/ right)	NS		NS	
Ctrl - Alt - F12	NS		NS	
Ctrl - Alt - 3			NS	
Ctrl - Alt - 8			NS	
Ctrl - Alt - Shift - Menu	NS			
Ctrl - Alt - Shift - 2 (on the Numeric pad)	NS			
Ctrl - Alt - Shift - 4 (on the Numeric pad)	NS			
Ctrl - Alt - Shift - 6 (on the Numeric pad)	NS			
Ctrl - Alt - Shift - 8 (on the Numeric pad)	NS			

GWC-JS Web Client interface: Overview

When you launch a Genero Web Client for JavaScript application, it is displayed in the browser tab within the GWC-JS client interface. The interface consists of the application panel and a navigation side bar. The interface also provides a toolbar with various menu options including a close application option.

- User interface: home page on page 180
- User interface: menus on page 182
- Debug interface on page 183

User interface: home page

The **Genero Web Client** page is displayed when the interface is launched. From this page, you can open an application.

To open the GWC-JS user interface, type the URL into your browser as follows: `http://host:port/gas/gwc-js/index.html`. For example working on the standalone dispatcher as the Web server, enter `http://localhost:6394/gwc-js/index.html`.

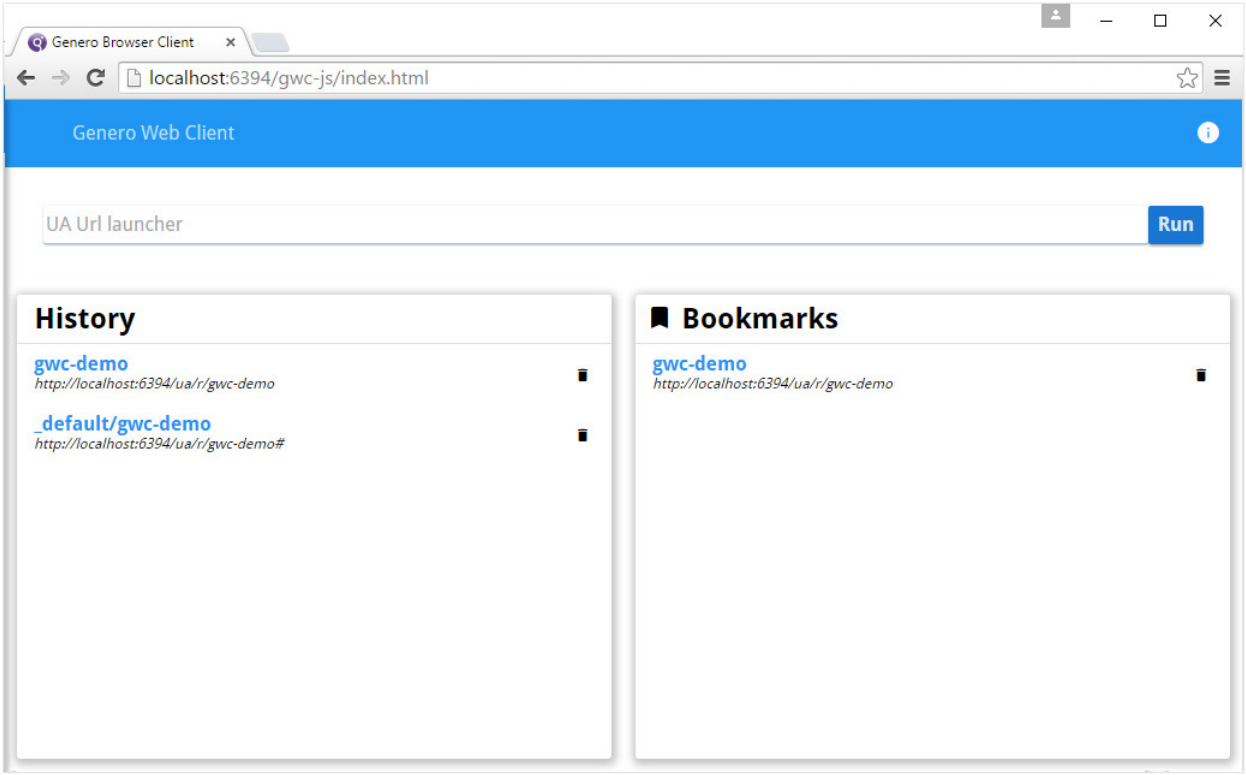


Figure 47: Genero Web Client user interface home page

You can run an application by typing its name (e.g. type `gwc-demo` to open the demo applications) in the **UA Url launcher** field and clicking **Run**.

Or you can open a recent application by selecting it in the **History** panel. If you have bookmarked an application, you can select it to open from the **Bookmarks** panel.

Panels

Panels organize the **Genero Web Client** home page into sections.

History	Open a recently-opened application.
Bookmarks	Open a bookmarked application.

- User interface: navigation on page 181

User interface: navigation
The GWC-JS interface provides a navigation panel or side bar showing the list of running applications.

The **navigation** panel lists all the applications that you have currently open in the **Genero Web Client** user interface. You can switch between your open applications, or windows of an application if more than one, by selecting them from the list to make them active in the main application panel to the right.

> **Note:** Depending on your browser's window size, you may not see the sidebar but just the "hamburger icon"
>
> 
>
> (three horizontal stripes) displayed. When you click on it, a drawer opens giving you access to all your opened applications.

When you close an application, it is removed from the navigation list.

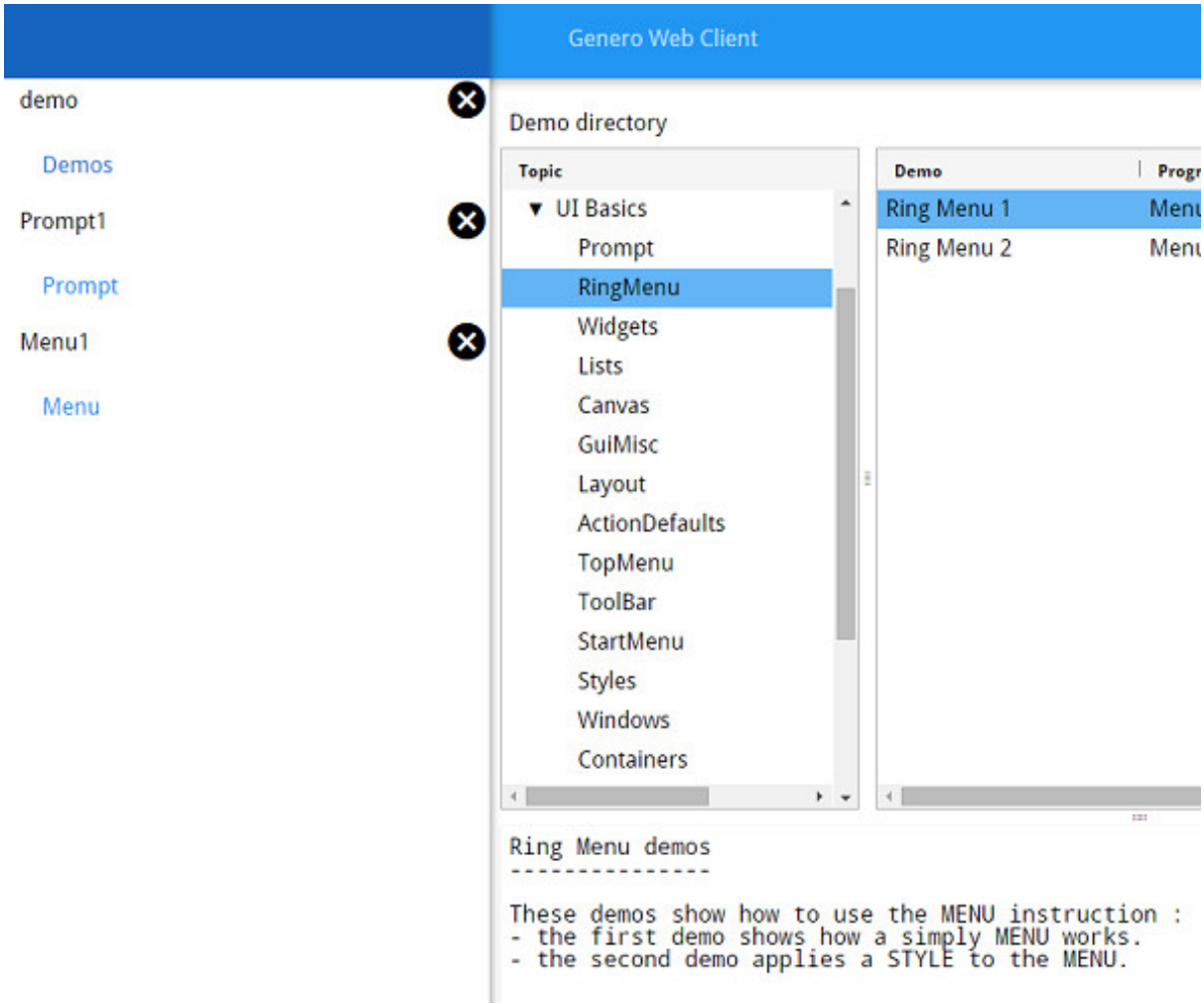


Figure 48: Genero Web Client user interface sidebar navigation

User interface: menus

The GWC-JS interface provides various menu options for working with applications.

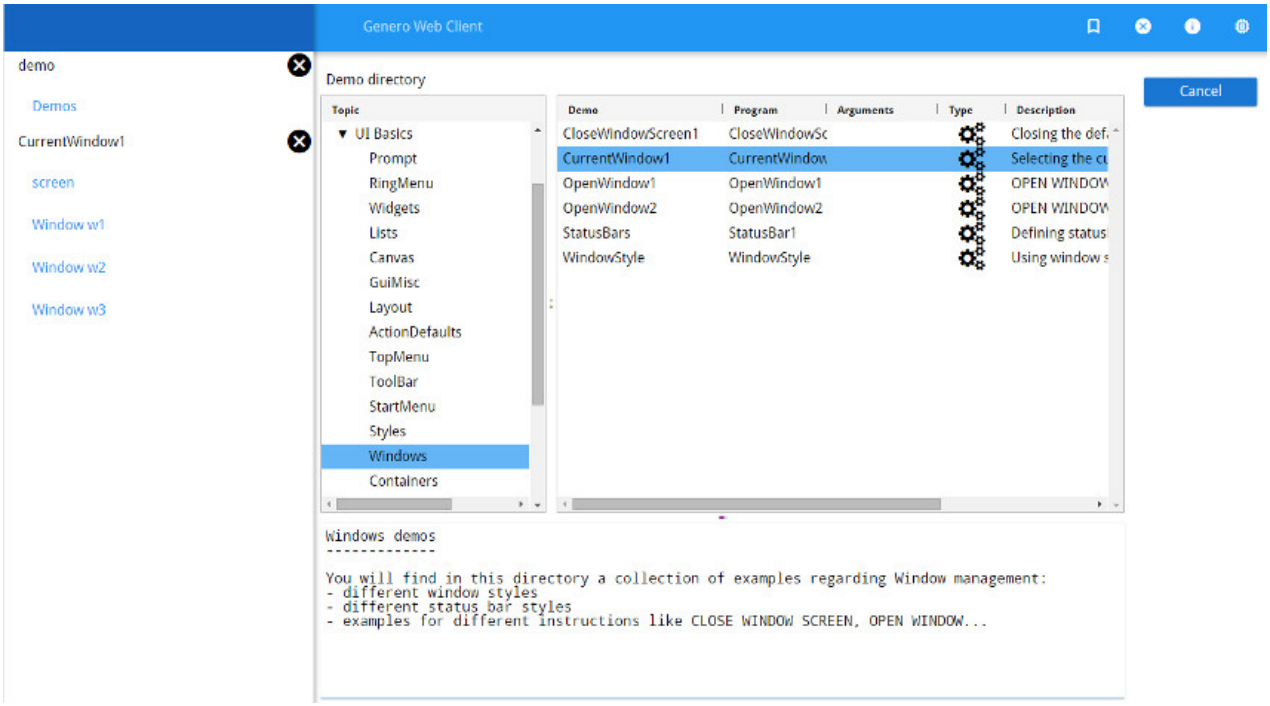


Figure 49: Genero Web Client user interface menus

From the **Genero Web Client** menus at the top of an application window, you can bookmark the application by clicking on the **Bookmark** icon. Selecting an application in the **navigation** panel activates the **Close window** icon for that application. Clicking on the **Close window** icon closes the selected application and returns you to either:

- The next application on the **navigation** list.
- Or, when closing the last application on the **navigation** list, you are returned to **The application ended** page from where you can, for example, view DVM and GWC proxy logs or restart the closed application.

Clicking on the **Application information** menu opens the **Product identification** window, which displays information about the GWC-JS version and the application URL.

Clicking on the **Debug Tools** menu opens the **Debug interface** window, see Debug interface on page 183.

Menus

When an application is selected in the **Genero Web Client** user interface, the following menu items may be shown in the panel at the top of the application page.

Bookmark	Bookmark the current application.
Close window	Close the application window. **Note:** The **Close window** menu may not be shown while an application is expecting user input or other action.
Application information	Open the **Product identification** window.
Debug Tools	Open the **Debug** interface.

Note: If you do not see the **Debug Tools** menu icon, you will need to configure debug mode. For details about launching the dispatcher in this mode, please see Configuring development environment on page 147

Debug interface

To troubleshoot and debug an application, you may need to view the application log files and the AUI tree.

To open the **Debug** interface, click on the **Debug Tools** icon (next to the application information icon) on the right-hand side of your web application window. The **GWC-JS Debug tools** page opens in a new browser tab. You can troubleshoot an application by selecting the nodes of its AUI tree in the panel on the left of the window. Their properties and values will be shown in the panel to the right.

> **Note:** If you do not see the **Debug Tools** menu icon, you will need to configure debug mode. For details about launching the dispatcher in this mode, please see Configuring development environment on page 147

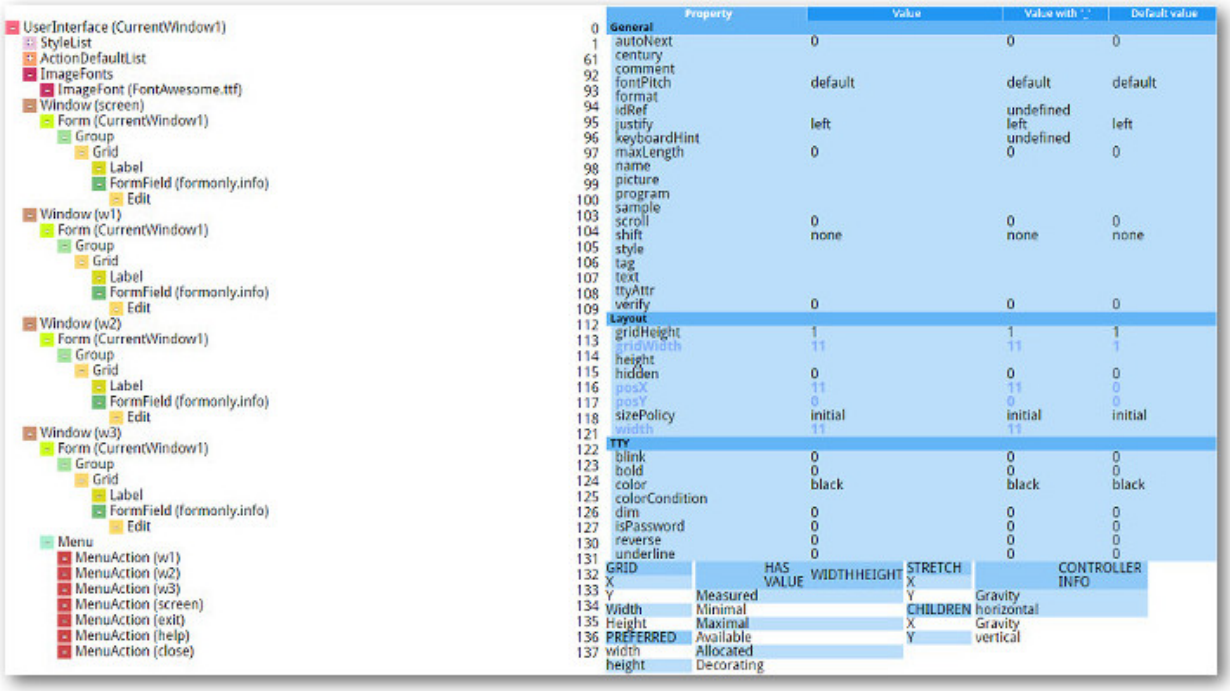


Figure 50: Genero Web Client for JavaScript debug interface

Panels

Panels organize the **Debug interface** into sections.

AUI Tree	Shows the AUI tree of the current application which is open in the GWC-JS user interface browser tab.
AUI node properties	Displays the properties of the selected node in the AUI tree.

Quick Start: Tour of GWC-JS interface

Use this tour to quickly become familiar with the GWC-JS client interface while exploring the demo applications.

- Quick start: stacked windows on page 184

- Quick start: run without waiting on page 185

Quick start: stacked windows
The GWC-JS user interface provides a mechanism for working with applications that use stacked windows.

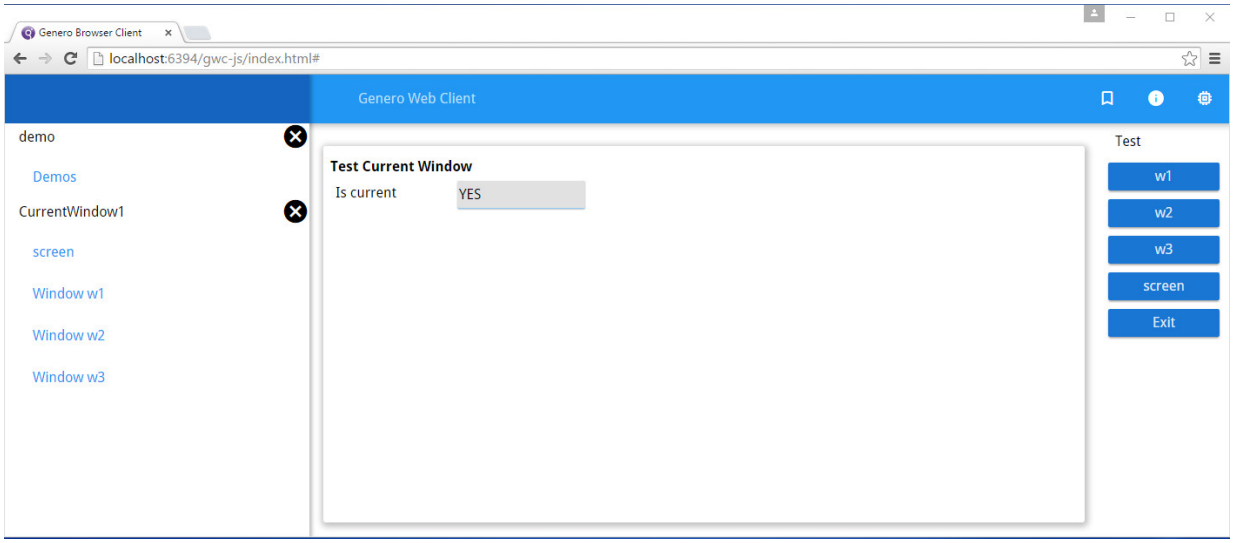
This quick start guide provides you with the steps to explore working with stacked windows in the demos Web application using GWC-JS . Imagine an application's window stack is composed of several windows, W1, W2, ..., Wn. Typically, the GWC-JS only displays the window at the top of the stack - the Wn window; also called the current window. The other windows, W1 to Wn-1, are not displayed but you can access them from the **Navigation** panel.

Before you begin:

Start the standalone dispatcher from the command line using `httpdispatch` and then open the demos application in your browser by entering the URL `http://localhost:6394/gwc-js/index.html?app=gwc-demo`.

1. In the **Topic** tree of the demo directory, navigate to **User Interface >> UI Basics>> Window**. Double-click on the demo **CurrentWindow1** in the panel to the right.

   Figure 51: Application windows in GWC-JS user interface



   - The window at the top of the stack (**Screen**) is displayed.
   - The **Navigation** panel is updated with the **CurrentWindow1** application tree, which has the following links to windows in the stack:
     - **Screen**
     - **Window w1**
     - **Window w2**
     - **Window w3**
2. To set an active window of the **CurrentWindow1** application, in the menu panel called **Test**, click on one of its menu buttons:

   The menu button options are:

   - **w1**
   - **w2**
   - **w3**
   - **Screen**

   The **Is current** field displays YES for the current window.

3. In the **Navigation** panel, click on one of the links in the **CurrentWindow1** tree to set the focus to a window in the stack.

   Note: Although you can switch to windows other than the active or current window in the **CurrentWindow1** stack, their functions are disabled and their menus invisible. In this example, you can only have one current window in the stack.

   The result of your window selection will depend on the following:

   - On the current window, the **Is current** field value is YES and the **Test** menu panel is displayed.
   - On the window not current, the **Is current** field value is NO and the **Test** menu panel is not displayed.

4. To change the current window to another window in the stack:

   In the current window's **Test** menu panel, select one of the other windows by clicking on its menu button. For example, if **Screen** is the current window, click on its **w1** button to make **Window w1** current.

What to do next

When you have completed the above steps, you can explore the mechanism for running Web applications in the background and switching between these using the GWC-JS user interface as described in Quick start: run without waiting on page 185 .

Quick start: run without waiting

The GWC-JS provides a mechanism for applications to be run as child programs that execute in the background, that is the parent program can continue to run without waiting for the child program to finish.

About this task:

This task provides you with the steps to explore working with GWC-JS when an application has sub processes running in the background. The example BDL program (Navigation.42m) starts an application with stacked windows. It gives you the option to use a `RUN WITHOUT WAITING` command to start another instance of the application as a background process.

1. Copy the example BDL program code to a text file and save it as Navigation.4gl

```
# Property of Four Js*
# (c) Copyright Four Js 1995, 2015. All Rights Reserved.
# * Trademark of Four Js Development Tools Europe Ltd
#   in the United States and elsewhere
#
# Four Js and its suppliers do not warrant or guarantee that these
# samples are accurate and suitable for your purposes. Their inclusion is
# purely for information purposes only.

MAIN
  DEFINE cw STRING
  DEFINE i INT

  IF num_args() = 0 THEN LET i=1 ELSE LET i=arg_val(1) END IF

  OPEN FORM f FROM "Navigation"
  DISPLAY FORM f
  DISPLAY "screen" TO fname

  OPEN WINDOW w1 WITH FORM "Navigation"
  CALL fgl_setTitle("Window w1 - " || i)
  DISPLAY "w1" TO fname

  OPEN WINDOW w2 WITH FORM "Navigation"
  CALL fgl_setTitle("Window w2 - " || i)
  DISPLAY "w2" TO fname
```

```
  OPEN WINDOW w3 WITH FORM "Navigation"
  CALL fgl_setTitle("Window w3 - " || i)
  DISPLAY "w3" TO fname

  WHILE 1
    DISPLAY "YES" TO info
    MENU "Test"
      COMMAND "w1"      LET cw="w1" EXIT MENU
      COMMAND "w2"      LET cw="w2" EXIT MENU
      COMMAND "w3"      LET cw="w3" EXIT MENU
      COMMAND "screen"  LET cw="screen" EXIT MENU
      COMMAND "RWW"     RUN "fglrun Navigation " || i+1  WITHOUT WAITING
      COMMAND KEY(INTERRUPT) "Exit" EXIT PROGRAM
    END MENU
    DISPLAY "NO" TO info
    CASE cw
      WHEN "w1" CURRENT WINDOW IS w1
      WHEN "w2" CURRENT WINDOW IS w2
      WHEN "w3" CURRENT WINDOW IS w3
      OTHERWISE CURRENT WINDOW IS SCREEN
    END CASE
    DISPLAY "YES" TO info
  END WHILE
END MAIN
```

The program's main code block opens four windows and displays form menus.

> **Note:** The "RWW" menu option with the following command:
>
> `COMMAND "RWW" RUN "fglrun Navigation " || i+1 WITHOUT WAITING`
>
> contains the instruction which allows you to start another instance of the application.

2. Copy the program's form specification code to a text file and save it as Navigation.per

```
# Property of Four Js*
# (c) Copyright Four Js 1995, 2015. All Rights Reserved.
# * Trademark of Four Js Development Tools Europe Ltd
#   in the United States and elsewhere
#
# Four Js and its suppliers do not warrant or guarantee that these
# samples are accurate and suitable for your purposes. Their inclusion is
# purely for information purposes only.

LAYOUT
GROUP(text="Test Current Window")
GRID
{
Window name [fw         ]
Is current  [info       ]
}
END
END
ATTRIBUTES
  LABEL fw=formonly.fname;
  EDIT info=formonly.info;
END
```

3. At the command line, compile the source code modules you have created (Navigation.4gl and Navigation.per) by using the `fglcomp` and `fglform` tools, e.g.

   Run the following commands:

   ```
   fglcomp Navigation.4gl
   ```

```
fglform Navigation.per
```

Compiled files are created: `Navigation.42f`, is created for the form and `Navigation.42m` for the source code module.

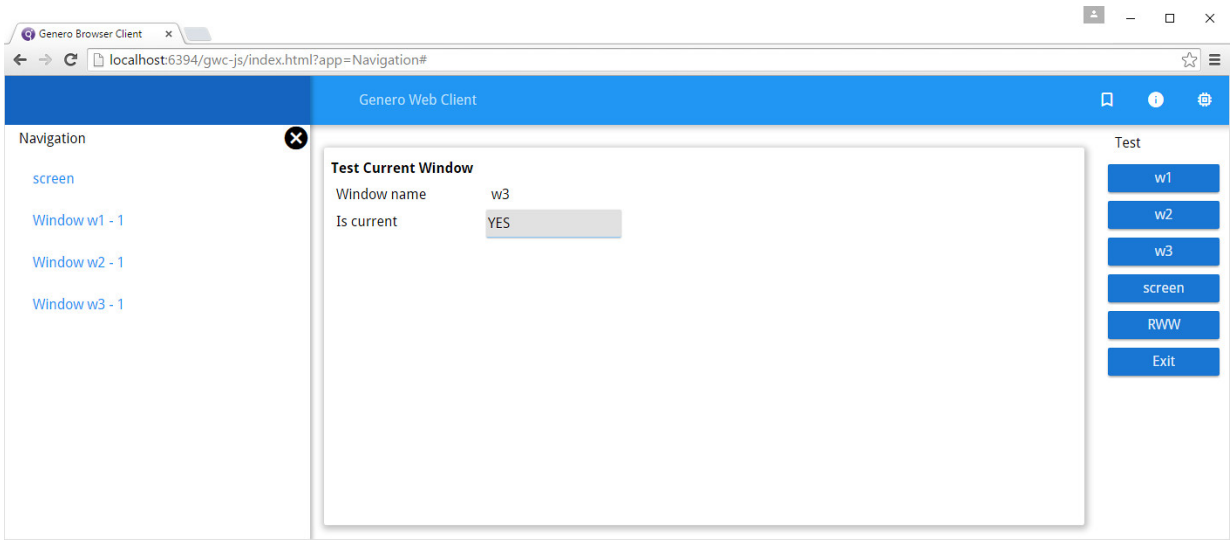
4. Create an application configuration file for your **Navigation** application. Provide an absolute path to the location of your compiled application file in the `PATH` element, and in the `MODULE` element specify the `Navigation.42m` module required to launch your application.

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/
gas/3.00/cfextwa.xsd">
 <EXECUTION>
      <PATH><path_to_your_local_directory></PATH>
      <MODULE>Navigation.42m</MODULE>
 </EXECUTION>
 <UA_OUTPUT>
    <GWC-JS>gwc-js</GWC-JS>
 </UA_OUTPUT>
</APPLICATION>
```

5. Save the configuration file (e.g. Navigation.xcf) in your `$(res.appdata.path)/app` directory.
6. Start the GAS standalone dispatcher from the command line using `httpdispatch`
7. Open the Navigation application in your browser by entering the URL `http://localhost:6394/ua/r/Navigation`.

Figure 52: Application's current window in GWC-JS user interface



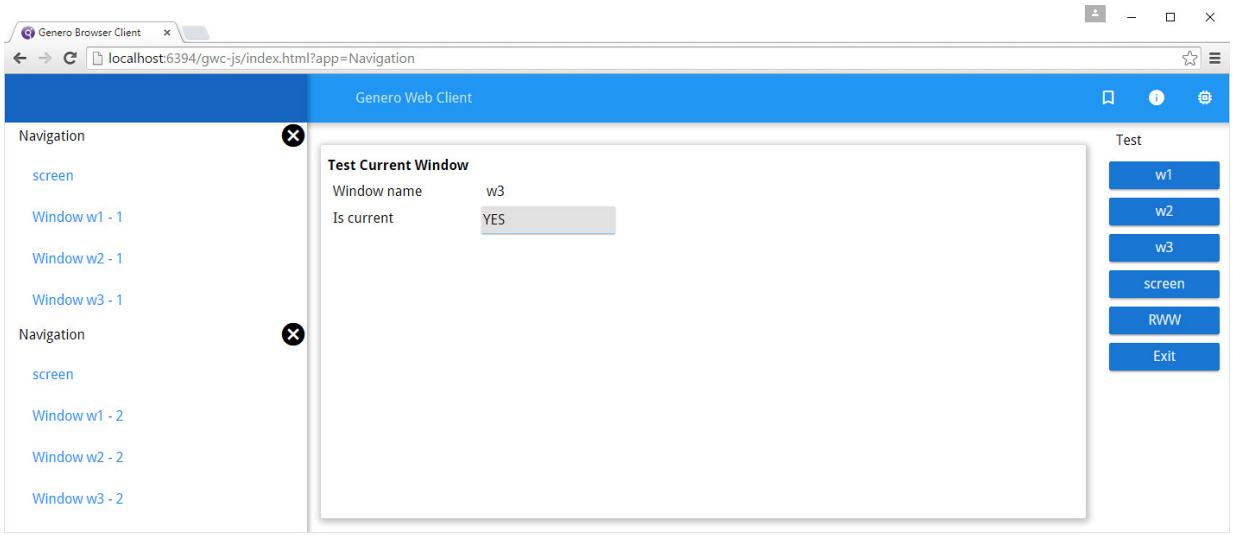
- The current window at the top of the stack (**W3**) is displayed in the GWC-JS user interface. The current window's menu button options are:
  - **w1**
  - **w2**
  - **w3**
  - **Screen**
  - **RWW**

    - **Exit**
  - The **Window Name** field displays **W3** for the current window.
  - The **Is current** field displays YES for the current window.
  - The **Navigation** panel or side bar is updated with the **Navigation** application tree, which has the following links to windows in the stack:
    - **Screen**
    - **Window w1 - 1**
    - **Window w2 - 1**
    - **Window w3 - 1**
8. To have another instance of the Navigation application running at the same time:

   In the current window's **Test** menu panel, select the **RWW** menu button.

   The **Navigation** panel or side bar is updated with another **Navigation** application tree.

   Figure 53: Two instances of an application in GWC-JS interface



9. From the **Navigation** panel or side bar, select the current windows in each running instance of the Navigation application, for example complete the following steps:

   a) Select Window w3 - 1

   b) Select Window w3 - 2

   You should see that both applications have current windows. The first instance of the Navigation application you started (the parent program) continues to run without waiting for the second instance of Navigation (the child program) to finish. The result of your selection will depend on the following:

   - On the current window, the **Is current** field value is YES and the **Test** menu panel is displayed.
   - On the window not current, the **Is current** field value is NO and the **Test** menu panel is not displayed.

Starting GWC-JS applications

As GWC-JS type applications start, a bootstrap mechanism is called to initialize information.

To start a GWC-JS type application, type the URL of your application into your browser as follows: `http://host:port/gas/ua/r/group/myapp`

As the application starts, a `bootstrap.html` file is called. This bootstrap page loads the required JavaScript files and provides information for rendering the application, such as the following:

- The application style sheet.

- The application icon.
- The JavaScript injectors.

The bootstrap is configured in the `GWC-JS` element of the `UA_OUTPUT` configuration element, see UA_OUTPUT on page 354.

The `bootstrap.html` file, including all the resources required for rendering the application, are located in the directory referenced by the `GWC-JS` element. The default directory is `gwc-js` located in `$FGLASDIR/web/`.

GWC-JS applications and use of cookies

This topic provides information about the use of *cookies* in GWC-JS Web applications.

Using cookies in GWS-JS Web applications

Currently GWC-JS Web applications do not use cookies. Nevertheless, you can use cookies in your applications if you decide to save or exchange some data that you want to persist between different runs of an application.

For example, Web applications that run on the same domain as your GAS and whose GWC cookie is set on the root path (`/`) of the URL are able to retrieve cookies in future requests for an application. When a request is made, the cookie is sent back from browsers to the GAS where it is recognized.

> **Note:** Even though GWC-JS Web applications do not use cookies, user agent browsers generally need to accept cookies because browsers themselves may need cookies turned on. With Firefox, for example, cookies and local storage are tied together. As GWC-JS uses local storage, if you do not have cookies turned on in your Firefox browser, you may get a blank page.

Customization for GWC-JS applications

This section provides you with topics about how to customize the GWC-JS user interface. It provides you with a customization overview and some procedures to get started with customizing the theme and adding headers and footers.

> **Note:** As development work on the GWC-JS continues, we will update this section with more customization options as new versions of the GWC-JS are released.

Customization framework

The GWC-JS framework consists of a set of JavaScript (js), template HTML (.tpl.html) and sass stylesheets (.scss) files that implement the various the components of the front-end. Customization consists of adapting these files that define the components in your project and implementing the change with the `grunt` tool to build the front end.

The customization framework overview

The `GWC-JS` infrastructure consists of two components:

Project directory	This directory contains the customizable JavaScript (js), template HTML (.tpl.html) and sass stylesheets (.scss) files used to create the front-end client

application, for more information see Project directory on page 192 page.

Tools

To develop and compile GWC-JS, you will need some third-party tools installed in your system: These tools should be downloaded by the developer working on the customization.

> **Note:** The `Node.js` download contains `npm`. Running `npm` install commands for `Bower` and `Grunt` completes the tools installation process, see Configuring your Environment on page 198

Tool	Description	Project directories/ files
`Node.js` (http://nodejs.org/)	Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes your GWC-JS Web applications lightweight and efficient.	`\customization\js`
`git` (http://git-scm.com/)	Git looks after the version control for the project development, i.e. keeping track of the changes you make to your project.	n/a
`npm`	Node Package Manager (npm) keeps track of your project's packages and their dependencies in a manifest file.	package.json
`Bower`	Bower is a package manager for the web. It	bower.json

Tool	**Description**	**Project directories/ files**
	downloads and installs packages the GWC-JS requires. Bower keeps track of these packages in a manifest file.	
`Grunt`	Grunt is the JavaScript task runner which builds the project with compilation options and automates the compilation and the reconfiguration of the GWC-JS client after customization.	Gruntfile.js
Sass	Sass is an extension of CSS that allows use of variables, mixins, and inline imports to implement the various components of the GWC-JS. The project directory contains a set of sass stylesheets that provide default definitions for all key HTML components and Genero application elements; such as forms, buttons, navigation and other interface components.	`\customization` `\sass`

Tool	Description	Project directories/ files
	By editing these styles in the `scss` files, you customize the components in your project. **Note:** The open source text editor Brackets (http://brackets.io/) makes working with `scss` and `json` files much easier.	

Project directory

The *project* directory contains the files needed to create your own customization for the look and feel of your GWC-JS front end Web client.

Sources are provided so you can customize the front end Web client to comply with your company's graphical and visual styles and theme layout.

Before you can customize your front end Web client, you need to unzip the project zip file to a directory so that you can modify the required files, see Configuring your Environment on page 198. A project directory is created and amongst its directories and files you will find the following:

project_dir	The GWC-JS project root directory, the default location is in your *`$FGLASDIR`*`/tpl` directory.
customization	The customization directory is dedicated to your project's customization. It contains a `default` directory, which provides you with sample customized files in its sub directories. It is recommended that you add within the `customization` directory, a subdirectory for each customization that you want to implement. You can have as many customizations as you wish while allowing you to activate them as required, see Activating Customization with custom.json on page 206.

To create your own customization, add a directory, for example, `myCustomizedApp`, and within it add sub directories as described in the table Your Customization Directory:

Table 18: Your Customization Directory

Directory	Description	Examples
`img`	For your project's logo and image files that customize the front-end look and feel.	n/a
`js`	For your project's JavaScript and widget template HTML files.	• See How GWC-JS works on page 175 • See Adding Header and Footer text on page 202
`locales`	For your project's translation texts for localization.	See Adding Localized Texts on page 205
`sass`	Cascading Style Sheet files for instructing the browser how to display specific widgets. You can change colors, font, layout, and how text is positioned, etc.	See Widget scss file on page 197
`theme.scss.json`	Change the default GWC-JS color and theme settings for your windows, widgets, messages, buttons, tables, etc.	See Customizing the theme on page 200

src

The *`project_dir`*`/src` directory contains the JavaScript, CSS, locale, and all the core source files that make up the default Web client. It is recommended that you examine the source files to familiarize yourself with how the front-end is implement. When you customize your Web client, you can extend the standard widget classes you need in new widget `js` files in your customized directory.

Caution: We recommend that you leave the core sources untouched.

dist

It contains the `web` subdirectory where the outputted/generated files, the result of your customization, are combined in a single file, or in as few files as possible, to reduce network traffic and for the benefit of the browser cache.

Make your custom GWC-JS alias/link in ($FGLASDIR/web) point to this "web" directory or you can configure the `GWC_JS_LOOKUP_PATH` `on page 320` element with the path to your *`project_dir`*`/dist`, see Configuring Applications for Custom GWC-JS on page 206.

- Configuring your Environment on page 198
- Customizing the theme on page 200

Understanding GWC-JS widgets

An understanding of how GWC-JS widgets works can aide you in your customization efforts.

The GWC-JS relies on a number of basic/built-in widgets that define the user interface: such as application, application host, layout grids, and widgets for form objects, and so on. Each widget is defined with the following:

- A JavaScript (js) file to manage the widget's creation, behavior, and listeners
- A template HTML (.tpl.html) file which contains a minimal HTML code structure of the widget
- A sass (.scss) file which defines the widget style

It is recommended that you examine the core source widget files that make up the default Web client in your *`project_dir`*`/src` directory to familiarize yourself with how the front-end is implement. The topics in this section will help you understand what widgets you need to extend to change the look and feel of your application.

This section explains how you can create your own customized widgets by extending the built-in widgets in your project's GWC-JS front-end web client.

- Widget template file on page 194
- Widget JavaScript file on page 195
- Widget scss file on page 197

Widget template file

The GWC-JS widget templates provides you with the basis for creating and customizing widgets.

The GWC-JS contains blocks of HTML code that can be easily selected by CSS. GWC-JS widget templates define two main types of HTML elements:

- Simple widget
- Container widget

Simple widget

Simple widget defines elements that implement basic functions such as form field elements. The `EditWidget`, `LabelWidget`, `TextEditWidget`, etc. are good examples where the * * container holds code for a single widget. For example the *`project_dir`*`/src/js/base/widget/widgets/formfields/EditWidget.tpl.html` template is shown in the sample.

```
<div class="mt-field gbc_dataContentPlaceholder">
  <input type="text" />
</div>
```

The CSS selector targets elements of both the `mt-field` and `gbc_dataContentPlaceholder` CSS classes. When a widget is used in the DOM, elements and attributes of these classes will be appended in the DOM node. The *input* element with type "text" represents markup that will be placed inside the DOM node.

Container widget
The container widget provides a placeholder for child widgets. It is defined with a class attribute with its value set to *containerElement*. The *`project_dir`*`/src/js/base/widget/widgets/actions/MenuWidget.tpl.html` template shown is a good example of this type of template.

```
<div>
  <div class="gbc_MenuWidgetTitle">
    <div class="gbc_MenuWidgetText"></div>
  </div>
  <div class="gbc_MenuWidgetScrollContainer">
    <div class="containerElement"></div>
  </div>
</div>
```

Widget JavaScript file
The GWC-JS widget `js` file contains the JavaScript code of the widget.

The GWC-JS stores the JavaScript code for widgets in a separate `js` file. The `js` file contains functions for rendering the widget template file and dynamically building the DOM element. If you wish to customize a widget, you only need to extend the standard widget class in a new widget `js` file in your project directory and make any changes required to the HTML structure in a new widget template file (`.tpl.html`). The sample code shows the minimal code you need to extend a basic widget in `js`.

Extending a basic widget

```
"use strict";

modulum('newWidget', ['baseWidget', 'WidgetFactory'],
  function(context, cls) {

    cls.newWidget = context.oo.Class(cls.baseWidget, function($super) {
      return {
        __name: "newWidget",

        /* your custom code */
      };
    });
    cls.WidgetFactory.register('widgetType', ‘widgetStyle’, cls.newWidget);
  });
```

Where:

- *newWidget* is your custom widget
- *baseWidget* is the built-in widget that it extends

The *widgetStyle* is registered as the new widget so that you can reference it in your Genero BDL application code:

```
EDIT f01 = formonly.edit, STYLE=”widgetStyle”;
```

Then only fields with the new style *widgetStyle* applied will be customized.

Example customizing a simple edit widget

For simple widget customization, see the example in *project_dir*/customization/default/js/MyEditWidget.js:

```
"use strict";

modulum('MyEditWidget', ['EditWidget', 'WidgetFactory'],

  function(context, cls) {

    /**
     * Edit widget.
     * @class classes.MyEditWidget
     * @extends classes.EditWidget
     */
    cls.MyEditWidget = context.oo.Class(cls.EditWidget, function($super) {
      /** @lends classes.MyEditWidget.prototype */
      return {
        __name: "MyEditWidget".

        setTitle: function(title) {
          $(this.getElement()).find(".title").text(title);
        },

        getTitle: function() {
          return $(this.getElement()).find(".title").text();
        }
      };
    });
    cls.WidgetFactory.register('Edit', cls.MyEditWidget);
  });
```

In this case, the customization extends the built-in *EditWidget*. You can refer to an element in the HTML template for the widget (*project_dir*/customization/default/js/MyEditWidget.tpl.html) by using this.getElement() function.

Example extending a container widget

For container widget customization, see the example in *project_dir*/customization/default/js/MyApplicationHostMenuWidget.js

```
"use strict";

modulum('MyApplicationHostMenuWidget', ['WidgetGroupBase', 'WidgetFactory'],

  function(context, cls) {

    cls.MyApplicationHostMenuWidget = context.oo.Class(cls.WidgetGroupBase,
 function($super) {
      /** @lends classes.MyApplicationHostMenuWidget.prototype */
      return {
        __name: "MyApplicationHostMenuWidget",

        _windowIconImage: null,
        _titleElement: null,
        _defaultTitle: "Customized Title Bar",

        _aboutMenu: null,
        _debugMenu: null,

        constructor: function() {
```

```
          $super.constructor.call(this);
          this._aboutMenu =
 cls.WidgetFactory.create('ApplicationHostAboutMenu');
          this._debugMenu =
 cls.WidgetFactory.create('ApplicationHostDebugMenu');
          this.addChildWidget(this._aboutMenu);
          this.addChildWidget(this._debugMenu);
        },

        [...]
        destroy: function() {
          this.removeChildWidget(this._aboutMenu);
          this.removeChildWidget(this._debugMenu);
          this._aboutMenu.destroy();
          this._aboutMenu = null;
          this._debugMenu.destroy();
          this._debugMenu = null;
          $super.destroy();
        },
        [...]
      };
    });
    cls.WidgetFactory.register('ApplicationHostMenu',
 cls.MyApplicationHostMenuWidget);
  });
```

For a Container widget the base widget is always *WidgetGroupBase*. You then add the child elements with `this.addChildWidget` or remove them with `this.removeChildWidget`.

Widget scss file

The GWC-JS widget `scss` file contains the widget styles.

The GWC-JS stores the style for widgets in a separate `scss` file. The `scss` file contains the colors, fonts and variables that define the style. When the sass file is processed, it takes the variables you define for colors, for example *$gbc-primary-medium-color*, and outputs normal CSS with our variable values placed in the CSS. The sample code shows the *`project_dir`*`/customization/default/sass/MyEditWidget.scss`:

Extending a basic widget style

```
.gbc_MyEditWidget {
  > .title {
    padding: 0 5px;
    font-size: 8pt;
    color: $gbc-primary-medium-color;
    text-transform: none;
  }
  > input {
    flex: 1 1 0;
    line-height: 32px;
    border-bottom: solid 2px $gbc-primary-light-color;
  }
  &.gbc_Focus > input {
    border-bottom: solid 2px $gbc-primary-medium-color;
  }
}
```

A widget with a style in the form has a dedicated selector:

For example, in the Genero BDL application code for your form (`per`) file you have:

```
EDIT f01 = formonly.edt, STYLE="mystyle1 mystyle2 …";
```

You get a generated selector in the HTML structure:

class="… *gbc_style_mystyle1 gbc_style_mystyle2* …"

Configuring your Environment
This topic provides information about how to configure your environment to customize the Genero Web Client for JavaScript (GWC-JS) front-end for your application.

About this task

The following procedure guides you through the process of configuring an environment for customizing a GWC-JS application.

Before you begin

To develop and compile GWC-JS, you will need to install the following tools, which can be downloaded from the URLs provided:

- node.js: https://nodejs.org/
- git: http://git-scm.com/download/

> **Note:** It is recommended that you download the stable version (v4.2.2) of node.js. To check you installation version, type the command: `node --version` at the command line.

1. Navigate to your *$FGLASDIR*`/tpl` directory and unzip the following zip file `fjs-`*`gwc-js-xxx`*`-build`*`123456789`*`-project.zip` to, for example, your *$FGLASDIR*`/tpl/` directory.

   > **Note:** You can unzip the project files anywhere you wish, however it is not recommended to put them in the *$FGLASDIR*`/web/` directory as this is the public directory where your sources may be exposed.

   - *gwc-js-xxx* is the version number, for example *gwc-js-1.00.02*
   - *123456789* is the build number, for example *201504141750*.

   A directory is created in the extracted location using the version number as the title in the format of *gwc-js-xxx*. This is your *project_dir*, see Project directory on page 192.
2. Navigate to your *`project_dir`* directory and open the `readme.md` file using a text editor.
3. To install all necessary tools, complete the following sub-steps, which can also be found in the `readme.md` file.
   a) Open the **Node.js command prompt** window at administrator level on Windows™, or as the user with privileges to install in Linux® or MAC® OS®.
   b) Navigate to your *`project_dir`* and enter the following Node Package Manager (npm) commands:
      - `npm install`
      - `npm install -g grunt-cli`
      - `npm install -g bower`
      - `grunt deps`

      Note:

      - The two `npm install` commands with the global option (`-g`) are required only the first time you install node.js on the machine. You can omit running them if you already have node.js installed and just run `npm install` and `grunt deps`.
      - For installation on Unix, you can use the Node Version Manager (nvm) tool nvm tool to install node.js. It allows you to switch between different versions of node.js. To install and use, for example, node.js 4.2.2, run these commands:
        - `nvm install 4.2.2`
        - `nvm use 4.2.2`

Node.js is now installed for your project and npm and bower dependency packages have been updated by `grunt`. If you encounter any problems during the installation, see Troubleshooting on page 199.

4. To build the project with default compilation options, you must run `grunt`
   When you have completed the above step, you should see that a `dist` directory has been created in *project_dir*. Inside the `dist/web` directory there is now a compiled version of the GWC-JS front-end.
5. Now that your *`project_dir`*`/dist/web` directory is created, you can use your custom GWC-JS directly in your local GAS server by creating a symbolic link (*link_name*) to this directory in your *`$FGLASDIR`*`/web` directory. To create a link, navigate to *`$FGLASDIR`*`/web/`, and choose from the following options:

   - On Windows®, run the command `mklink /D` *link_name* *`project_dir`*`/dist/web`.
   - On Linux®/UNIX™, run the command `ln -s` *`project_dir`*`/dist/web` *link_name*

     Tip: Alternatively, you can configure the `GWC_JS_LOOKUP_PATH` element to specify the path to your custom GWC-JS front-end client, see GWC_JS_LOOKUP_PATH on page 320.

What to do next

When you have completed configuring your environment in the above steps, make sure your GWC-JS Web client is working as expected by opening an application in your browser. You can do this by configuring an application for your project, see Configuring demo application for custom GWC-JS on page 207.

If your GWC-JS Web client is working, your next task is to activate customization in your *`project_dir`*`/custom.json`. This is detailed in Activating Customization with custom.json on page 206.

Troubleshooting

Unable to connect to github.com

Git is required to fetch bower components. When doing a `grunt deps`, if you get this error:

```
fatal: unable to connect to github.com: github.com
[0: 192.30.252.129]: errno=Connection refused Warning: Task "bower-install-
simple:default" failed.
Use --force to continue.
```

This may mean that you may have a proxy that is preventing the retrieving of `bower` components. A solution is to perform the procedure described in the following steps:

1. Configure `git` by typing the following at the command line

   ```
   git config --global url.https://github.com/.insteadOf git://github.com/
   ```

2. Clean your installation by choosing from the following options:

   ```
   grunt clean
   ```

   or

   ```
   npm cache clean
   ```

   Or, you can remove the *project_dir*`/node_modules` directory.
3. Redo the node components installation as described in steps in Install Components

   For more information and support contact your local Four Js support center.

Customizing the theme

The simplest way to customize the look and feel of your GWC-JS front end Web client is to change styles in the *theme.scss.json* file.

theme.scss.json

This topic describes how you can modify the default GWC-JS theme to customize your Web client. Understanding this customization requires some knowledge of JSON file syntax.

The `theme.scss.json` file in your *`project_dir`*`/customization/mycustomization` directory allows you to edit and change the main GWC-JS theme settings for windows, widgets, messages, buttons, tables, etc. For example, you can set the UI's primary and secondary colors, background colors, border styles, window size for sidebar size, etc., as required with theme settings.

> **Note:** You are not modifying the theme installed with your product, you are modifying the theme for your project. This means that you will not lose your work when you upgrade the product.

Syntax

The syntax of each theme setting consists of a `variable_name`, followed by a colon (:) and then the `variable_value`. Variable names and values are in double quotes. Theme styles are separated by commas.

```
"variable_name"   :   "variable_value",
```

- The *variable_name* is a unique identifier used throughout the GWC-JS project to set theme styles, e.g. the UI window's primary color is referenced by the variable *gbc-primary-background-color*. These variables are standard theme variables and should not be subject to change.
- The *variable_value* can be a color, a number or a variable, depending on the style. If the value is defined by a variable, for example *$vname*, “vname” is an existing variable name in the GWC-JS project.

Color variables naming convention

Variables that define colors follow this naming convention:

```
$mt-colorname[-intensity]
```

- *colorname* is the name of the color as defined in the material color palette.
- *intensity* (optional) is the number for the gradient, for example: "*$mt-grey-200*", defines an intensity of light grey. For more information on material color, go to http://www.google.com/design and navigate to the "Color palette" page.

Example of customized theme styles

The `theme.scss.json` file contains a typical list of styles you can customize as shown in the example. See the table Table 19: theme.scss.json on page 201 for a description of the variables and values.

```
{
  "gbc-primary-background-color"          : "$mt-grey-200",
  "gbc-secondary-background-color"        : "$mt-white",
  "gbc-field-background-color"            : "$mt-white",
  "gbc-field-disabled-background"         : "rgba(0,0,0,0.04)",
  "gbc-primary-color"                     : "$mt-green-700",
  "gbc-primary-medium-color"              : "$mt-green-500",
  "gbc-primary-light-color"               : "$mt-green-100",
  "gbc-secondary-color"                   : "$mt-grey-600",
  "gbc-disabled-color"                    : "$mt-grey-400",
  "gbc-separator-color"                   : "$mt-grey-400",
  "gbc-header-color"                      : "$mt-grey-100",
  "gbc-message-color"                     : "$mt-grey-800",
```

```
  "gbc-error-color"                       : "$mt-red-800",
  "gbc-sidebar-always-visible-min-width" : "999999px",
  "gbc-sidebar-default-width"             : "350px",
  "gbc-animation-duration"                : "0.200s"
}
```

Table 19: `theme.scss.json`

Variable name	Variable Value	Description
gbc-primary-background-color	$mt-grey-200	Window primary color
gbc-secondary-background-color	$mt-white	Window/containers (group, drop-down menu) secondary color
gbc-field-background-color	$mt-white	Widget's background (bg) color
gbc-field-disabled-background	rgba(0,0,0,0.04)	Widget's disabled background color grey specified with an opacity value (.04) using the alpha property with rgba.
gbc-primary-color	$mt-green-700	Button/widget's color gradient for dark green
gbc-primary-medium-color	$mt-green-500	Button/widget's color gradient for normal intensity green
gbc-primary-light-color	$mt-green-100	Button/widget's color gradient for light green
gbc-secondary-color	$mt-grey-600	Widget's (RADIOGROUP, CHECKBOX) border color
gbc-disabled-color	$mt-grey-400	Widget's disabled color grey
gbc-separator-color	$mt-grey-400	Toolbar separator color grey
gbc-header-color	$mt-grey-100	Table header background color light grey
gbc-message-color	$mt-grey-800	Message background color dark grey
gbc-error-color	$mt-red-800	Error message background color deep red
gbc-sidebar-always-visible-min-width	999999px	Window size (in pixels) for sidebar to appear. This settings hides the sidebar.
gbc-animation-duration	0.200s	Animation duration (in seconds), e.g. Speed of product identification window pop-down appearance.

The `UA_OUTPUT on page 354` configuration element of your application will need to specify the customization project directory you used to provide the application look-and-feel.

- Configuring your Environment on page 198

Adding Header and Footer text

The topic provides information on how to add text to the header and footer of the GWC-JS user interface.

About this task:

The GWC-JS provides place holders that state "Place your header here" and "Place your footer here" in a customized widget for the main application container widget. To add header and footer text to your GWC-JS project user interface, modify the place holders in the `MyMainContainerWidget.tpl.html` file.

Before you begin:

It is assumed you have created a project directory and your *`project_dir`*`/customization/default/js` directory contains the widget template file for the main container, `MyMainContainerWidget.tpl.html`.

1. To customize the header and footer, navigate to the *`project_dir`*`/customization/default/js` directory and open the `MyMainContainerWidget.tpl.html` file with a text editor.
2. To change the header and footer text, replace the place holder text between the header and footer tags with your text and save your changes:

```
<div>
  <header>
        Place your header here
  </header>
  <!-- GWC-JS will be rendered in this div -->
  <div class="containerElement"></div>
  <footer>
        Place your footer here
  </footer>
</div>
```

3. To compile the changes, in the **Node.js command prompt** window, navigate to *project_dir* and run `grunt` or alternatively you can run `grunt dev` to continuously build the project.

What to do next

View the changes by opening the GAS demos application in your browser. You can do this by configuring the demos application for your project, see Configuring demo application for custom GWC-JS on page 207.

The GWC-JS header and footer text is now changed.

Creating your own widgets

The ModelHelper class in the GWC-JS project provides APIs to help you customize widgets.

The GWC-JS two main widgets types:

- Simple widget, (for an example of creating and customizing, see Simple widget on page 194)
- Container widget (see Container widget on page 195)

The `ModelHelper` class in the GWC-JS project provides APIs to help customization. It is available at *`project_dir`*`/src/js/base/helpers/ModelHelper.js`.

The table shows a list of the `ModelHelper` APIs and describes their function.

Table 20: ModelHelper APIs

Function	Description
`addNewApplicationListener(fct)` returns `unfct`	`fct` is the function called when a new application is started. `unfct` is the function to call to unregister the listener

Function	Description
`addCloseApplicationListener(fct)` returns `unfct`	`fct` is the function called when a new application is started. `unfct` is the function to call to unregister the listener
`addCurrentWindowChangeListener(fct)` returns `unfct`	`fct` is the function called when the current window changes. `unfct` is the function to call to unregister the listener
`addAuiUpdateListener(fct)` returns `unfunct`	`fct` is the function called when any DVM response is received . Be aware that this mechanism is global to all applications. On heavy updates this can slow down the application. `unfct` is the function to call to unregister the listener
`getCurrentApplication` returns `win`	`win` is the current application object or null if the application cannot be found
`getApplication` returns `app`	`app` is the application to which the widget belongs
`getNode(`*idref*`)` returns `node`	`idref` is the AUI tree id reference `node` is the related widget node or null if the node is not found
`getUserInterfaceNode` returns `node`	`node` is the widget representing the User Interface node in the AUI tree or null if not found
`getAnchorNode` returns `node`	`node` is the node holding the representation of value in the AUI tree
`getFieldNode` returns `node`	Applies to FormField, Matrix or TableColumn `node` is the field node corresponding to the widget or null if it does not apply
`getDecorationNode` returns `node`	Applies to FormField, Matrix or TableColumn `node` is the node holding the visual information (Edit, CheckBox, ComboBox, etc...) or null if not found

Extending a basic widget using ModelHelper

The sample code creates a basic widget for the header bar using some functions in the `ModelHelper` class. The code in the example is found in *`project_dir`*`/customization/default/js/MyHeaderBarWidget.js`

```
"use strict";

modulum('MyHeaderBarWidget', ['WidgetBase', 'WidgetFactory'],
```

```
  /**
   * @param {gbc} context
   * @param {classes} cls
   */
  function(context, cls) {

    /**
     * @class classes.MyHeaderBarWidget
     * @extends classes.WidgetBase
     */
    cls.MyHeaderBarWidget = context.oo.Class(cls.WidgetBase,
 function($super) {
      /** @lends classes.MyHeaderBarWidget.prototype */
      return {
        __name: "MyHeaderBarWidget",
        /** @type {classes.ModelHelper} */
        _model: null,
        /** @type {number} */
        _appsCount: null,

        constructor: function() {
          $super.constructor.call(this);
          this._appsCount = 0;
          this._model = new cls.ModelHelper(this);

 this._model.addNewApplicationListener(this.onNewApplication.bind(this));

 this._model.addCloseApplicationListener(this.onCloseApplication.bind(this));

 this._model.addCurrentWindowChangeListener(this.onCurrentWindowChanged.bind(this));
        },

        onNewApplication: function(application) {
          ++this._appsCount;
          var elt = this.getElement().querySelector(".MyHeaderBarWidget-
counter");
          elt.textContent = this._appsCount.toString();
        },

        onCloseApplication: function(application) {
          --this._appsCount;
          var elt = this.getElement().querySelector(".MyHeaderBarWidget-
counter");
          elt.textContent = this._appsCount.toString();
        },

        onCurrentWindowChanged: function(windowNode) {
          var elt = this.getElement().querySelector(".MyHeaderBarWidget-
title");
          if (windowNode) {
            elt.textContent = windowNode.attribute('text');
          } else {
            elt.textContent = "<NONE>";
          }
        }
      };
    });
  }
);
```

Adding Localized Texts

You can customize localization by adding your custom translation texts in the locale file and referencing them in a widget template file.

Adding a Custom Title for the Current Window

The Genero Web Client for JavaScript (GWC-JS) front-end provides a mechanism to internationalize your application interface. For more details about this mechanism see Translations for GWC-JS on page 47 For example to add a custom title for the current window, perform the following:

- **Add a Localization Key**

  In the `<project_dir>/src/locales` directory, choose your localization file or create your own one named `xx-YY.json`, where "xx-YY" follows the standard localization code used for languages. For example, to the locale file for French `<project_dir>/src/locales/fr-FR.json`, add your custom keys:

```
{
  "mycusto":{
    "window":{
      "currentTitle": "Titre de la fenêtre courante"
    }
  }
}
```

  In this case the translation key is *mycusto.window.currentTitle*

- **Reference the Translation Key in a Widget Template file**

  Reference your custom translation key with the HTML attribute *data-i18n* in the widget template file for the header bar. For example, in `<project_dir>/customization/default/js/MyHeaderBarWidget.tpl.html` add this within a `<span>` element with default text.

```
<span class="…" data-i18n="mycusto.window.currentTitle">Currently opened
 window title</span>
```

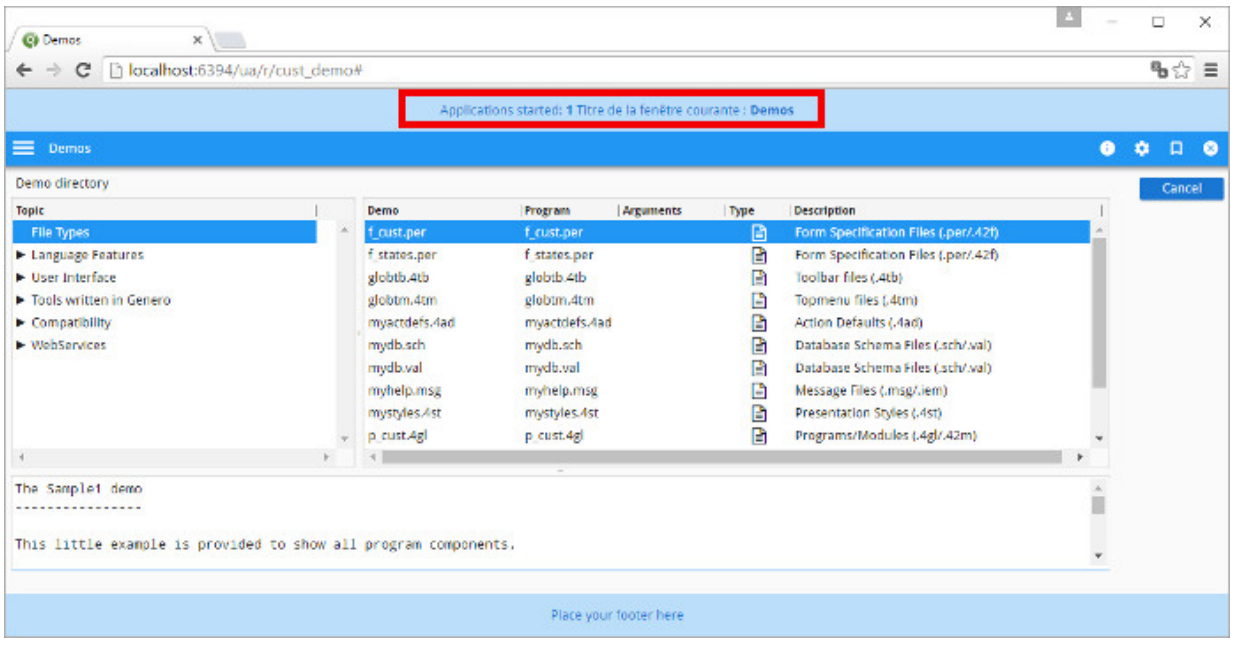


Figure 54: GWC-JS User Interface French Locale

An English browser displays "Currently opened window title", and a French browser displays "*Titre de la fenêtre courante*".

Activating Customization with custom.json

This topic provides information about how to activate customization for your Genero Web Client - JavaScript (GWC-JS) customization project.

About this task

The following procedure provides you with the steps to set compilation options and activate your GWC-JS project for customization using custom.json.

1. Navigate to your *project_dir* directory and open the `custom.json` file using a text editor.

```
{
  "compile": {
    "mode": "cdev",
    "customization": true,
    "with": {
      "nwjs": false,
      "android": false,
      "ios": false
    }
  }
}
```

2. To set the value for compilation *mode*, set the value for *mode* to `cdev`.

   Note: Compilation modes are set to "cdev" for development, and "prod" for production customization set.

3. To specify a project to activate, set the value for *customization* to a directory in your *`project_dir/`*`customization` that contains your project, for example `customization/my_project_directory` as shown in the example.

```
{
  "compile": {
    "mode": "cdev",
    "customization": "customization/my_project_directory",
    "with": {
      "nwjs": false,
      "android": false,
      "ios": false
    }
  }
}
```

   Tip: If you wish to use the default GWC-JS user interface instead of your customized project, you can simple set *customization* to `false`.

4. Save your changes.
5. To compile the changes, in the **Node.js command prompt** window, navigate to *project_dir* and run `grunt` or alternatively you can run `grunt dev` to continuously build the project.

What to do next

To view your customized GWC-JS Web client, open an application in your browser. You can do this by configuring an application for your project, see Configuring demo application for custom GWC-JS on page 207.

Configuring Applications for Custom GWC-JS

This topic provides information on how to configure applications to use your custom GWC-JS.

About this task:

To configure the application to use your custom GWC-JS, you need to configure the *GWC-JS* element in your application's `xcf` file as shown in this procedure.

Before you begin:

The following is assumed:

- Your have created a project directory *project_dir*
- You have created an alias/link in $*FGLASDIR*/web to point to the directory *project_dir*/dist/web. If not, see Configuring your Environment on page 198.

1. Create an application configuration file for your application, for example **myapp.xcf**. Provide an absolute path to the location of your compiled application files in the PATH element, and in the MODULE element specify the module required to launch your application.

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/
gas/3.00/cfextwa.xsd">
 <EXECUTION>
    <PATH>/home/generoapps/prog1</PATH>
    <MODULE>myapp.42m</MODULE>
 </EXECUTION>
 <UA_OUTPUT>
    <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
    <GWC-JS>gwc-dev</GWC-JS>
 </UA_OUTPUT>
</APPLICATION>
```

   Note: gwc-dev is a symbolic link from *$FGLASDIR*/web to your *project_dir*/dist/web directory.

2. Save the configuration file (e.g. myapp.xcf) in your $(res.appdata.path)/app directory.

What to do next

When you have completed the above steps, view the application in your browser. You can do this by starting the standalone dispatcher from the command line using httpdispatch and then opening the application in your browser by entering the URL http://localhost:6394/*ua/r*/myapp. Your application should load. If the application fails to load, see Troubleshooting on page 208.

Configuring demo application for custom GWC-JS

About this task:

To configure the GAS demos application to use your custom GWC-JS, you need to create a configuration xcf file for it and you need to configure its *GWC-JS* element as shown in this procedure.

Before you begin:

The following is assumed:

- Your have created a project directory *project_dir*
- You have created an alias/link in $*FGLASDIR*/web to point to the directory *project_dir*/dist/web. If not, see Configuring your Environment on page 198.

1. Create an application configuration file, for example **cust_demo.xcf**.

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

```
<APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/
gas/3.00/cfextwa.xsd">
  <EXECUTION>
     <PATH>$(res.path.fgldir.demo)</PATH>
```

```
      <MODULE>demo.42r</MODULE>
   </EXECUTION>
   <UA_OUTPUT>
      <PROXY>$(res.uaproxy.cmd)</PROXY>
      <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
     <GWC-JS>gwc-dev</GWC-JS>
      <TIMEOUT>Using="cpn.wa.timeout"</TIMEOUT>
   </UA_OUTPUT>
</APPLICATION>
```

Note: `gwc-dev` is a symbolic link from *`$FGLASDIR`*`/web` to your *`project_dir`*`/dist/web` directory.

2. Save the configuration file (e.g. cust_demo.xcf) in your `$(res.appdata.path)/app` directory.

What to do next

When you have completed the above steps, view the demo application in your browser. You can do this by starting the standalone dispatcher from the command line using `httpdispatch` and then opening the GAS demos application in your browser by entering the URL `http://localhost:6394/ua/r//cust_demo`. The demo application should load. If the application fails to load, see Troubleshooting on page 208.

Troubleshooting

If the application fails to load in your browser, for example you get:

- “Unable to expand bootstrap template”. This means that your “bootstrap.html” has not been found. You may need to rebuild your project directory.
- Or if the GWC-JS displays a blank page, check to see in your browser's debug window if you are getting a JavaScript "HandleBars is missing" message. This may indicate that you have forgotten to do `grunt deps` before compiling your custom GWC-JS with `grunt`.

For more information see Configuring your Environment on page 198, or contact your local Four Js support center.

Migrating from GDC to GWC-JS

There are features supported by the Genero Desktop Client (GDC) that are not fully supported by the Genero Web Client for JavaScript (GWC-JS). Understanding these limitations and planning for them during the migration will greatly help your migration efforts.

Read about the HTML5 theme and understand the features and limitations prior to migrating from the GDC application to the GWC-JS. This will help you determine whether the GWC-JS is a suitable Front End for your application.

Note: This document does not intend to be complete. It gives you tips and recommendations based on experience.

- Evaluate your GWC-JS application needs on page 208
- Migration tips on page 210

Evaluate your GWC-JS application needs

Even if most Genero Desktop Client (GDC) features are supported in Genero Web Client for JavaScript (GWC-JS), you still have points to consider before migrating.

Before you migrate your application for the GWC-JS, examine the limitations and authentication specifics. You must modify your application to handle unsupported features.

- Features and limitations on page 176
- Evaluate authentication on page 210

One method of evaluating is to run the application and see what happens! If you encounter an issue, you need to determine if this is a limitation, a customization or a bug. See Migration tips on page 210 for hints aimed at helping you solve your issues during migration.

Evaluate limitations

These limitations are grouped under rendering or behaviors.

Rendering

Pop-up windows

If you have some pop-up windows in your GDC application, set the *windowType* attribute to *modal* in the Genero Style for `Window.dialog` in order to achieve a similar display with the GWC-JS.

> **Note:** Any style (`4st`) element bound to a Window may contain the `windowType` attribute.

Excerpt from `$FGLDIR/lib/default.4st`:

```
<Style name="Window.dialog">
  <StyleAttribute name="windowType" value="modal" />
  <StyleAttribute name="sizable" value="no" />
  <StyleAttribute name="position" value="center" />
  <StyleAttribute name="actionPanelPosition" value="bottom" />
  <StyleAttribute name="ringMenuPosition" value="bottom" />
  <StyleAttribute name="toolBarPosition" value="none" />
  <StyleAttribute name="statusBarType" value="none" />
  <StyleAttribute name="errorMessagePosition" value="popup" />
</Style>
```

If you use the predefined style dialog, the window displays as a pop-up window.

Example of Genero code:

```
OPEN window msg WITH FORM "MyWindow" ATTRIBUTES (STYLE="dialog")
```

Multiple Document Interface

Multiple Document Interface (MDI) is not applicable for web applications.

Behaviors

File transfer

Due to browser securities, `FILE_TRANSFER` `on page 317` in GWC-JS is not transparent. The browser asks for permission before downloading anything on the client side.

Front End Calls

Due to browser securities, intrusive operations cannot be performed. Front end calls that access the desktop file systems (disk) are not supported (including file transfer). Check the available Front End function calls for GWC-JS in the *Genero Business Development Language User Guide*. The alternative is to write your own Front End Call in JavaScript™.

Evaluate authentication

Most GDC applications connect to the server with the user permissions and profile. As a result, the applications are run under this user identity.

When the application is deployed through GAS, the user is the user that started the GAS. All the applications are launched using that specific user, unless you ask GAS to impersonate. Depending on the authentication system, adaptations may be needed.

Authentication

The Single sign-on (SSO) authentication mechanism now works for both Genero Desktop Client (GDC) and GWC-JS. If you were using SSO with GDC, you can continue to use it with the GAS. See How to implement Single sign-on (SSO) on page 114 for more details.

System users

If only the login is needed, you can get the web server to make a preliminary authentication. This authentication is transmitted to Genero applications by the dispatcher as an environment variable. The user login can be retrieved from the runtime environment with the FGL instruction:

```
fgl_getenv("FGL_WEBSERVER_REMOTE_USER")
```

Application login

With an application login, there is nothing additional to do as the application handles the login.

Migration tips

When migrating an application from the Genero Desktop Client (GDC) to the Genero Web Client for JavaScript (GWC-JS), you may encounter issues. Here are some tips for when you encounter migration issues.

Topics:

Configuration issues

The GAS is likely serving several types of applications, which might use different versions of the Genero Business Development Language and different databases. You need to tell the GAS in which environment your application will be running. This is done through a configuration file.

If you already have your application configured for GDC-HTTP, you simply need to enable the rendering for GWC-JS.

Example of configuration for a GDC application:

```
<APPLICATION Id="my-app" Parent="defaultgdc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

To allow GWC-JS rendering, simply replace *defaultgdc* by *defaultgwc*:

```
<APPLICATION Id="my-app" Parent="defaultgwc">
```

```
  <EXECUTION>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

The first time you configure an application for GWC-JS, see the Configuring applications on GAS on page 95 topic to configure your execution environment, the database access and the resource deployment (pictures, reports, and so on).

If the application fails to start, check the Genero Application Server logs to see what happened. Check the dispatcher log first, then check the session log. To get the session number, click on AUI tree and have a look at the application URL or view the HTML source page.

Example Log Entry:

```
/ua/sua/7e26fadb0c9f6939c65702fc9a1ff2a4/0
```

The session number is *7e26fadb0c9f6939c65702fc9a1ff2a4*.

To debug the configuration, see the Using the debugger on page 157 topic to assist you in setting up a console. With a console, the GAS does not launch the application, but instead launches a console with the application environment set up. You can then compare the environment variables to your GDC working environment, you can run command lines to check and update your environment variables, and you can even display the application on the GDC by changing the FGLSERVER variable and verify that the application runs properly with GDC.

Rendering issues

Graphical widgets that are not rendered properly or displayed with a different style are considered rendering issues. Examples of rendering issues include misalignments, 4st colors not applied, and widgets that are rendered when they should not.

When you encounter a rendering issue:

1. First check in AUI tree (equivalent of GDC debug tree for GWC-JS) to verify that node and properties of the user interface screen you are currently working on are loaded properly and that the widgets attributes are correctly set. See Configuring development environment on page 147
2. Use *Firebug* on FireFox or *Developer Tools* for Internet Explorer to check:

- That the resources files (.css, .js, and so on) are accessible and not displaying a 404 error message.
- That the HTML is properly generated. For example, check whether the color for the edit is set or is missing in the generated page.

Application issues

Application issues are behavior related. Most of the time you need to contact your local support center. But you can first check for:

- Relevant messages in the logs
- JavaScript™ errors

If clues are provided in either the logs or in the errors, try to build a simple test program that replicates the issue and contact your local support center.

If your application involves a Rich Text version of a textedit widget, review the migration note regarding Rich Text editing and the differences between the Genero Desktop Client and the Genero Web Client for JavaScript.

Network issues

Once the migration is almost done and you are in load tests phase, you might encounter sporadic disconnections. Carefully read the chapters on GAS installation for your web server and verify the web servers and GAS timeouts are compatible. You can also have a look at:

- Web server log (error/access log)
- The Genero Application Server logs
- Network sniffer (like Wireshark)

Web Components

In GAS 2.50, web components are deployed under *$FGLASDIR*/web/components directory.

Starting from GAS 3.00, with uaproxy, the default path for a web component is *appdir*/webcomponents, where "appdir" is the application directory. See the *WEB_COMPONENT_DIRECTORY* element in your *$FGLASDIR*/etc/as.xcf configuration file:

```
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
   […]
   <DVM>$(res.dvm.wa)</DVM>
   <WEB_COMPONENT_DIRECTORY>$(application.path)/webcomponents</
WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

You can change the default web components location by configuring a *WEB_COMPONENT_DIRECTORY* element in your application's configuration. In this example, the web component is no longer located in *appdir*/webcomponents but in *appdir*/mycomponents.

```
<APPLICATION Parent="defaultgwc" …>
  <EXECUTION>
    <PATH>/home/myapp</PATH>
    <MODULE>myapp</MODULE>
   <WEB_COMPONENT_DIRECTORY>/home/myapp/mycomponents</
WEB_COMPONENT_DIRECTORY>
  </EXECUTION>
</APPLICATION>
```

Migrating from GWC-HTML5 to GWC-JS

Moving your Genero Web Client for HTML5 (GWC-HTML5) applications to Genero Web Client for JavaScript (GWC-JS) involves some changes to the customization used, topics to help your migration efforts.

> **Note:** It is important to review the features and limitations prior to migrating from the GWC-HTML5 application to the GWC-JS. This topic provides you with tips and recommendations based on migration experience.

Navigating Open Applications

With GWC for HTML5, each application started with RUN or RUN WITHOUT WAITING opens a new tab in your browser. GWC for JavaScript provides a **Navigation** panel (sidebar). Each application started with RUN or RUN WITHOUT WAITING replaces the application in the current window. You can access the other applications and make them current by selecting them from the **Navigation** panel or sidebar. See User interface: navigation on page 181.

File Upload to Server

To upload a file with GWC-HTML5 Web client, you used an `EDIT` field with the style `FileUpload` to create a file chooser dialog.

```
EDIT sfile1=formonly.sfile1, style="FileUpload";
```

> **Important:** The `FileUpload` style is not supported by GWC-JS.

To migrate from GWC-HTML5 to GWC-JS, you need to remove `FileUpload` style and add a call to `openFile` front call, followed by call to `fgl_getfile`. This is the same file upload method as you use in Genero Desktop Client (GDC):

1. Use the `openFile` front call to open the file chooser dialog so that the user can select a file to upload.

```
DEFINE infile STRING

    CALL ui.Interface.frontCall("standard", "openFile",
    ["c:\\fjs\\doc","doc.pdf","*.pdf","Choose a file to upload"],
    infile)
```

> **Note:** For GWC-JS, the path parameter is ignored, and wildcards can only hold one type of file extension. For more information on the use of this command, please see the "Standard front calls" section in the *Genero Business Development Language User Guide*.

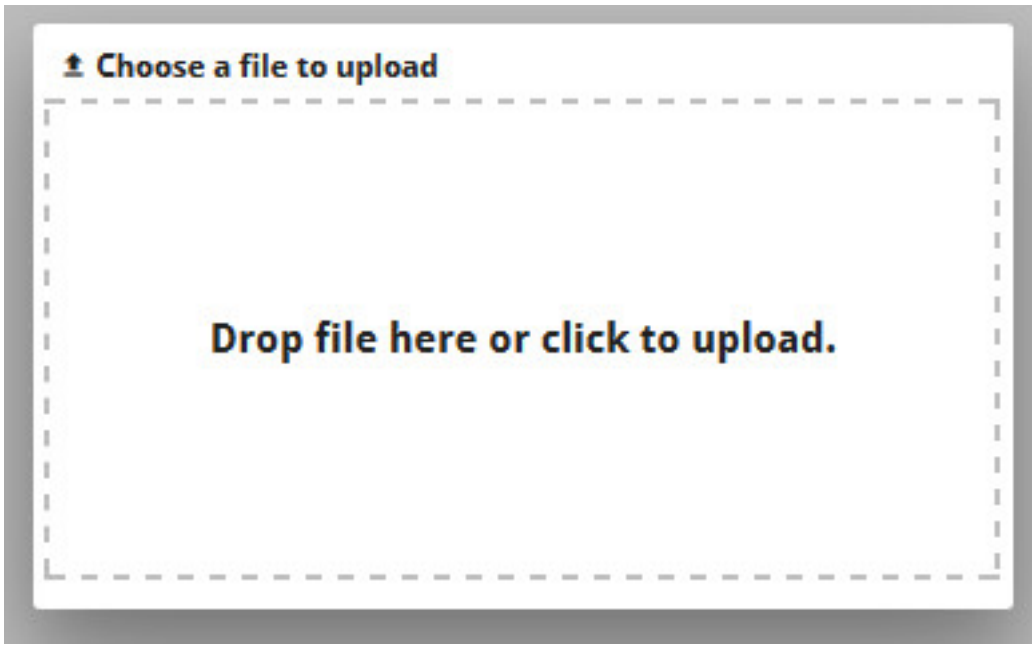


Figure 55: GWC-JS File Upload Pop-up Window

2. Use the `fgl_getfile` to upload the file the user has chosen. The call to the `fgl_getfile` function requires no interaction from the user so it can be called immediately to upload the file to a directory specified in the application server.

```
TRY
        CALL fgl_getfile(infile, /opt/myapp/received_files)
        CALL fgl_winmessage("File uploaded", infile, "info") # Display a
 window with message after uploading
    CATCH
        ERROR sqlca.sqlcode, " ", sqlca.sqlerrm  # Catch runtime execution
 errors from the SQLCA diagnostic record
    END TRY
```

Genero Web Client for HTML5 (GWC-HTML5)

GWC-HTML5 delivers your applications over the Web using browser-based themes. This version of the GWC has been deprecated; new development should use the Genero Web Client for JavaScript (GWC-JS) instead.

GWC for HTML5

The GWC-HTML5 is installed as part of the Genero Application Server.

The GWC-HTML5 allows you to deliver true Web applications with applications developed in the Genero Business Development Language (BDL). Having the underlying source written in Genero BDL means that the GWC is flexible enough to let you build from a simple Web application to a corporate Web application with only a few limitations. It brings BDL applications to the Internet world and the ability to be integrated in a Web site.

GWC-HTML5 uses themes, comprised of *templates* and *snippet sets*, to create dynamic web pages.

Note: For details regarding these themes, consult the *Genero Application Server 2.50 User Guide*.

The GWC-HTML5 is no longer the default client for all Genero Web Client applications. The GWC-JS is another Web client that is also installed as part of the Genero Application Server, see Genero Web Client for JavaScript (GWC-JS) on page 173.

How the GWC-HTML5 works

The GWC-HTML5's snippet-based rendering engine (SBRE) creates application Web pages dynamically on the Application Server for delivery to the client browser using technologies, such as Client-Side Framework, understood by a browser. The AUI tree (provided by the DVM) and the template and snippet files (set by the application configuration) are used to create an XML / xHTML document that is passed to the user agent.

It can deliver the application to any device equipped with a Web browser.

Genero Web Services

Add a Web service application.

When configuring a Web services application, you can either provide the configuration details in the GAS configuration file, or you can create a separate application-specific configuration, see Configuring applications for Web service on page 101 (one per application). For complete details on configuring Web services applications and the configuration parameters and settings, see SERVICE_LIST on page 346.

The topics in this section provide more options for accessing, validating, and delivering Web Services applications on the GAS.

Accessing the Web Service (Web Services URI information)

Examples of the Web Services URI.

In the following examples, "appid" would be replaced by the application Id and "service" would be replaced by the name of the service.

To get the WSDL for a specified service:

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

To access the Web service:

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

If the Web service uses a group:

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

Access through a Web Server:

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

Service invalidation

The service invalidation feature provides notification for when the configuration of a Genero Web service is invalid. The configuration must be modified before the service can start.

A Web service is typically an automated process, called on behalf of a user or process. When a Web service has an invalid configuration, there is no graphical message informing an end user of the error. Requests continue to fail. To prevent requests from failing forever, if the Genero Web Services service (gwsproxy) cannot start within `DVM_AVAILABLE` on page 313 time, it informs the dispatcher.

For troubleshooting and monitoring Web services, please see Example: Get list of services invalidated by dispatcher on page 56.

The next time a request is received, the dispatcher remembers that the `Proxy: gwsproxy` on page 267 cannot be started and sends a 503 HTTP status to the user agent.

The first time a service is not able to start, this message is returned:

Application or service has been stopped due to a fatal error

Any new request returns this message until the configuration is modified and corrected:
Bad configuration prevents application or service to start.

The service resumes when the configuration is updated.

If the service is configured in the Genero Application Server configuration file, you must restart the Genero Application Server.

If the service is configured in an external application configuration file (`xcf`), the configuration is reloaded each time it is modified. You do not need to restart the Genero Application Server.

This feature is only supported for Genero Web services. It is not supported for Genero Desktop Client or Genero Web Client applications.

Sticky Web services

A sticky Web service is instantiated to handle all requests coming from a specific user agent. The gwsproxy manages HTTP cookies to implement sessions with sticky Web services.

A *sticky Web service* is a Web service that uses an HTTP cookie to ensure that requests from a specific user agent are always routed to the same DVM handling the Web service by the Genero Application Server.

The first time a user agent connects to the Genero Application Server, the `Proxy: gwsproxy` on page 267 starts a new DVM to handle the sticky Web service instance. The sets an HTTP cookie, called `GWS_S_SESSION`, in the first response to the user agent. This cookie is then sent with any further requests

from the user agent to the Genero Application Server. The `gwsproxy` can use the cookie to identify the DVM in charge of that cookie and dispatch the request to the correct DVM.

Configure a sticky Web service

To enable sticky sessions in a Web service, add the mode attribute to the application node in the Web service application configuration, and set the value to "sticky". The `KEEP_ALIVE` element specifies the session lifetime, in seconds.

```
<APPLICATION Parent="ws.default" mode="sticky">
<EXECUTION>
  <PATH>$(res.path.app)/services/sticky-ws-service</PATH>
  <MODULE>sticky-main</MODULE>
</EXECUTION>
<TIMEOUT>
  <KEEP_ALIVE>60</KEEP_ALIVE>
</TIMEOUT>
</APPLICATION>
```

You can change the name of the cookie by setting the environment variable `FGL_GWSPROXY_COOKIE_NAME` in the application configuration with the value of the new cookie name.

```
<APPLICATION Parent="ws.default" mode="sticky">
<EXECUTION>
  <ENVIRONMENT_VARIABLE Id="FGL_GWSPROXY_COOKIE_NAME">NEW_NAME
  </ENVIRONMENT_VARIABLE>
  <PATH>$(res.path.app)/services/sticky-ws-service</PATH>
  <MODULE>sticky-main</MODULE>
</EXECUTION>
<TIMEOUT>
  <KEEP_ALIVE>60</KEEP_ALIVE>
</TIMEOUT>
</APPLICATION>
```

When does a session expire

The `KEEP_ALIVE` element specifies the session lifetime, in seconds. A DVM that does not get a request from the user agent it is servicing will stop after the time specified by `KEEP_ALIVE` has passed.

If a request comes from the user agent after the time specified by `KEEP_ALIVE`, and therefore after the DVM has stopped, it will return an HTTP 400 error.

Stop a sticky Web service

To properly stop a sticky Web service, have a dedicated method to be called by the user agent when it needs to close the session. The DVM can respond to the request, unset the cookie, and stop the Genero program properly.

Deploying with Genero Archive

The Genero Archive deployment feature provides a simple process for packaging applications and services into an archive to deploy on a GAS installation. After deployment, the applications and services packaged are available and can be used.

The deployment framework provides an interface to:

- Deploy applications and services packaged into a Genero Archive
- List deployed archives
- Disable a deployed archive
- Enable / disable applications and services provided by an archive
- List enabled / disabled applications and services
- Undeploy a deployed archive

The deployment of applications and services does not include:

- Database installation and setup
- Any Genero software packages installation and setup
- Any other form of external dependencies

Any operation on Genero Archive MUST be performed in mutual exclusion to ensure the deployment process and archive management safety.

What is a Genero Archive?

A Genero Archive is a zip archive containing a MANIFEST file providing installation instructions and the list of application and services to make available.

The Genero Archive is a simple zip file of a directory (or files). The archive name can be any name you wish. It need not reflect the applications or services contained within.

The `MANIFEST` file provides deployments instructions, and must be at the root of the archive tree.

It is possible to embed external tools into a Genero Archive, as the content is not strictly restricted to compiled Genero applications. But taking advantage of it must remain the responsibility of the user packaging the applications (portability considerations, etc.)

Example

An example Genero Archive file could include:

- `./MANIFEST`
- `./modules/app.42m`
- `./modules/app.42r`
- `./forms/app.42f`
- `./xcf/app.xcf`

In the previous example, the files were organized within the root directory by a series of sub-directories. Such directories are not required. We could have placed all the files directly in the root and have provided the following archive contents for the same application:

- `./MANIFEST`
- `./app.42m`
- `./app.42r`
- `./app.42f`
- `./app.xcf`

Quick start: deploying applications

The procedures in this section provide you with methods to deploy applications and services.

Application deployment overview

Some ways to manage public and private images when deploying applications.

Before deploying applications, it is recommended that you plan how images are going to be used by your applications so as to take advantage of the optimization and caching feature provided by the GAS for Web and GDC applications. For examples, you can divide your images into these two categories:

Public images	Images that are common or that can be shared by all your applications
Private images	Images that are private or specific to an application

Public images shared by your applications

Images (e.g. logos, background images, etc.) that are common to all or several of your GWC-JS applications and that do not change during an application's lifetime, are considered "public".

This means that at runtime images found in subdirectory paths of the *$(res.public.resources)* and the *$(res.appdata.path)*`/public/deployment` directory (see Paths to application resources on page 219) will be put in the browser cache where they can be delivered quickly to the front end client without having to access an application's configuration file each time.

You can deploy public images in either of the following ways:

- Using the deployment framework (Recommended):

  Initially, you need to place public images in a specific directory in the application archive, see Building an archive with public resources on page 221. The `resources` switch of the framework (e.g. `fglgar --resources`) specifies the directory in the archive with your public images. Then when you deploy the archive, the deployment framework will deploy it as follows:

  - A subdirectory is created in your *$(res.appdata.path)*`/deployment` directory identified by the archive name and the date and time deployed, e.g. *$(res.appdata.path)*`/deployment/myApp1_20150423-130838`.

    The application configuration file (`xcf`) and all the source files contained in the archive (`gar`) are placed in this directory.

- Another subdirectory is created in your *$(res.appdata.path)*/public/deployment/ directory identified by just the archive name, e.g. *$(res.appdata.path)*/public/deployment/myApp1.

  All the image files contained in the archive's resource directory are copied into this directory and the xcf files of the archive will get updated in order to have their public directory resource *$(res.public.resources)* set to this directory.

- Or if you would prefer not to use the deployment framework, you must then copy your public images by hand into the directory specified in your *PUBLIC_IMAGEPATH*.

  Caution: Public images should not be placed in the /public root directory as the fglrun does not look for images to be served via the GAS there; searches start in subdirectory paths, i.e. public/common and public/deployment.

Sample application and image deployment

```
####$(res.appdata.path)
       ####deployment
       #        ##myApp1_dateTimeStamp (deployed application and private
 images)
       #              ##webcomponents (deployed application's Web
 components)
       ####public
              ##-common (default PUBLIC_IMAGEPATH directory containing
 public images)
              #
              ##-deployment
                     ##myApp1 (public images deployed with application by
 deployment framework)
```

Private images of your application

Private images are resources that are only used by one application. Before deploying, you should place them in the root directory of the application's archive. Then when you deploy the application with fglgar, they will be placed in the $(res.appdata.path)/deployment/ directory created for the application. The fglrun automatically searches for application resources in the application's root directory.

Note: If you change your application's private resource location down into subdirectories of the root, you will need to ensure that the environment variable *FGLIMAGEPATH* is included in the application's configuration file (.xcf) and is configured correctly, as shown in the example.

```
<ENVIRONMENT_VARIABLE Id="FGLIMAGEPATH">pics$(sep)images$(sep)private/
images</ENVIRONMENT_VARIABLE>
```

You can specify a hierarchy of directories, e.g. "private/images", but then only the $(root)/private/images directory will be searched by fglrun for images, not the parent directory. If there are resources in several subdirectories then you have to specify each subdirectory separately (note *$(sep)* is a built-in path delimiter which can be used for both Windows™ and Unix™ platforms). For more details on *$FGLIMAGEPATH*, see the *Genero Business Development Language User Guide*.

Paths to application resources

Describes paths to directories specific to application resource used by the DVM.

Starting with Genero 3.00, three new predefined resources have been added:

- A public resource path for all applications,*$(res.appdata.path)*/public
- A resource for common images used by applications, *$(res.public.resources)*, which expands to *$(res.appdata.path)*/public/common

- A resource where the Application Server stores files temporarily during file transfer, $(res.path.tmp), which expands to *$(res.appdata.path)/*`tmp`*/$(dispatcher.name)*

At dispatcher start up, a public directory with write permissions located in *`$(res.appdata.path)/`*`public` is created (if it doesn't already exists) which contains two sub directories:

- `common`
- `deployment`

These directories will contain public images (i.e. images common to all applications) that are cacheable by the browser and from where they can be delivered quickly to the front end client without having to access an application's configuration file each time.

For details about deploying public images, see Application deployment overview on page 218 and Building an archive with public resources on page 221.

> **Note:** To ensure that resources are cacheable by the browser, the default public resource settings, `/public` and `/public/common,` are not configurable.

Quick start: Genero Archive

Follow these steps to quickly archive and deploy applications and services.

As a prerequisite, you must have your applications and/or services created and tested. You must have created an external configuration file for each application or service.

This initial procedure provides you with a quick overview of the main tasks of archiving and deploying an application followed by some examples of building archives and deploying applications:

- Building an archive with public resources on page 221
- Building an archive with deployment triggers on page 221
- Deploy your application on your machine on page 221
- Run the deployed application on page 222

For a full understanding of what Genero archiving offers, please read all archiving topics.

1. Consolidate all necessary files under a root directory.

   Within this directory, you must have a subdirectory for public images if you are deploying public images with your application, see Building an archive with public resources on page 221. You can also have sub-directories as needed to assist with the organization of your files.

   > **Note:** When you create the archive, you will specify a root directory only. All files and directories within that directory will be included in the archive. If you wish to add a subset of the files in a directory, you can use another zip tool to create the file. See Create a Genero Archive on page 226.

2. Update the application's configuration file (`xcf`) to set all resources relative to the resource `$(res.deployment.path)`. See Create a Genero Archive on page 226.
3. Create a MANIFEST file by hand (optional) or (recommended) use the deployment framework feature of `fglgar` to automatically generate the MANIFEST.

   See The MANIFEST file on page 224 and The fglgar command on page 267.
4. Create the archive file.

   See Create a Genero Archive on page 226.
5. Deploy the archive file.

   If you are on the application server, you deploy with the `gasadmin` command. See Deploy an archive on page 227.

   If you are on a remote computer, you can deploy with the FGL tool, `PublishGar`, or you can use other tools such as `curl`. See Genero Archive deployment service on page 232.
6. Enable the archive.

   See Activate (enable) a deployed archive on page 229.

The applications, services and resources included in the archive are available for your end users.

Building an archive with public resources

About this task:

This procedure guides you through steps for building an archive for an application that is deployed with public images. At this stage it is assumed you have :

- Consolidated all the necessary files for your application archive under a root directory.
- Updated the application's configuration file (e.g. `xcf`) to set its deployment <PATH> element to the resource `$(res.deployment.path)`.

1. Put your application's public images in a dedicated directory (you can name it, for example, "myAppPublicImages") in your archive.
2. Create the MANIFEST file by hand (optional) or (recommended) use the deployment framework feature of `fglgar` to automatically generate the MANIFEST, as shown in the next step.

   The *RESOURCES* element specifies your public images directory. See Application deployment overview on page 218 and The MANIFEST file on page 224.
3. Create the archive (`gar`) file to deploy your application by typing the following in a terminal window of the archive directory:

   ```
   fglgar --gar --resource myAppPublicImages --application myApp.xcf
   ```

   A Genero archive file (`gar`) is created in your current directory that has the same name as the directory. For more information about `fglgar`, see The fglgar command on page 267.

Building an archive with deployment triggers

About this task:

This procedure guides you through steps for building an archive for an application that is deployed with trigger commands or programs to execute when deploying and undeploying your application on the GAS. Before you begin make sure you have:

- Consolidated all the necessary files for your application archive under a root directory.
- Updated the application's configuration file (e.g. `xcf`) to set its deployment <PATH> element to the resource `$(res.deployment.path)`.

1. Create the MANIFEST file by hand (optional) (see The MANIFEST file on page 224) or (recommended) use the deployment framework feature of `fglgar` to automatically generate the MANIFEST while creating the archive file, as shown in the next step.
2. Create an archive (`gar`) file to deploy your application.

   To have `fglgar` create the MANIFEST file and the archive (`gar`), type the following in a terminal window of the archive directory.

   > **Note:** The *--deploy-trigger* and *--undeploy-trigger* (see TRIGGERS (for manifest) on page 224) switches contain your *DEPLOY* and *UNDEPLOY* commands, as shown in the example.

   ```
   fglgar --gar --application myApp.xcf --trigger-component
   cpn.wa.execution.local --deploy-trigger "fglrun mydeploy.42r" --undeploy-
   trigger "fglrun myundeploy.42r"
   ```

   A Genero archive file (`gar`) is created in your current directory that has the same name as the directory. For more information about `fglgar`, see The fglgar command on page 267.

Deploy your application on your machine

About this task:

Once you have created an archive for your application in the steps above, you can now deploy your application locally on your machine to test it by performing the following tasks:

1. Start the standalone dispatcher from the command line by typing `httpdispatch`.
2. Deploy your (`gar`) file locally on your machine.

   To deploy an archive named `myApp_deploy.gar`:

   ```
   fglrun $(FGLDIR)/web_utilities/services/deployment/bin/PublishGar http://
   localhost:6394/gas deploy myApp_deploy.gar
   ```

3. Enable your deployed application locally on your GAS with the `PublishGar` tool.

   ```
   fglrun $(FGLDIR)/web_utilities/services/deployment/bin/PublishGar http://
   localhost:6394/gas enable myApp_deploy.gar
   ```

   This enables the application by copying its configuration file (e.g. **myApp.xcf**) to your `$(res.appdata.path)` directory. It modifies the application's `xcf` file so that `fglrun` knows where to look first for the application-specific resources.

Run the deployed application

About this task:

Once you have deployed your application on your machine in the steps above, you can now run your application. If the standalone dispatcher is not already started, start it from the command line using `httpdispatch`:

In a browser enter the address of your deployed application, e.g. `http://localhost:6394/ua/r/myApp`
You should see your application displayed and be able to interact with it. You have successfully deployed an application.

Deploying application resources for GWC for HTML5

To enable access to your images and Web components add them to one of the default directories.

This topic describes features the GAS provides for deploying resources with applications for GWC HTML5v1 applications.

Deploying application images for GWC for HTML5

By default, the GAS looks for images in `$FGLASDIR/pic` and in the application directory (defined by the `PATH` element). To quickly enable access to your images, add them to one of these directories.

The default image directory is set by the `WEB_APPLICATION_PICTURE_COMPONENT` element, see WEB_APPLICATION_PICTURE_COMPONENT on page 357.

For example, if you are deploying an application using GWC for HTML5, this `WEB_APPLICATION_PICTURE_COMPONENT` defines the default image directory:

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.html5.picture">
  <PATH Id="Resource" Type="WEBSERVER">$(connector.uri)/fjs</PATH>
  <PATH Id="Image" Type="APPSERVER"
   ExtensionFilter="$(res.image.extensions);.less;.svg">
    $(res.path.tpl.html5)/img;$(res.path.pic);$(application.path)</PATH>
  <PATH Id="SetHtml5" Type="APPSERVER"
   ExtensionFilter="$(res.web.extensions);.less;.svg"
   DVMFallbackAllowed="FALSE">$(res.path.tpl.html5);$(res.path.tpl.common)</
PATH>

</WEB_APPLICATION_PICTURE_COMPONENT>
```

Note: For your legacy GWC-HTML5 applications, the `WEB_COMPONENT_DIRECTORY` configuration entry is ignored. Web components in this case are still required to be located in *`$(FGLASDIR)/`*`web/components` as before. For more details on web component usage, see the *Genero Business Development Language User Guide*.

To explicitly override with an image path specific to the application, add a `PICTURE` element to the application configuration file. In this example, a PICTURE path is added for the GWC for HTML5 theme:

```
<APPLICATION Parent="defaultgwc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/
cfextwa.xsd">
  <RESOURCE Id="application.path"
   Source="INTERNAL">$(res.path.demo.app)/card/src</RESOURCE>
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
   <PICTURE>
        <PATH Id="Image" Type="APPSERVER">
          $(res.path.demo.app)/card/src/photo;
          $(res.path.pic);$(application.path)</PATH>
      </PICTURE>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

Deploying application resources for GWC-JS

To enable access to your application's images and Web components, add them to the relevant directory.

This topic describes features the GAS provides for deploying resources with applications for GWC-JS applications.

Deploying application images for GWC-JS

A public data directory (see Paths to application resources on page 219) is used especially for `ua` protocol applications to enable quicker access to common or public resources such as images, reports, etc. belonging to all deployed applications on the GAS.

This allows the dispatcher to cache resources, which can then be served without having to access an application's configuration file each time.

For example, if your FGL application has an instruction to display a public image, such as `DISPLAY "photo.jpg"`, you must put the image file in the default *`$(APPDATA)/`*`public/common` , the default *PUBLIC_IMAGEPATH* directory so that it can be correctly served by the GAS.

You can also use the application deployment framework method to add public images, see Application deployment overview on page 218 and Building an archive with public resources on page 221.

Deploying web components for GWC-JS

You can add Web components manually for your applications in the default *`$(application.path)/`*`webcomponents` directory, or you can add them to any directory provided the `WEB_COMPONENT_DIRECTORY` element is correctly configured to reflect it, see WEB_COMPONENT_DIRECTORY on page 361

Genero Archive lifecycle

A Genero Archive has five stages in its lifecycle: deployed, active, deactivated, undeployed, and cleaned.

Once you have created a Genero Archive, the lifecycle of that archive can include these five stages:

- *Deployed* - The archive is deployed but not activated.
- *Active* - Applications and services provided by an archive are available to users.
- *Deactivated* - Applications and services provided by an archive are no longer available to users. A deactivated archive can be reactivated.
- *Undeployed* - The archive is no longer deployed, and cannot be reactivated. The MANIFEST file is removed. Physical cleanup of the archive is not done.
- *Cleaned* - The archive is physically removed.

The MANIFEST file

A Genero Archive `MANIFEST` file is a simple XML file providing the list of applications and services to make available. The `MANIFEST` file MUST be included in any Genero Archive file. You must create a MANIFEST file for each archive you create.

A `MANIFEST` file has a `MANIFEST` element at its root. Child elements can include the following,

> **Important:** Element order. If child elements are present, they must be set in the order listed below or as shown in the sample.

- One `DESCRIPTION` element. The `DESCRIPTION` element provides a textual description of the archive. It displays when listing archives.
- 0, or 1 `TRIGGERS` element. The `TRIGGERS` element (see TRIGGERS (for manifest) on page 224) specifies a trigger component which contains a program (e.g. `fglrun`) or script to run to deploy or undeploy the application if the deployment framework is used.
- 0, or 1 `RESOURCES` element. The `RESOURCES` element specifies the directory in your archive file where images that are specific to your applications are found. The directory must be in the archive.
- 0, 1 or more `APPLICATION` elements. The `APPLICATION` element takes one mandatory attribute. The `xcf` attribute specifies the path to the application configuration file. The application configuration file must be in the archive. Include an `APPLICATION` element for each application in the archive.
- 0, 1 or more `SERVICE` elements. The `SERVICE` element takes one mandatory attribute. The `xcf` attribute specifies the path to the Web service configuration file. The Web service configuration file must be in the archive. Include a `SERVICE` element for each service in the archive.

Sample MANIFEST file:

```
<MANIFEST>
  <DESCRIPTION>my description</DESCRIPTION>
  <TRIGGERS component='comp_name'>
    <DEPLOY>deploy.sh</DEPLOY>
    <UNDEPLOY>undeploy.sh</UNDEPLOY>
  </TRIGGERS>
  <RESOURCES>res_dir</RESOURCES>
  <APPLICATION xcf='app.xcf' />
  <SERVICE xcf='app.xcf' />
</MANIFEST>
```

- TRIGGERS (for manifest) on page 224

TRIGGERS (for manifest)

The `TRIGGERS` element defines a set of deployment parameters that can be used when deploying an application with the deployment framework. It takes a **component** attribute, which specifies the

environment context for the runtime (`fglrun`) required by the DEPLOY and UNDEPLOY elements. These child elements specify commands to execute when deploying and undeploying applications on the GAS.

Syntax

```
<TRIGGERS component ='comp_name'>
   <DEPLOY> myDeploy </DEPLOY >
   <UNDEPLOY> myUnDeploy  </UNDEPLOY>
</TRIGGERS>
```

Child elements

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

1. Zero or one `DEPLOY` element.
2. Zero or one `UNDEPLOY` element.

Example

```
<TRIGGERS component = 'cpn.wa.execution.local'>
  <DEPLOY>fglrun mydeploy.42r</DEPLOY>
  <UNDEPLOY>fglrun myundeploy.42r</UNDEPLOY>
</TRIGGERS>
```

In this example, the **component** value - **cpn.wa.execution.local** - can be referenced by the `--trigger-component` and the `DEPLOY` value by `--deploy-trigger` when you are creating an application archive using `fglgar`, for example:

```
fglgar --gar --application myapp.xcf --trigger-component
cpn.wa.execution.local --deploy-trigger "fglrun mydeploy.42r" --undeploy-
trigger "fglrun myundeploy.42r"
```

Note:

- In the case that the `--deploy-trigger` fails, the entire deployment will fail. Whereas if the `--undeploy-trigger` fails, the undeployment is still carried out.
- Deployment triggers are typically not required, you can deploy your applications without them by simply specifying the application's `xcf` file. See The MANIFEST file on page 224.

Parent elements

This element is a child of the MANIFEST element in The MANIFEST file on page 224

File system layout of a deployed archive

A Genero Archive contains application source files organised into subdirectories for modules, forms, etc. includes the MANIFEST and the external application configuration file. The directory structure is recreated when deployed.

Genero Archives are unpacked into a deployment directory. The deployment directory path is available and configurable in the GAS configuration files. The resource `res.deployment.root` points to the root directory for deployed archives. By default, it is configured with the value `$(res.appdata.path)/deployment`.

Each archive is unpacked into a dedicated directory under the `$(res.deployment.root)` directory. By default, the deployment tool will use the archive name (without the file extension) as the deployment directory. The directory name is completed with a timestamp representing the installation date.

Example

If the Genero Archive file included these files (in the same root directory):

- `./MANIFEST`
- `./modules/app.42m`
- `./modules/app.42r`
- `./forms/app.42f`
- `./xcf/app.xcf`

The directory structure after unpacking the Genero archive would be similar to this, with the current timestamp used:

- `<APPDATA>/deployment/myapp-20130522-123456/MANIFEST`
- `<APPDATA>/deployment/myapp-20130522-123456/modules/app.42m`
- `<APPDATA>/deployment/myapp-20130522-123456/modules/app.42r`
- `<APPDATA>/deployment/myapp-20130522-123456/forms/app.42f`
- `<APPDATA>/deployment/myapp-20130522-123456/xcf/app.xcf`

Genero Archive procedures

The procedures provide you with the instructions to create and use your Genero Archives when on the application server host.

> **Tip:** Use the Genero Archive deployment service, see Genero Archive deployment service on page 232, when working from a remote server.

Create a Genero Archive

A Genero Archive is a zip archive containing a MANIFEST file providing installation instructions and the list of application and services to make available.

As a prerequisite, all of your application files, along with your application configuration (xcf) file, must sit under one directory. For example, this would be a valid organization of files:

```
./fuzzy/modules/app.42m
./fuzzy/modules/app.42r
./fuzzy/forms/app.42f
./fuzzy/xcf/app.xcf
./fuzzy/appPublicImages/
```

> **Note:** Public images need to go in a dedicated directory, for more information on deploying images see Application deployment overview on page 218.

Follow these steps to create a Genero Archive from the contents of a directory. All folders and files in the directory are included in the archive.

> **Note:** When you use the fglgar tool to create an archive, you are creating a zip file. The tool is designed to take a single directory as its parameter. If you need a more sophisticated archive tool

(to add only specific files to the archive, for example), you can use any other zip tool to create your GAR archive.

1. Create a MANIFEST file. See The MANIFEST file on page 224.
2. Update the application's configuration file (`xcf`).

   All resources must be set relative to the resource `res.deployment.path`. For example, if your compiled files were in the `/bin` directory of your archive, you would update `<PATH>` to:

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

   Complete this change for all resources used in the configuration file: pictures, templates, forms, modules, and so on.
3. Use the `fglgar` tool to create a Genero Archive.

   If you are in the directory containing your MANIFEST file and your program files:

   ```
   fglgar --gar
   ```

   This creates an archive (`gar`) file with the same name as the archive directory.

   If you need to specify the directory where the archive content is located, include the `--input-source` option:

   ```
   fglgar --gar --input-source ./fuzzy
   ```

   This creates an archive file with the same name as your program, drawing its content from the `./fuzzy` directory.

   If you wish to specify a name for your archive, use the --output option:

   ```
   fglgar --gar --input-source ./fuzzy --output myfuzzy.gar
   ```

   This creates an archive file with the name `myfuzzy.gar`, drawing its content from the `./fuzzy` directory.

   Note: The archive name has no importance.

Deploy an archive

When you deploy an archive on a Genero Application Server, the applications and services referenced in the archive are placed on the host, yet are not yet available to users.

As a prerequisite, you have a Genero Archive file.

The archive will be unpacked in the deployment directory. Static resources are zipped for better performance at runtime.

`gasadmin` will deploy the .gar but the application deployed is named with the .xcf.

Enter the `gasadmin` command with the appropriate arguments.

To deploy an archive named `fuzzy.gar`:

```
gasadmin --deploy-archive fuzzy.gar
```

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
gasadmin --deploy-archive fuzzy.gar --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

XML output example:

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Genero Archive successfully deployed.</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to deploy the Genero Archive</ERROR>
  <ERROR>An archive with the same name is already deployed</ERROR>
</DEPLOYMENT>
```

Text output example:

```
Command succeeded.
Genero archive successfully deployed.
```

```
Command failed.
ERROR: Failed to deploy the Genero Archive
ERROR: An archive with the same name is already deployed.
```

All applications and services are deployed in the default group of the application server.

List all deployed archives

You can list all deployed archives with a single command.

Enter the `gasadmin` command with the appropriate arguments.

To list all deployed archives:

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

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
gasadmin --list-archives --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

An example of text output for an archive listing:

```
Command succeeded.

2 archives deployed:

  Name : archive1
  State : Disabled
  Description : This my incredible first archive
  Application(s):
      - 22223.xcf
      - 22224.xcf
  Service(s) :
      - 15233.xcf

  Name : archive2
  State : Disabled
  Description : This my incredible second archive
  Application(s):
      - 22315.xcf
  Service(s) :
```

```
      - 17746.xcf
      - 19164.xcf
```

An example of XML output for an archive listing:

```
<DEPLOYMENT success=”TRUE”>
  <ARCHIVE name=”MyArchive” enabled=”TRUE”>
    <DESCRIPTION> This is my first achive </DESCRIPTION>
    <APPLICATION xcf=”MyApp1.xcf” />
    <APPLICATION xcf=”MyApp2.xcf” />
    <SERVICE xcf=”MyServ1.xcf” />
    <SERVICE xcf=”MyServ2.xcf” />
  </ARCHIVE>
  <ARCHIVE name=”MyArchive2” enabled=”TRUE”>
    <DESCRIPTION> This is my second archive </DESCRIPTION>
    <APPLICATION xcf=”app/MyApp3.xcf” />
    <APPLICATION xcf=”app/MyApp4.xcf” />
  </ARCHIVE>
</DEPLOYMENT>
```

Note: The `DEPLOYMENT` node may contains some `MESSAGE` or `ERROR` nodes.

Activate (enable) a deployed archive

A deployed archive is not implicitly activated. When you activate (enable) a deployed archive, the applications and services provided by the archive become available to users.

Enabling an archive means enabling all applications and services provided by the Genero Archive.

1. Enter the `gasadmin` command with the appropriate arguments.

   To show a list of archives and details of their status:

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

   To enable an archive named `fuzzy`:

   ```
   gasadmin --enable-archive fuzzy
   ```

   By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

   ```
   gasadmin --enable-archive fuzzy --xml-output
   ```

   The exit status is a 0 (zero) in case of success, 1 in case of error.
2. Copy the application configuration file (`xcf`) into a group directory, if desired.

XML Output examples:

```
<DEPLOYMENT success=”TRUE”>
  <MESSAGE>Archive <archive-name> successfully enabled</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success=”FALSE”>
  <ERROR>Failed to enable <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure....</
ERROR>
</DEPLOYMENT>
```

Deactivate (disable) a deployed archive

When you deactivate (disable) a deployed archive, the applications and services provided by the archive are not exposed (no longer available to users).

As a prerequisite, you have a deployed archive that has been enabled.

Disabling an archive means disabling all applications and services provided by the Genero Archive. The archive remains deployed. Any running applications or services are not stopped.

> **Tip:** To disable a single application or service, remove its configuration file from the group directory. The original configuration files remain available in the deployed archive directory.

Enter the `gasadmin` command with the appropriate arguments.

To disable a Genero Archive named `fuzzy.gar`:

```
gasadmin --disable-archive fuzzy.gar
```

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
gasadmin --disable-archive fuzzy.gar --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

XML output examples:

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully undeployed</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to undeploy <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure....</
ERROR>
</DEPLOYMENT>
```

Undeploy a deployed archive

When you undeploy an archive, the archive is no longer deployed and cannot be reactivated.

An archive can be undeployed. When you undeploy an archive:

- All applications and services provided by the archive that are not disabled are disabled at this time.
- The `MANIFEST` file renames to `MANIFEST.undeployed`. Any archive with the file `MANIFEST.undeployed` represents an undeployed archive.

When you undeploy an archive, no files are removed. The cleanup of the undeployed archive directory must be explicitly requested.

Enter the `gasadmin` command with the appropriate arguments.

To undeploy an archive named `fuzzy.gar`:

```
gasadmin --undeploy-archive fuzzy.gar
```

By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

```
gasadmin --undeploy-archive fuzzy.gar --xml-output
```

The exit status is a 0 (zero) in case of success, 1 in case of error.

XML output examples:

```
<DEPLOYMENT success="TRUE">
  <MESSAGE>Archive <archive-name> successfully undeployed</MESSAGE>
</DEPLOYMENT>
```

```
<DEPLOYMENT success="FALSE">
  <ERROR>Failed to undeploy <archive-name> archive.</ERROR>
  <ERROR>... and error messages indicating the reasons of the failure...</
ERROR>
</DEPLOYMENT>
```

Text output examples:

```
Command suceeded.
Genero archive successfully deployed.
```

```
Command failed.
ERROR: Failed to undeploy the Genero Archive
ERROR: ... and error messages indicating the reasons of the failure...
```

Clean up undeployed archives

The deployment tool provides a cleanup command that physically removes undeployed archives. This process is executed on user request.

As a prerequisite ensure that applications you want to remove are not running any more. You can list the running applications with "gasadmin -l".

One or more archives have been undeployed.

The cleanup operation will only remove directories in the deployment directory if:

- The directory name matches a deployment directory name pattern, i.e. *archive-name-timestamp*
- The archive directory contains a file name `MANIFEST.undeployed`.

1. Enter the `gasadmin` command with the appropriate arguments.

   To clean up all undeployed archives:

   ```
   gasadmin --clean-archives
   ```

   For each undeployed archive, you are asked to confirm the cleanup by entering y to clean up, n to skip and continue to the next undeployed archive.

   By default, the results are output in text format. When the console output uses the form of an XML document, it can be easier to parse by other applications than textual output. To output in XML, specify the `--xml-output` option:

   ```
   gasadmin --clean-archives --xml-output
   ```

   To disable confirmation for each archive removal, add the `-all` option:

   ```
   gasadmin --clean-archives -all
   ```

   The exit status is a 0 (zero) in case of success, 1 in case of error.
2. To quit the clean up, enter `quit`.

Upgrade an archive

You can upgrade an application without having to kill processes or wait for users to log out.

As a prerequisite, your applications and services were previously deployed as an archive. You still have the requisite archive files (see Create a Genero Archive on page 226.)

1. Deploy the new version of the archive. See Deploy an archive on page 227.
2. Disable the previously deployed archive. See Deactivate (disable) a deployed archive on page 230.
3. Enable the newly deployed archive. See Activate (enable) a deployed archive on page 229.
4. Undeploy the previously deployed archive. See Undeploy a deployed archive on page 230.

Genero Archive deployment service

A deployment service has been written in Genero and can be accessible on a base URL of `/ws/r/services/DeploymentService`. It provides several actions in the form of REST APIs.

The deployment service is a Web service provided for the Genero Application Server. To prevent this application being accessed remotely, the default configuration restricts access to the localhost (127.0.0.1). If you want to enable it for other client machines / IP addresses, you must customize the `ALLOW_FROM` tag or remove the `ACCESS_CONTROL` tag. The service configuration can be found at `$FGLDIR/web_utilities/services/DeploymentService.xcf`.

Table 21: Actions available for the Genero Archive deployment service

Action	Description	HTTP request details
deploy	Deploys a Genero Archive on the Genero Application Server by calling `gasadmin --deploy-archive`. Returns the command result in XML format.	HTTP PUT request of following form: URL: /ws/r/services/DeploymentService/deploy?archive=name **Note:** archive=name is mandatory to identify the archive on the GAS. To send the archive to the Genero Application Server, you could use curl with a PUT instruction: `curl -X PUT -T test.gar http://app_server:port/connector/ws/r/services/DeploymentService/deploy?archive=name` where: • *app_server* is the server name or IP address. • *port* is the port where the application server or Web server is listening. • *connector* is the Genero Application Server connector (`/gas`, for example).
enable	Enables all applications and services of a given archive by calling `gasadmin --enable-archive`. Returns the command result in XML format.	HTTP GET request of following form: URL: /ws/r/services/DeploymentService/enable?archive=name

Action	Description	HTTP request details
disable	Disables all applications and services of given archive by calling `gasadmin --disable-archive`. Returns the command result in XML format.	HTTP GET request of following form: URL: /ws/r/services/DeploymentService/disable?archive=name
undeploy	Undeploys a genero archive on the Genero Application Server by calling `gasadmin --undeploy-archive`.	HTTP GET request of following form: URL: /ws/r/services/DeploymentService/undeploy?archive=name
list	Returns a list and status of all archives available on the Genero Application Server by calling `gasadmin --list-archive`. Returns the command result in XML format.	HTTP GET request of following form: URL: /ws/r/services/DeploymentService/list
cleanup	The cleanup operation is not available withthe REST service, as it requires user interaction to its prompts. To perform a cleanup operation, you must use `gasadmin` on the application server(performed by `gasadmin --clean-archives`).	N/A

Upgrading

These topics talk about what steps you need to take to upgrade to the next release of Genero Application Server, and allows you to identify which features were added for a specific version.

Review the list of migration recommendations each time you move to a new version of the Genero Application Server. Failure to do so can result in issues when rendering your Web applications.

New Features of the Genero Application Server

These topics provide an look back at the new features introduced with each release of the Genero Application Server.

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

This topic includes information about new features added for 3.00 Maintenance Releases (MRs) of the GAS and changes in existing functionality.

> **Important:** Please read What's new in Genero Application Server, v 3.00 on page 235, for a list of features that were introduced with Genero 3.00 General Availability release.

Table 22: Genero Web Client for JavaScript (GWC-JS), Version 3.00

Overview	Reference
Enhancements for the GWC-JS (v1.00.16): • Canvas elements are now supported. • Front calls *setvar* and *getvar* are supported for session variable management. See the *Genero Business Development Language User Guide* for more details about their usage.	See Features and limitations on page 176.

Table 23: Engine and Architecture, Version 3.00 (Maintenance Releases)

Overview	Reference
The `GWC_JS_LOOKUP_PATH` element (added as a child of `INTERFACE_TO_CONNECTOR` on page 324) allows you to configure the location of your custom GWC-JS front end.	See GWC_JS_LOOKUP_PATH on page 320

Overview	Reference
A new URI dedicated to the lookup of the GWC-JS directory. The complete format of the URI is `ua/w/$(GWC-JS)/<filename>`.	See Application URIs on page 49

Table 24: Deployment, Version 3.00 (Maintenance Releases)

Overview	Reference
The `WEB_COMPONENT_DIRECTORY` element allows for multiple paths to be specified.	See WEB_COMPONENT_DIRECTORY on page 361

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

What's new in Genero Application Server, v 3.00

This topic includes information about new features and changes in existing functionality.

Table 25: GWC for HTML5, Version 3.00

Overview	Reference
Genero Web Client for JavaScript (GWC-JS) is a new Web client for developing Genero Web Client applications.	See Genero Web Client for JavaScript (GWC-JS) on page 173 .

Table 26: Single Sign-On, Version 3.00

Overview	Reference
The Single sign-on (SSO) mechanism now works for all clients: Genero Desktop Client (GDC) and Genero Web Client (GWC).	See What is Single sign-on (SSO)? on page 37.
OpenID Connect is introduced as SSO protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under *$FGLDIR/* `web_utilities/services/openidconnect`.	See OpenID Connect SSO on page 115

Table 27: Web Services and the GAS, Version 3.00

Overview	Reference
A new element called *REQUEST_RESULT* has been added to the Web services time out component, which, if set, allows the GWS proxy to release the DVM in charge of a service that has not responded within a given time frame (seconds).	See SERVICE_APPLICATION_TIMEOUT_COMPONENT on page 345

Table 28: Engine and Architecture, Version 3.00

Overview	Reference
The `gasadmin` command has been updated with new features to manage archives. You can now list, enable, deploy and undeploy archives with options of the `gasadmin` command.	See The gasadmin command on page 269.
The GAS supports a new timeout feature called AUTO_LOGOUT_COMPONENT which can be defined in the timeout	See AUTO_LOGOUT_COMPONENT on page 305

Overview	Reference
component for GWC and GDC applications. If set, an application will get a logout page or screen after a specified time (in seconds) of user inactivity.	
GAS 3.00 introduces a new universal proxy for applications using GDC v3 and GWC-JS interfaces. It is called `uaproxy` (ua). It replaces the `gdcproxy` and `html5proxy` proxies. It provides protocol improvements and better performance overall.	See Proxy: uaproxy on page 266
There is a new bootstrap mechanism for starting GWC-JS applications, which is used to initialize information for rendering an application.	See Starting GWC-JS applications on page 188
A new element called `GWC_JS_LOOKUP_PATH` added to `INTERFACE_TO_CONNECTOR` on page 324 allows you to configure the location of your custom GWC-JS front end.	See GWC_JS_LOOKUP_PATH on page 320
The new user agent protocol, the `uaproxy` for GDC, GMA, GMI and GWC-JS, introduces a new set of resource URLs. The *ua* protocol does not use snippets sets or output maps as the *wa* protocol did previously to specify output theme.	See Application URIs on page 49
Genero Ghost Client is a new Java framework introduced for testing different scenarios by emulating user interaction on Genero applications.	See Ghost Client and Testing Tools on page 272
GAS 3.00 introduces a new configuration entry for the Report Viewer which allows you to configure the location of the Genero Web Report Viewer. A corresponding report viewer URL prefix `/ua/grv` is provided to the Genero Report Engine (GRE).	See REPORT_VIEWER_DIRECTORY on page 340 See Application URIs on page 49.

Table 29: Deployment, Version 3.00

Overview	Reference
The deployment framework (`fglgar` tool) provides you with new resource management features for public image files.	See The fglgar command on page 267
Three new predefined resources have been added: • A public resource path for all applications, *`$(res.appdata.path)/public`* • A resource for common images used by applications, *$(res.public.resources)* • A resource where the Application Server stores files temporarily during file transfer, *$(res.path.tmp)*	See GAS directories on page 38 and Paths to application resources on page 219
A `WEB_COMPONENT_DIRECTORY` has been added, it contains the path(s) where Web components are located for an application. It replaces the `WEB_COMPONENT` element, which has been removed from the `EXECUTION` element of an application.	See WEB_COMPONENT_DIRECTORY on page 361
The `DOCUMENT_ROOT` now allows multiple document root paths.	See DOCUMENT_ROOT on page 312
New entries in the application MANIFEST file: • A new `RESOURCES` entry is added to the MANIFEST file that specifies the directory in your archive file where public images for your applications are found.	See The MANIFEST file on page 224 and TRIGGERS (for manifest) on page 224

Overview	Reference
• A new TRIGGERS element defines a set of deployment parameters that can be used when deploying an application with the deployment framework.	

Table 30: Miscellaneous, Version 3.00

Overview	Reference
The GAS can now be plugged in to Internet Protocol Version 6 (IPv6) Web servers without any additional configuration if the front-ends (GWC-JS, GDC, Web Service) need to use IPv6 user agents. As communication between the GAS and Web servers is on localhost, IPv4 continues to be used.	**Note:** If you have to restrict access of some applications to specific IP addresses, in that case the *ACCESS_CONTROL* entry must be configured with IPv6 addresses. See ACCESS_CONTROL on page 298
In addition to specifying access by IP address, the ACCESS_CONTROL and MONITOR configuration elements have been updated with two access control keywords (*NOBODY, ALL*) which can be used with ALLOW_FROM.	See ALLOW_FROM on page 300
Introducing a new resource, res.access.control, that specifies access control for Web services, application programs, such as demos, and MONITOR. It is defined with the keyword *NOBODY* by default.	**Note:** The default deployment does not allow access to demo programs and MONITOR. Access has to be configured explicitly with ALLOW_FROM. See MONITOR on page 328 and ACCESS_CONTROL on page 298
A new element called ROOT_URL_PREFIX added to INTERFACE_TO_CONNECTOR on page 324 supports the use of reverse proxy server between the client and the GAS. It allows for a URL prefix to be specified for the Web server so as to provide the correct interface to the client.	See ROOT_URL_PREFIX on page 344

Genero Application Server v 2.50 New Features

Features introduced with Genero Application Server 2.50.

Table 31: GWC for HTML5, Version 2.50

Overview	Reference
The HTML5 is now the default theme for all Genero Web Client applications. The AJAX, Silverlight, iPhone and Basic themes are deprecated.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Browser-based themes, in the Genero Application Server 2.50 User Guide.
HTML5 theme: while you can still customize your Genero Web Client application with CSS, the files involved have changed and any previous customization efforts will need to be revisited.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topics: • Cascading Style Sheets

Overview	Reference
	• HTML5 theme in the Genero Application Server 2.50 User Guide.
The HTML5 theme adds support for frozen columns, splitters (in forms, between the form and the actionPanel, between the form and the startMenu), and GridChildrenInParent.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see Html5 Theme topics in Genero Application Server 2.50 User Guide.
The html5proxy manages Genero Web Client applications using the HTML5 theme.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see Html5 Theme topics in the Genero Application Server 2.50 User Guide.
The HTML5 theme supports StartMenus and applications displayed in folder tabs.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Enable the StartMenu and applications in folder tabs, in the Genero Application Server 2.50 User Guide.
Add language support to a snippet file (HTML5).	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Translations in the snippets, in the Genero Application Server 2.50 User Guide.
Start an application while ignoring the application's stored settings (HTML5).	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Start an application without stored settings (GWC for HTML5), in the Genero Application Server 2.50 User Guide.
GDCAX is deprecated in favor of the HTML5 theme. GDC HTTP is still supported.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topics, Html5 Theme, in Genero Application Server 2.50 User Guide.
Control the folder size of an application rendered by the HTML5 client.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Control the folder size, in Genero Application Server 2.50 User Guide.
UI enhancements for the GWC (HMTL5 theme): • The tooltip now displays beside the field. An icon (triangle) allows you to show or hide the tooltip. You can also hide the tooltip by clicking on the message text. • The display of the calendar widget has been improved to prevent overlapping with other widgets. While it usually displays under the date field, it displays to the side if there is not enough space under.	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the topic, Features and limitations, in the Genero Application Server 2.50 User Guide.
The browser back and forward buttons can now trigger actions.	See Browser Back and Forward Buttons.
The demos application is restricted to localhost by default.	See Display demo applications with the Genero Web Client on page 25.

Table 32: Single Sign-On, Version 2.50

Overview	Reference
Kerberos is deprecated. Promoted SSO solutions are OpenID and SAML.	See What is Single sign-on (SSO)? on page 37.
Security Assertion Markup Language (SAML) is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under `$FGLDIR/web_utilities/services/saml`.	See SAML SSO on page 128.
OpenID is a Single sign-on (SSO) protocol supported by the Genero Application Server. It is based on a Genero REST service and is delivered in the Genero Web Services package under `$FGLDIR/web_utilities/services/openid`.	See OpenID SSO on page 120.
Delegate the start of a GWC (`/wa/r`) or GDC (`/ja/r`) application or a GWS (`/ws/r`) service to another REST service in order to perform some controls (such as authentication, authorization, or monitoring) in a single and centralized Genero program.	See How to implement delegation on page 105.

Table 33: Web Services and the GAS, Version 2.50

Overview	Reference
Support for sticky Web services.	See Sticky Web services on page 215.
Service invalidation provides notification when a Genero Web Service configuration is invalid. Once identified as having invalid configuration, the dispatcher will not attempt to start the gwsproxy for the service until the configuration is modified.	See Service invalidation on page 215.

Table 34: Engine and Architecture, Version 2.50

Overview	Reference
Better internal/temporary data organization.	See GAS directories on page 38.
The `gasadmin` command has been updated to handle multiple dispatchers. You can target the dispatcher with the -d option of the `gasadmin` command.	See The gasadmin command on page 269.
DVM standard error and standard output are now directed to dedicated log files.	See Logging on page 156.
The Genero Application Server on UNIX™ will use UNIX™ domain sockets to communicate between the dispatcher and the proxies.	See SOCKET_FAMILY on page 348 and SOCKET_PATH on page 349.
Enable and disable resource compression in the `imt.cfg` file. Compress resources (static files) using the -z option of the `gasadmin` command.	See Compression in Genero Application Server on page 146 and The gasadmin command on page 269.

Table 35: Deployment, Version 2.50

Overview	Reference
Genero Archive provides a means for packaging and deploying applications.	See Deploying with Genero Archive on page 217.

Table 36: Miscellaneous, Version 2.50

Overview	Reference
Genero Front Call ActiveX is deprecated.	No additional reference.

Genero Application Server 2.41 New Features

Features introduced with Genero Application Server 2.41.

Table 37: Genero Application Server Version 2.41 New Features

Overview	Reference
When using GWC for Silverlight, if the toolbar is larger than the form, the default behavior is to have a left and right button on the toolbar (horizontal scrolling).	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.41 User Guide.
GWC for HTML5 is a supported theme. It is no longer considered a "preview version". **Important:** The default theme for the next major release of Genero will be GWC for HTML5. Both the AJAX and iPhone themes will be deprecated in the next major release. If you are developing applications for the iPhone, it is recommended that you use the HTML5 theme, which is fully supported on iOS.	The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the topic, Html5 Theme, in the Genero Application Server 2.41 User Guide.
The Basic theme is deprecated. There will be no further development on the Basic theme. It is recommended that you use the HTML5 theme for future development.	No additional reference.
The GWC hybrid mode takes Genero applications and delivers them as native applications for Android or iOS mobile platforms.	The supporting topic has been removed, as GWC hybrid mode is deprecated. Please see the topic, Genero Web Client hybrid mode, in the Genero Application Server 2.41 User Guide.
New template paths: Application hierarchy, Window hierarchy, Layout hierarchies, and Widgets hierarchies have been added.	The supporting topics have been removed, as the template paths theme is deprecated. Please see the topics: • Template Paths - Application hierarchy • Template Paths - Window hierarchy • Template Paths - Layout hierarchies • Template Paths - Widgets hierarchies in the Genero Application Server 2.41 User Guide.

Genero Application Server 2.40 New Features

Features introduced with Genero Application Server 2.40.

Table 38: 2.40 New Features: Common

Overview	Reference
Support to add summary lines on TABLEs defining AGGREGATE form fields for the Genero Web Client.	Please see the topic, Features and limitations, in the Genero Application Server 2.40 User Guide.
The Genero Application Server can handle submit parameters with the POST method.	See Handling POST Method Submit Parameters.
Performance improvements on the GAS have been made to provide users with a better experience using Genero Web Applications. • Less memory consumed for GWC rendering • Less data traffic for tables and windows • Better stack management in threads	No additional reference.

Table 39: 2.40 New Features: Silverlight theme

Overview	Reference
The Silverlight theme layout is improved to closely match the layout proportions (widgets size, widgets redesign after window resizing,etc,) found in Genero Desktop Client.	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.
In the Silverlight theme, user preferences are saved in the stored settings and frozen columns for tables are supported	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.
In the Silverlight theme, most decoration common style attributes described in the *Genero Business Development Language User Guide* are now supported. Some decoration common style attributes not supported are: • border • localAccelerators • imageCache • showAcceleratorInToolTip	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.
In the Silverlight theme, rich text editing is now supported for `TextEdit` fields with `textFormat` html. An embedded toolbox with classic editing actions (bold, italic, underline, font size, etc.) is provided for text editing. **Text editing limitations compared to GDC / GWC for Ajax** • No list (ordered or unordered) • No text indentation • No local action binding • xHTML is mandatory • New lines in paragraph are replaced by spaces	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.

Overview	Reference
The Silverlight theme provides a user-friendly file download window.	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide.
The Silverlight theme provides support for type ahead inputs.	The supporting topic for Silverlight has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.40 User Guide. For generic information about the Type Ahead input mechanism, see Type Ahead.

Table 40: 2.40 New Features: HTML5 theme

Overview	Reference
Introduction to the HTML5 theme (preview version) for the Genero Web Client. The DUA_HTML5 output map is provided as a preview set. It is designed to run on modern desktop browsers as well as on mobile browsers. **Important:** The next major release of Genero will find the default theme changed to the HTML5 theme. The AJAX theme will be desupported in the subsequent release (the second major release after the current 2.40 release).	The supporting topics have been removed, as the HTML5 theme is deprecated. Please see the Genero Application Server 2.40 User Guide.

Genero Application Server 2.32 New Features

Features introduced with Genero Application Server 2.32.

Table 41: Genero Application Server Version 2.32 New Features

Overview	Reference
The Genero Application Server adminstration tool `gasadmin` is provided. This tool is a command to list, ping and kill applications sessions.	See The gasadmin command on page 269.
`MAX_REQUESTS_PER_DVM` is a new entry for GWS pool configuration. It allows a limited number of requests to be processed before the DVM is stopped.	See MAX_REQUESTS_PER_DVM on page 328.
A new log output allows logging both in a directory and to the console.	See LOG on page 326.

Genero Application Server 2.30 New Features

Features introduced with Genero Application Server 2.30.

Table 42: 2.30 New Features: New architecture

Overview	Reference
GAS 2.30 introduces dispatchers and proxies for more reliability, better performances and better integration in web servers.	See Overview and What is a dispatcher? on page 35

Overview	Reference
Java™ Application Server integration: The GAS server can now be fully integrated into any J2EE Servlet container using the GWC dispatcher j2eedispatch.	See Java Servlet Installation and Web Server Configuration on page 91.
New monitoring information: The monitor has been revised to display information that is relevant to the new architecture. In addition to application-specific information, there are also statistics shown for the dispatchers and proxies.	See Monitoring on page 150.
New LOG system: The log format and categories have been adapted to the new architecture with log files created for each dispatcher, proxy, and DVM started.	See LOG on page 326.
The web services pool management has been enhanced to explicitly limit the number of DVMs that can be started for a specific Web service. The Timeout configuration has changed to prevent the GWS proxy and DVMs from running indefinitely if the GAS dispatcher or the web server crashes.	See Services Pool (GWS Only) on page 31

Table 43: 2.30 New Features: GWC themes

Overview	Reference
GWC for Silverlight is based on Microsoft™ Silverlight technology. It is a fully-implemented Genero Front-end, like the Genero Desktop Client, with the ability to be customized, like GWC for AJAX.	The supporting topic has been removed, as the Silverlight theme is deprecated. Please see the Genero Application Server 2.30 User Guide.
GWC for iPhone provides a dedicated Genero Web Client template and set of snippets to render Genero applications as close as possible to native iPhone applications.	The supporting topic has been removed, as the iPhone theme is deprecated. Please see the Genero Application Server 2.30 User Guide.
GWC for AJAX provides support for iPhone and iPad web browsers. With Safari on iPhone providing the interface for all web content on iPhone, most of the feature set of a desktop browser is made available to mobile users. When it comes to displaying GWC applications on the iPhone or iPod Touch, the AJAX mode is fully functional with Safari on iPhone.	The supporting topic has been removed, as the AJAX theme is deprecated. Please see the Genero Application Server 2.30 User Guide.

Table 44: 2.30 New Features: Miscellaneous

Overview	Reference
GWC for AJAX and GWC for Silverlight support drag-and-drop.	The supporting topics have been removed, as the AJAX and Silverlight themes are deprecated. Please see the Genero Application Server 2.30 User Guide.
GAS provides a URI to launch GDC applications without having to configure a shortcut in the GDC monitor. Shortcuts can be exported as `.gdc`. GAS delivers these shortcuts through a URI for any application.	See URI Examples on page 55

Overview	Reference
Note: • On Mac, you need to manually add the .gdc association for Safari or save the .gdc and double click to launch the application • Currently, any access to /da/r url is handled by DUA_GDC map, no matter which OutputMap argument is set in the url	
The ACCESS_CONTROL element specifies which IP addresses are allowed to execute an application or web service. By default, all IP addresses are allowed.	See ACCESS_CONTROL on page 298.
The new resource `configuration.filepath` provides the absolute path to the GAS configuration file.	See Common GAS predefined resources on page 288.
New topics have been introduced to the documentation about each of the following themes (snippet sets): • AJAX theme • Silverlight theme • Basic theme • iPhone theme	The supporting topics have been removed, as the AJAX, Silverlight, Basic and iPhone themes are deprecated. Please see the Genero Application Server 2.30 User Guide.
New topics have been introduced to the documentation about ISAPI, FastCGI, and JAVA GAS dispatcher configuration, installation and integration to their dedicated Web servers. New topic added explaining GDC to GWC migration.	See • ISAPI Installation • FastCGI Installation • Java™ Servlet Installation • The supporting topics, as the GWC HTMLv1 theme is deprecated. Please see the GDC to GWC migration topic in the Genero Application Server 2.30 User Guide.

Genero Application Server 2.22 New Features

Features introduced with Genero Application Server 2.22.

Table 45: 2.22 New Features

Overview	Reference
Applications started by a StartMenu can be rendered in folder tabs.	See Enable the StartMenu and applications in folder tabs.
Textedit with textFormat html now displays a toolbox with classic editing actions (bold, italic, fontsize, and so on). Local actions are also created so you can create your own action views (global toolbar, and so on).	See Rich text editing.

Genero Application Server 2.21 New Features

Features introduced with Genero Application Server 2.21.

Table 46: 2.21 New Features: Backwards compatibility

Overview	Reference
GAS 2.21 is compatible with the Genero 2.11 product suite (FGL, GWS).	No additional reference.

Table 47: 2.21 New Features: Genero Web Client New Features

Overview	Reference
The layout of the AJAX output driver has been enhanced in several ways.	The supporting topic has been removed, as the AJAX theme is desupported. Please see the Genero Application Server 2.21 User Guide.
Picture deployment enhancements • The list of file system paths searched for images can be configured according to the user agent type. • Pictures that are not found on the GAS file system are requested from the DVM. The ResourceURI has been enhanced to support these requests.	The supporting topic has been removed, as the template path theme is desupported. Please see the Genero Application Server 2.21 User Guide.
Internet Explorer 8 is now supported.	The supporting topic has been removed as GWC for HTML5v1 is deprecated. Please see the topic, Features / browsers supported, in the Genero Application Server 2.21 User Guide.
The authenticated user and the remote IP address from the client is accessible to the Genero application through environment variables.	See Application environment on page 42
The list of supported snippet-based rendering engine (SBRE) template paths is accessible through a specific URL.	The supporting topic has been removed, as the template path theme is desupported. Please see the topic, Template Paths Overview, in the Genero Application Server 2.21 User Guide.
The Template Element Identifiers feature provides a means to customize the template element identifier name used to reference elements during the incremental update of the page, according to the target markup language.	The supporting topic has been removed, as the template path theme is desupported. Please see the topic,Template Element Identifiers, in the Genero Application Server 2.21 User Guide.
The `application/ui/decimalSeparator` path gets access to the currently active decimal separator character.	The supporting topic has been removed, as the template path theme is desupported. Please see the topic,Template Paths - Application hierarchy, in the Genero Application Server 2.21 User Guide.
The `SpinEdit/minValue` and `SpinEdit/maxValue` paths get access to the corresponding SpinEdit attributes.	The supporting topic has been removed, as the template path theme is desupported. Please see the topics: • SpinEdit/minValue • SpinEdit/maxValue in the Genero Application Server 2.21 User Guide.
Applications in configuration files support both a short description and a long description.	See DESCRIPTION on page 311

Genero Application Server 2.20 New Features

Features introduced with Genero Application Server 2.20.

> **Note:** Links to outdated documentation pages have been removed. If you are using version 2.20, you may wish to view the documentation created for version 2.20.

Table 48: 2.20 New Features: Application Server

Overview	Reference
Multi-threaded capabilities adapt the Genero Application Server daemon (**gasd**) capabilities to the Application Server demand. One **gasd** process will be able to adapt itself to the number of pending requests, providing better efficiency.	The architecture of the Genero Application Server has since changed. See Architecture of the Genero Application Server on page 27.
The following improvements have been made in GAS 2.20: • New plug-able components (GAS Isapi and GAS FastCGI) are available for each targeted application server, providing a new connection architecture that adapts the external world model to the GAS model. • A new generic resourceURI() function associated to the new PATH element definition allows you to specify multiple resource locations accessible from the browser on an application-level basis.	The architecture of the Genero Application Server has since changed. See Architecture of the Genero Application Server on page 27.
The GAS gives access to server and applications details through the "/monitor" URL. The monitor page exposes details about DVMs, Web applications, Web services, GDC applications, and GAS activity.	See Monitoring on page 150.

Table 49: 2.20 New Features: GDCAX management

Overview	Reference
Protocol between GDC and the application server is no longer character-oriented; it is now binary-oriented. As such, the connection is more efficient. They now communicate on a unique URL, which improves the authenticated and secured connections.	The supporting topic has been removed, as the Genero Desktop Client ActiveX feature is deprecated. Please see the Genero Application Server 2.20 User Guide.

Table 50: 2.20 New Features: Web application management (Genero Web Client)

Overview	Reference
Tree views are fully supported by the Genero Web Client.	The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of limitations of what the HTML5 theme supports .
The new **XUL** snippet set offers a desktop-like rendering. The XUL snippet set will be activated if you use a Prism user agent. You can also activate XUL snippet sets in any Firefox browser by using the DUA_XUL output map.	The supporting topic has been removed, as the XUL theme is deprecated. Please see the Genero Application Server 2.20 User Guide.

Overview	Reference
A new GAS 2.20 framework allows JavaScript™ functions to handle the BDL frontCall function on the browser. BDL 2.20 also introduces two new frontcall functions (*launchUrl* and *feinfo/screenresolution*) which are fully supported by the GWC.	The *frontcall* supporting topic has been removed, as the HTML5 theme is deprecated. Please see the JavaScript topic in the Genero Application Server 2.20 User Guide.
Modern browsers fully support SVG and SWF. Image widget snippets create the correct HTML code so the browser will render images with the extension `.svg` or `.swf`.	The supporting topic has been removed, as the HTML5v1 theme is deprecated. Please see the Genero Application Server 2.20 User Guide.
Select multiple rows using the usual key and mouse combination to enhance your Display Array.	The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of limitations of what the HTML5 theme supports .
Enable Sorting in Input Array is now possible.	The supporting topic has been removed, as the HTML5 theme is deprecated. Please see the Limitations topic in the Genero Application Server 2.20 User Guide for a complete listing of limitations of what the HTML5 theme supports .
Hyperlinks in Label / TextEdit are now supported.	See Display a widget as a hyperlink
Button type <<link>>: A classic Genero BDL button, but displayed as an hyperlink.	No additional reference.

Upgrade Guides for the Genero Application Server

Each upgrade guide is an incremental upgrade guide that covers only topics related to a specific version of Genero. It is important that you read all of the upgrade guides that sit between your existing version and the desired version.

- GAS 3.00 upgrade guide on page 248
- GAS 2.50 upgrade guide on page 250
- GAS 2.41 upgrade guide on page 251
- GAS 2.40 upgrade guide on page 251
- GAS 2.30 upgrade guide on page 253
- GAS 2.22 upgrade guide on page 256
- GAS 2.21 upgrade guide on page 256
- GAS 2.20 upgrade guide on page 256
- Upgrading from GAS 2.10.x or GWC 2.10.x on page 258
- GAS (GWC) 2.10 upgrade guide on page 260
- GAS 2.00 upgrade guide on page 261

GAS 3.00 upgrade guide

Complete these tasks when migrating to Genero Application Server 3.00 from version 2.50. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Genero Web Client

GWC for JavaScript will be the default rendering for Genero Web Client applications, replacing GWC for HTML5. New development should use GWC for JavaScript.

Genero Desktop Client ActiveX (GDCAX) is desupported

For new development, use GWC for JavaScript.

Single Sign-On Authentication

The Kerberos authentication mechanism for Single Sign-on authentication is deprecated. Any new development requiring Single Sign-on should plan to use OpenID Connect, SAML or OpenID. See How to implement Single sign-on (SSO) on page 114. For alternative solutions, please contact your Four Js support center.

Java Dispatcher

For Genero 3.0, the Java Dispatcher, see Dispatcher: java-j2eedispatch on page 266, requires at least version 3.0 of the Java servlet API. This is supported on the following Java web servers:

- IBM® WebSphere® (since version 8.0), see http://en.wikipedia.org/wiki/IBM_WebSphere_Application_Server
- Tomcat (since version 7.0.x), see http://tomcat.apache.org/whichversion.html
- Glassfish (since version 3.0), see https://glassfish.java.net/public/comparing_v2_and_v3.html

Web Services

For Genero 3.0, the following FGLGWS connections and web services are supported:

- **FGLGWS 3.00**

Protocols	FGLGWS 3.00
ua	GDC 3.00, GWC-JS, GMA, GMI
ja	n/a
wa	n/a

- **FGLGWS 2.50**

Protocols	FGLGWS 2.50
ua	n/a
ja	GDC 2.50
wa	GWC-HTML5

Web Server side Resources

In GAS 2.50, using `gwcproxy` and `html5proxy` proxies, the path to the image directory is configured to fetch resources on the Web server side in the `PATH` element, see PATH with Type WEBSERVER on page 333.

```
<PICTURE>
    <PATH type="WEBSERVER">a_uri</PATH>
</PICTURE>
```

This is usually configured to improve performance. The Web server delivers static files or images instead of the GAS.

With GAS 3.00, for `uaproxy`, all the resources are delivered by the GAS. By default the application's public directory is defined by the *PUBLIC_IMAGEPATH* in *$FGLASDIR*`/etc/as.xcf` configuration file:

```
<PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
```

> **Note:** *$(res.public.resources)* defaults to the path *appdata*`/public/common`, see PUBLIC_IMAGEPATH on page 338.

If you use Genero Archive you can specify public resources for your applications by adding a *RESOURCES* element in the Genero Archive manifest. Then the resources are copied in *$FGLASDIR/appdata/*`public/deployment/deployappname`, where "deployappname" is the name given to the deployed application directory by the Genero Archive. See Application deployment overview on page 218.

Web Components

In GAS 2.50, web components are deployed under *$FGLASDIR*`/web/components` directory, see WEB_COMPONENT_DIRECTORY on page 361.

Starting from GAS 3.00, with `uaproxy`, the default path for a web component is *appdir/*`webcomponents`, where "appdir" is the application directory. See the *WEB_COMPONENT_DIRECTORY* element in your *$FGLASDIR*`/etc/as.xcf` configuration file:

```
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
   […]
   <DVM>$(res.dvm.wa)</DVM>
   <WEB_COMPONENT_DIRECTORY>$(application.path)/webcomponents</
WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

You can change the default web components location by configuring a *WEB_COMPONENT_DIRECTORY* element in your application's configuration. In this example, the web component is no longer located in *appdir/*`webcomponents` but in *appdir/*`mycomponents`.

```
<APPLICATION Parent="defaultgwc" …>
  <EXECUTION>
    <PATH>/home/myapp</PATH>
    <MODULE>myapp</MODULE>
   <WEB_COMPONENT_DIRECTORY>/home/myapp/mycomponents</
WEB_COMPONENT_DIRECTORY>
  </EXECUTION>
</APPLICATION>
```

For more details on Web component usage, see the *Genero Business Development Language User Guide*.

Genero Web Client hybrid mode (deprecated)

The GWC hybrid mode feature has been deprecated. Applications for Android or iOS mobile platforms which used the GWC hybrid mode will now need to use the more-featured GMA or GMI instead. If you do not have equivalent features in GM, contact your local Four Js support center.

GWC-JS LOOKUP PATH

The GWC_JS_LOOKUP_PATH on page 320 is a new mandatory element in `as.xcf`. If you are upgrading to GAS versions 3.00.11 or later and wish to keep your existing `as.xcf` configuration file, you need to add this element to the INTERFACE_TO_CONNECTOR on page 324 element. Otherwise, you will see the following error message at GAS startup:

```
Application Server startup . ........................................
 httpdispatch
"Configuration ERROR" Code:1871 Message:Element 'TEMPORARY_DIRECTORY': This
 element is not expected. Expected is ( GWC_JS_LOOKUP_PATH ).
[fail]  httpdispatch "Main Configuration"
Cannot build main configuration
```

To find out your GAS version, at the command line run the `gasadmin -V` command. For more information on the `gasadmin` tool, see The gasadmin command on page 269 topic.

GAS 2.50 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.50 from version 2.40. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

- GWC for HTML5 will be the default rendering for Genero Web Client applications, replacing GWC for AJAX. Any new development should plan to use GWC for HTML5.
- Customization using CSS has changed to provide a flat design to fit all platforms (desktop, tablet, and smartphone) using the HTML5 theme. All CSS customization is now contained within `css_customization.css`. This should ease both customization and migration between versions of the Genero Application Server. As the HTML5 theme is deprecated, please see the Cascading Style Sheets and HTML5 theme topics in the Genero Application Server 2.50 User Guide.
- The Genero Administration Application (GAD) is no longer supported.
- The `ForwardDVMStdout` attribute for the `MAP` element is no longer permitted in the configuration file, as the DVM output and errors are now logged to a separate log file. Remove this attribute from your Genero Application Server configuration file.
- The `CONNECTOR_PREFIX` element is no longer valid in the Genero Application Server configuration file. Remove it from your configuration file, if present.
- The behavior of a simple RUN command has changed.

  Previously, when an child application was launched using a simple RUN command, the child application replaced the main application in the same browser window. Only the child application was visible. When you exited the child application, the main application resumed. Throughout, there was a single browser window.

  With version 2.50, an application launched by a simple `RUN` command opens a second window (or browser) containing the child application, as it also does for `RUN WITHOUT WAITING`. The child application runs normally. The main application is frozen, waiting for the child application to terminate. If you attempt to leave or close the main application when a child application is still running, a prompt asks you to confirm that you want to leave.
- Genero Front Call ActiveX is now deprecated.
- Logging for the Genero Application Server has changed. Logs are now created for the dispatchers, the proxies, and the virtual machines. DVM logs are redirected to files when DAILYFILE is set for the log output type.

- The default location of the Genero Application Server log files has moved. If you set logrotate or another tool to manage your GAS log files, you will need to reset. See GAS directories on page 38 for details on the location of log files.
- Some Windows™ directories have restricted permissions. Depending on where the Genero Application Server is installed, you might not have access to log, tmp and sessions directories. These directories are now located at `C:\ProgramData\FourJs\gas\gas_version` where `gas_version` is the version of the Genero Application Server. You can configure their location by modifying "res.appdata.path" in the Genero Application Server configuration file.

GAS 2.41 upgrade guide

There are no migration tasks specific for the Genero Application Server 2.41 release.

While there are no migration tasks specific for the Genero Application Server 2.41 release, you should be aware that starting with the next major release (tentatively 2.50):

- GWC for HTML5 will be the default rendering for Genero Web Client applications, replacing GWC for AJAX. Any new development should plan to use GWC for HTML5.
- The Genero Administration Application (GAD) will be de-supported.

GAS 2.40 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.40 from version 2.30. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Template and snippets

Namespaces

To optimize template and snippets rendering, all namespaces need to be declared in the main template. You need to move all the namespaces used in your snippets to the main template. The changes have to be applied on all snippet sets.

Example in 2.3x:

$FGLASDIR/tpl/SetAjax/main.xhtml

```
<html xmlns:gwc="http://www.4js.com/GWC" xmlns="http://www.w3.org/1999/
xhtml"
 ...>
```

$FGLASDIR/tpl/common/Canvas.xhtml

```
<svg:svg
   xmlns:svg="http://www.w3.org/2000/svg"
    viewBox="0 0 1000 1000"
    preserveAspectRatio="none">
  <svg:rect x="0" y="0" width="1000" height="1000" fill="lightgray"
 stroke="black"/>
  <span gwc:repeat="i items" gwc:replace="i" />
</svg:svg>
```

Example in 2.40:

$FGLASDIR/tpl/SetAjax/main.xhtml

```
<html xmlns:gwc="http://www.4js.com/GWC" xmlns="http://www.w3.org/1999/
xhtml"
```

```
 xmlns:svg="http://www.w3.org/2000/svg" ...>
```

GWC for Silverlight behaviors

Starting with GAS 2.40, GWC for Silverlight follows the Model-View-ViewModel (MVVM) pattern to enable even more designers to customize the view of their applications with tools like Microsoft™ Expression Blend.

This change leads to the replacement of the GWC.Behaviors module by a view models layer. Therefore all the XAML markup that used the bhv prefix needs to be updated. Here is an example of the migration of an action view.

In 2.32:

```
<sr:Button
 bhv:Action.Observer="{action && action/isActive ?
 [action/IDID, 'Activate'] : null}"
  bhv:Media.URI="['{resourceURI("images/close.png",
 "SetSL")}', 'SmallImage']"
   IsEnabled="{action/isActive || false}"
   IsTabStop="False" />
```

In 2.40:

```
<sr:Button
 Command="{{Binding ClickCommand,Mode=OneTime}"
  SmallImage="{{Binding Image.Source}"
  IsEnabled="{action/isActive || false}"
  IsTabStop="False">
 <sr:Button.DataContext>
  <vm:ActionView Image="{resourceURI('images/close.png','SetSL')}">
   <vm:ActionView.ServerActions
 gwc:condition="action && action/isActive">
    <vmsa:Action ServerID="{action/IDID}"
 IsEnabled="True" Event="Click" />
   </vm:ActionView.ServerActions>
  </vm:ActionView>
 </sr:Button.DataContext>
</sr:Button>
```

The bhv:Action.Observer and bhv:Media.URI behaviors have been replaced by a more conventional mechanism based on Data bindings that gets their properties from the ActionView view model.

GWC for Silverlight template snippet splitting

To ease the customization, some template snippets have been splitted into more parts:

- The user interface part of the main.xaml template has been put into the UserInterface.xaml snippet.
- The ending part of the main.xaml template has been put into the EndingPage.xaml snippet.
- The toolbar and the top menu parts of the WindowContent.xaml snippet have been put into the ToolMenu.xaml snippet.

Template paths

application/interrupt/did and **application/interrupt/xdid** template paths are replaced by **application/interrupt/url**. For more details on the new template path usage see the main templates.

GAS 2.30 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.30 from version 2.22. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Starting the GAS

The gasd command no longer exists, it is replaced by dispatchers. To run the GAS in standalone mode use httpdispatch command. On windows, from the start menu, there is a shortcut to start the dispatcher in standalone mode.

Configuration

Starting with version 2.30, the configuration file has been simplified. Some configuration parameters are no longer needed due to the new architecture, and others are now handled transparently (such as the web services pool and load).

--development

To start a GWC application in development mode, you must add the option --development to the gwcproxy entry in the appropriate xcf file.

Example:

```
<RESOURCE Id="res.gwcproxy.param" Source="INTERNAL">--development</RESOURCE>
```

This resource *res.gwcproxy.param* is used as parameter when launching the GWC proxy

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

Configuration file (xcf) inheritance

Prior to version 2.30, when an application was started with a RUN, the child application inherited the configuration of the parent *unless the child application has its own configuration file*. If the child application had its own configuration file (where the .xcf file shares the same name as the child application), then the child configuration took priority over the parent configuration, and was used for the child application.

Starting with version 2.30, this is no longer true. A 4gl application started with RUN or RUN WITHOUT WAITING will inherit the configuration of the parent application, and this cannot be changed. The configuration used to start the first application (the first FGLRUN) will be used for all child applications (child FGLRUNs).

While you might want to review your parent/child applcations, this will likely not have an impact, as a customer survey determined that most had only provided a configuration file for the parent application and had not provided a configuration file for the child applications.

Resource of type FILE

Starting with version 2.30, resources of type FILE are no longer supported.

Example:

```
<RESOURCE Id="res.theme.default.gdc.template" Source="FILE">
 $(res.path.tpl)/fglgdcdefault.html
</RESOURCE>
```

ALIAS entry removed

Starting with version 2.30, the entry ALIAS has been removed. You can configure PICTURE element instead.

THREAD_POOL entry removed

Starting with version 2.30, the entry THREAD_POOL has been removed.

LOG output of type PATH de-supported

Starting with version 2.30, you cannot set a log output of type PATH.

REQUEST_QUEUE and REQUEST_RESULT for Web Services are removed

Starting with version 2.30, Web Service REQUEST_QUEUE and REQUEST_RESULT have been removed.

Instead of these timeouts, the web server one is used. For example, the fastcgi *idle timeout*.

DVM_FREE removed

Starting with version 2.30, Web Service DVM_FREE has been removed.

It is no more needed as the Web Service pool management uses statistics of previous requests to decide whether to stop a DVM or not.

Hot Restart no longer necessary

There is no longer a need for a hot restart, as changes in an external XCF file are immediately taken into account at:

- Application startup for a web application
- GWS DVM startup for web services that could be at each new request

As a result, you should check the changes you have made to your XCF files and ensure they are correct *before* you save the file. For example, you could create a test.xcf file and validate that the test.xcf file is correct; then replace the production xcf file (assume the file is named prod.xcf for this discussion) by archiving prod.xcf and renaming test.xcf to prod.xcf.

Socket port selection

The following three entries of the INTERFACE_TO_DVM configuration are deprecated and will be removed in the next release.

The selection of a free socket port will be in charge of the operating system for performance issues.

- TCP_BASE_PORT
- TCP_PORT_RANGE
- EXCLUDED_PORT

License Consumption and Web Service Applications

Prior to this release, when it came to consuming licenses, you were able to go over the setting of MAX_AVAILABLE, up to the limit specified for the application by the MaxLicenseConsumption attribute.

This is no longer the case. MaxLicenseConsumption is no longer available as an application attribute and will be ignored by the Genero Application Server. The maximum limit for licenses is now given by MAX_AVAILABLE. You will not be allowed to go over this limit.

Support

XUL snippets set is no longer supported

Starting with version 2.30, the XUL snippet set is no longer supported. Instead, there is GWC for Silverlight, which covers the same kind of usage as the XUL snippet set, but with a more powerful technology.

Following OS are no longer supported: osf0510, sco0507 and uxw0711

These Operating Systems (HP Tru64 unix V5.1B, SCO OpenServer 5.0.7 and SCO UnixWare 7.1.3 - 7.1.4 & OpenServer 6.00) are very old and do not fit with the minimal OS requirements needed for the GAS redesign (specifically multi-threading).

Template functions

XPathConfig migration

Starting with version 2.30, the configuration provided in XML format to the GWC has changed; you have to adapt all of your customized snippet files containing XPathConfig expressions. Only the XML configuration needed by the GWC is loaded, so all the XPathConfig expressions must be simplified as follows:

- **Remove APPLICATION node**

  ```
  XPathConfig('/APPLICATION/TIMEOUT/USER_AGENT/text()')
  ```

  to

  ```
  XPathConfig('/OUTPUT/MAP/TIMEOUT/USER_AGENT/text()')
  ```

- **Remove DUA Id attribute**

  ```
  XPathConfig('APPLICATION/OUTPUT/MAP[@Id='DUA_AJAX']/RENDERING/MIME_TYPE/
  text()')
  ```

  to

  ```
  XPathConfig('/OUTPUT/MAP/RENDERING/MIME_TYPE/text()')
  ```

It is no longer possible to access all XPATH with function XPathConfig(). We can only access the node OUTPUT and its descendants.

Legacy connectors

Starting with version 2.30, due to the new multi-threaded architecture, legacy connectors are no longer provided in the package.

Mod_fcgid

Mod_fcgid is no more supported. Mod_fcgid architecture did not fit GAS 2.2x stateful process. Mod_fgcid has not been reconsidered in GAS 2.30.

GAS 2.22 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.22 from version 2.21. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Template path application/meta/variable

`application/meta/variable[name]` no longer returns the value of the variable.

To get the variable value, use the new path: `application/meta/variable[name]/value`.

If the new path is not used, you will get an error such as this:

```
Rendering error...
Template snippet: _default, style '_default', line 107
  Message: The 'variable' path element cannot be rendered
  automatically; please use its attributes
```

GAS 2.21 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.21 from version 2.20. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Topics

- Configuration

Configuration

- Picture

  For web applications, the picture configuration has moved from *APPLICATION/PICTURE* to *APPLICATION/OUTPUT/MAP/PICTURE*.

See PICTURE on page 335 configuration for more details.

GAS 2.20 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.20 from version 2.10.x. If you are migrating from an earlier version of Genero Application Server, first complete the migration tasks for all versions between your existing version and the target version.

Configuration

Timeout

For web applications, the timeout configuration has moved from *APPLICATION/TIMEOUT* to *APPLICATION/OUTPUT/MAP/TIMEOUT*.

See TIMEOUT configuration for more details.

Session Variables and Cookies

The cookie configuration has moved from *APPLICATION/OUTPUT/MAP* to *APPLICATION/OUTPUT*.

See HTTP_COOKIES configuration for more details.

Blob URLs

If you are using blob urls (Note: for more information please see the Template Paths - Document hierarchy topic in the Genero Application Server 2.20 User Guide), beginning with 2.20, you need to allow the access to the resources, by default the access is disabled. See WEB_APPLICATION_EXECUTION_COMPONENT on page 356 for more details.

DVM ping timeout

Since 2.20.09, the DVM ping timeout is configurable. DVM_PINGTIMEOUT

Error document

INTERFACE_TO_CONNECTOR/ERROR_DOCUMENT is no more available. This should now be configured and handled by the web server.

Snippets sets

The snippets sets are renamed with more explicit names. *set1* is renamed into *setAjax*. *set2* and *set3* are merged and becomes *setBasic*. A new set, *setXul*, for output map DUA_XUL is added.

These changes imply some migrations for your customized snippets.

- setAjax (formely known as set1)

  Adapt main.xhtml to reflect the last enhancement on resource deployment.
- setBasic (formely known as set2 and set3)

  As setBasic is the result of the merging of set2 and set3, it is recommended that you rework your customization from this new set.

Legacy connectors installation

Such an architecture is no longer recommended. It is provided to ease the migration to 2.20. The installation of the connector assumes that you have the rights to install the product in the web server directories.

To install the legacy connectors:

- On IIS, you have to create a virtual directory named "gas" for example and assign **execution** permission to this virtual directory
- On Apache, configure your own ScriptAlias named "gas" for example or use the default ScriptAlias directory "/cgi-bin/".
- Then copy the content of $FGLASDIR/legacy_connectors to the directory.

Validate the installation by accessing demo program with a URL like:

```
http://myWebServer/gas/fglcisapi.dll/ua/r/gwc-demo (with IIS)
http://myWebServer/gas/fglccgi.exe/ua/r/gwc-demo   (with Apache)
http://myWebServer/gas/fglccgi/ua/r/gwc-demo
```

If you encounter any issues or need to configure connector.xcf, please refer to GAS manual prior to 2.20.

Templates and snippets

- double the left curly brace "{{" to escape the embedded expression processing.

Upgrading from GAS 2.10.x or GWC 2.10.x

Complete these tasks when migrating from Genero Application Server 2.10.x to a later version.

Application configuration

Add `noNamespaceSchemaLocation` attribute in external application configuration file

All external application configuration files must be updated by adding the `noNamespaceSchemaLocation` attribute. When defining external application files, the "noNamespaceSchemaLocation" attribute should have this value:

- For web applications: `xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.xsd"`
- For web services applications:`xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextws.xsd"`

For example, consider the following examples of the Edit.xcf web application configuration file:

The old Edit.xcf:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <APPLICATION Parent="defaultgwc">
03    <EXECUTION>
04      <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
05    </EXECUTION>
06  </APPLICATION>
```

The new Edit.xcf:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <APPLICATION Parent="defaultgwc"
03      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04      xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/
cfextwa.xsd">
05    <EXECUTION>
06      <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
07    </EXECUTION>
08  </APPLICATION>
```

If this attribute is missing, the corresponding application will fail to start, and the following message will be written to the log file:

```
Can't find 'noNamespaceSchemaLocation' attribute in external
 application file '/home/f4gl/gwc/app/Edit.xcf'.
```

Output drivers for Internet Explorer

Specific output drivers DUA_AJAX_HTML and DUA_PAGE_HTML have been added to support certain features (such as the Canvas widget) on Internet Explorer. As a result, all customized snippets specified for DUA_AJAX will also need to be specified for DUA_AJAX_HTML.

The original CardStep1.xcf:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <APPLICATION Parent="defaultgwc"
03    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
04    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/
cfextwa.xsd">
05    <EXECUTION>
06      <PATH>$(res.path.demo.app)/card/src</PATH>
07      <MODULE>card.42r</MODULE>
08    </EXECUTION>
09    <OUTPUT>
10    <MAP Id="DUA_AJAX">
11      <THEME>
12        <SNIPPET Id="Image" Style="Picture">
           $(res.path.demo.app)/card/tpl/set1/Image.xhtml</SNIPPET>
13      </THEME>
14    </MAP>
15    </OUTPUT>
16  </APPLICATION>
```

The new CardStep1.xcf:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <APPLICATION Parent="defaultgwc"
03    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04    xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/
cfextwa.xsd">
05    <EXECUTION>
06      <PATH>$(res.path.demo.app)/card/src</PATH>
07      <MODULE>card.42r</MODULE>
08    </EXECUTION>
09    <OUTPUT>
10    <MAP Id="DUA_AJAX">
11      <THEME>
12        <SNIPPET Id="Image"
           Style="Picture">$(res.path.demo.app)/card/tpl/set1/Image.xhtml
          </SNIPPET>
13      </THEME>
14    </MAP>
15    <MAP Id="DUA_AJAX_HTML">
16      <THEME>
17        <SNIPPET Id="Image"
           Style="Picture">$(res.path.demo.app)/card/tpl/set1/Image.xhtml
        </SNIPPET>
18      </THEME>
19    </MAP>
20    </OUTPUT>
21  </APPLICATION>
```

Likewise, all customized snippets specified for DUA_PAGE will also need to be specified for DUA_PAGE_HTML.

To change output drivers default behaviors, see Automatic Discovery of User Agent.

URL parameters

By default, parameters in the URL are not taken into account. They are not transmitted to the DVM. Only the parameters defined in the configuration files are transmitted.

To use URL parameters, in the EXECUTION tag, you have to set **AllowUrlParameters** to TRUE.

Caution, parameters are transmitted to the DVM in this order: configured parameters in PARAMETERS on page 331 tag followed by the URL parameters.

Template and snippets

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Deprecated functions and paths

Some functions have been renamed due to enhancements on the Front End protocol. The default logging includes the DEPRECATED category that displays warnings. If any deprecated functions are used, this kind of warning is logged:

```
[ TASK=1808 VM=1860 WA=115128484 TEMPLATE ]
 Event(Time='7.481526', Type='Using deprecated function')
 / function(Name='makescrollpagesizedid')
```

Deprecated template paths

- DID becomes IDID

Deprecated template functions

- makeCompoundRowSelectionDID becomes makeRowSelectionIDID
- makeScrollPageSizeDID becomes makePageSizeIDID
- makeScrollOffsetDID becomes makeScrollOffsetIDID

GAS (GWC) 2.10 upgrade guide

If you have been working with the Genero Web Client prior to the release of GWC 2.10, you will have already done some configuration and possibly customization to deliver your Genero applications as Web applications using the initial built-in rendering engine (no longer supported).

All new development and advances is focused on the snippet-based rendering engine introduced with GWC 2.10.

- Migrating to the snippet-based rendering engine on page 260

Migrating to the snippet-based rendering engine

To take full advantage of the snippet-based rendering engine, you must follow the procedures outlined in this manual, regardless of whether or not you have previously deployed your application using the pre-2.10 GWC.

If you have previously deployed the application with the pre-2.10 GWC, ensure you revisit the following:

- Customization now includes presentation styles and snippet sets; previous customization of the template files are no longer valid with the snippet-based rendering engine. Customization previously implemented using JavaScript™ will now likely be implemented using HTML.
- The limitations of the GWC prior to 2.10 included lack of accelerator key support, StartMenus, ProgressBars, ON IDLE, StatusBars, and Genero Presentation Styles. You may have modified your application to work around these limitations. (Please see the topic, Features and limitations, in the Genero Application Server 2.10 User Guide.)

  Note: Links to pages made obsolete by more current releases have been removed from this page. If you are migrating to GWC 2.10, we recommend you use the documentation released with version 2.10.

GAS 2.00 upgrade guide

Complete these tasks when migrating to Genero Application Server 2.00 from any earlier version of the Genero Application Server.

fglxslp migration tool

When migrating from Genero Application Server (GAS) 1.3x to 2.00, it is necessary to update your GAS configuration file to conform to the XML specifications of GAS 2.00. A migration tool, fglxslp, has been added to assist you in this migration.

Usage:

```
$FGLASDIR/bin/fglxslp $FGLASDIR/etc/gasxcf1xxto200.xsl
 $FGLASDIR/etc/as-132.xcf > $FGLASDIR/etc/myas.xcf
```

Note:

- *fglxslp* is the migration tool.
- *gasxcf1xxto200.xsl* is the XSL style sheet that describes the GAS 2.00 XML configuration file
- *as-132.xcf* is the configuration file to migrate (GAS 1.3x).
- *myas.xcf* is the result (new configuration file for GAS 2.00).

fglxmlp XML preprocessor

The XML Preprocessor can be used as part of the BDL development process. It fetches data in a XML resource file to "fill" the content of a source file that contains the dollar tag expression.

Usage:

```
$FGLASDIR/bin/fglxmlp -i src1.4gx -o src1.4gl -r resource.xrf
```

Note:

- *src1.4gx* is the file to be processed through the XML Preprocessor.
- *src1.4gl* is the output file.
- *resource.xrf* is the XML resource file containing the definition of a complex 4GL record.

Using the XML Preprocessor

In this example, two source files will be "expanded" through the XML resource file. The resource file contains the definition of a complex 4GL record. The extension of files to be processed through the XML Preprocessor is .4gx. The extension for the resource file is .xrf (XML Resource File).

```
fglxmlp -i src1.4gx -o src1.4gl -r resource.xrf
```

```
fglxmlp -i src2.4gx -o src2.4gl -r resource.xrf
```

The resulting.4gl files are compiled and link as usual:

```
fglcomp -c src1.4gl
```

```
fglcomp -c src2.4gl
```

```
fgllink -o project.42r src1.42m src2.42m
```

Files used in the example

src1.4gx :

```
01 FUNCTION useRecord (myRecord)
02    DEFINE myRecord $(record)
...
06 END FUNCTION
```

resource.xrf :

```
01 <?xml version="1.0" ?>
02
03 <RESOURCE_FILE>
04   <RESOURCE_LIST>
05     <RESOURCE Name="record"><![CDATA[
06       RECORD
07         nb_columns INTEGER,
08         nb_lines INTEGER,
09         name CHAR (8)
10       END RECORD
11     ]]></RESOURCE>
12   </RESOURCE_LIST>
13 </RESOURCE_FILE>
```

The output file src1.4gl :

```
01 FUNCTION useRecord (myRecord)
02   DEFINE myRecord
03       RECORD
04         nb_columns INTEGER
05         nb_lines INTEGER,
06         name CHAR (8)
07       END RECORD
...
15 END FUNCTION
```

Migrating Templates and Snippets Customizations

Starting with Genero 3.00, templates and snippets are deprecated

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Reference

Genero Application Server reference.

Tools and Commands

Information about the dispatchers, proxies, and command line utilities.

GAS Dispatchers

GAS dispatchers refer to the connectors in charge of dispatching a GAS request to the appropriate proxy.

- httpdispatch (Standalone)
- fastcgidispatch (Web Server)
- java-j2eedispatch (Java Servlet)
- isapidispatch, delivered as a DLL; dedicated to IIS (Windows™ platforms)

Proxy

Proxies refer to binaries started by a dispatcher to serve a type of request.

- uaproxy (GWC-JS, GDC)
- gwsproxy (GWS)

Utility

Binaries delivered to ease XML files processing, such as configuration migration.

- fglxslp
- fglgar
- gasadmin

Dispatcher: httpdispatch

`httpdispatch` is the standalone dispatcher that starts the Genero Application Server (GAS) in command line. No web server is needed. It is only used in development mode.

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

Note: The GAS is configured through the GAS configuration file. This configuration file can be the default configuration file (**$FGLASDIR/etc/as.xcf**) or a custom configuration file that is specified when the Genero Application Server is started. For more information on configuration parameters, see GAS configuration file on page 289.

Syntax

```
httpdispatch [options]
```

Options

Table 51: httpdispatch options

Option	Description
`-h` `--help`	Displays help information.
`-p` *directory* `--as-directory` *directory*	Specify the Genero Application Server directory.
`-f` *configuration_file* `--configuration-file` *configuration_file*	Specify which configuration file to use when starting the Genero Application Server dispatcher. If not specified, the default configuration file, `$FGLASDIR/etc/as.xcf`, is used.
`-k` `--no-keepalive`	Disable keep alive for http connections. For debug purpose only.
`-E` *name*`=`*value* `--resource-overwrite` *name*`=`*value*	Overwrites the resource defined in the configuration file or creates a new one. Example: `httpdispatch -E res.dvm.wa=$FGLDIR/bin/myrun` If in the configuration file "res.dvm.wa" has another value it is now set to myrun. The **final value** is the one set in the **option**.
`-V` `--version`	Displays version information.

What does "address already in use" mean ?

The message "address already in use" means that an application server (dispatcher) has already been started on the same port. Check the GAS configuration file (default `$FGLASDIR/etc/as.xcf`) to identify the port where the application server (dispatcher) started. The port number is identified in the following section:

```
<INTERFACE_TO_CONNECTOR>
  <TCP_BASE_PORT>6300</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
```

```
    ...
</INTERFACE_TO_CONNECTOR>
```

The default port specified is 6394 - derived by adding the base port (6300) to the port offset (94). Set the values to a port which is not used by another application, see INTERFACE_TO_CONNECTOR on page 324.

Restarting the standalone GAS

To restart httpdispatcher, use:

```
kill -9
```

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

Ctrl + C or sending SIGTERM will stop the standalone dispatcher, and in both cases the dispatcher will request all proxies to stop. With `kill -9` the dispatcher process is killed yet the sessions remain alive and untouched. When the dispatcher is restarted,the sessions continue to be active. Notice, the fastcgi dispatcher will stop sessions on Ctrl-C too if started in standalone mode. But not on SIGTERM.

Dispatcher: fastcgidispatch

`fastcgidispatch` is the dispatcher for web server supporting FastCgi protocol.

It can be started by the web server or in command line. See FastCGI Installation and Web Server Configuration on page 85 for more details.

Syntax

```
fastcgidispatch [options]
```

Options

Table 52: fastcgidispatch options

Option	Description
`-p` *`directory`* `--as-directory` *`directory`*	Specify the Genero Application Server directory.
`-f` *`configuration_file`* `--configuration-file` *`configuration_file`*	Specify which configuration file to use when starting the Genero Application Server dispatcher. If not specified, the default configuration file, `$FGLASDIR/etc/as.xcf`, is used.
`-E` *`name=value`* `--resource-overwrite` *`name=value`*	Overwrites the resource defined in the configuration file or creates a new one. Example: `fastcgidispatch -E res.dvm.wa=$FGLDIR/bin/myrun` If in the configuration file "res.dvm.wa" has another value it is now set to myrun. The **final value** is the one set in the **option**.

Option	Description
`-s` `--standalone`	Start the fastcgi dispatcher in standalone mode, where the dispatcher is started prior to any actual request from an application or web server. The web server's FastCGI extension must be configured to connect to an external GAS.
`-h` `--help`	Displays help information.
`-V` `--version`	Displays version information.

For information on restarting the fastcgidispatch, see Restarting the FastCGI dispatcher on page 90.

Dispatcher: java-j2eedispatch

The java-j2eedispatch is a Java servlet that manages the Genero Application Server in a Java Web container, such as Tomcat™ or JBoss.

The Genero Application Server provides a Java servlet dispatcher called `java-j2eedispatch` by default, which can be deployed on any Java server supporting servlets. It offers the same functionality as Dispatcher: httpdispatch on page 263 and Dispatcher: fastcgidispatch on page 265 but has the advantage of opening the Genero technology to the Java web servers such as Tomcat™, WebSphere®, GlassFish or JBoss.

The `java-j2eedispatch` dispatcher requires at least version 3.0 of the Java servlet API. This is supported on the following Java web servers :

- WebSphere (since version 8.0), see http://en.wikipedia.org/wiki/IBM_WebSphere_Application_Server
- Tomcat™ (since version 7.0.x), see http://tomcat.apache.org/whichversion.html
- Glassfish (since version 3.0), see https://glassfish.java.net/public/comparing_v2_and_v3.html

Proxy: uaproxy

The uaproxy manages a GWC-JS or GDC application session.

It must be started by a dispatcher. If you attempt to run the `uaproxy` from the command line, you will likely get an error. The proxies are not intended to be run from the command line, but started by a dispatcher in the proper environment.

Syntax

```
uaproxy [options]
```

Options

Table 53: uaproxy options

Option	Description
`-h, --help`	Display help.
`-V, --version`	Show version.

Option	Description
--development	Enable the development mode that gives access to the application AUI tree and logs. See Configuring development environment on page 147

Proxy: gwsproxy

The gwsproxy manages all services for a single Genero web application.

The `gwsproxy` must be started by a dispatcher.

If you run the `gwsproxy` from the command line, you will get the message "ERROR: FGL_VMPROXY_SESSION_ID not set".

Proxy: html5proxy

Starting with Genero 3.00, the `html5proxy` proxy is deprecated.

It must be started by a dispatcher. If you attempt to run the `html5proxy` from the command line, you will likely get an error. The proxies are not intended to be run from the command line, but started by a dispatcher in the proper environment.

Syntax

```
html5proxy [options]
```

Options

Table 54: html5proxy options

Option	Description
--development	Enable the development mode that gives access to the application gtree and log.
--dump-io	Enable scenario dump. Log internal information for debugging purpose. Do not set it unless directed by your support center.

The fglxslp command

The `fglxslp` command is the same tool as the `xsltproc` tool (a command line tool for applying XSLT stylesheets to XML documents), mainly used to have a XSLT processor on every system. For more details on the `fglxslp` command, enter `fglxslp -h` from the command line.

Important: This utility is not supported.

The fglgar command

The Genero Archiver tool (`fglgar`) enables you to build a java Web Archive (`war`) to deploy Genero Application Server (GAS) on Java Web servers or to build a Genero archive (`gar`) to deploy Genero applications on GAS.

It configures the Java™ Servlet to use the GAS directory and GAS configuration file. It also provides a deployment framework for managing application resources and deploying public images on the GAS.

Syntax

```
fglgar [options]
```

Options

Table 55: fglgar options

Option	Description
`-V` `--version`	Displays version information.
`-h` `--help`	Displays help information.
`-o` `--output`	Name of archive file to create.
`-v` `--verbose`	Displays the verbose output of additional information.
`-q` `--quiet`	Silent mode
`-d` `--output-directory`	Directory where to save the archive file.
`-s` *directory* `--input-source` *directory*	Directory to archive.
`-w` `--war`	Option to generate the Java™ Web Archive. To run `fglgar --war` on a Windows or Mac, the user running the command must have write permission on the `FGLASDIR` directory. If you cannot grant write permissions on the `FGLASDIR` directory, move all of the *`FGLASDIR/`*`war` directory to another directory with write permission. See Building the Java Web Archive (WAR) on page 91
`-g` `--gar`	Option to generate the Genero Web Archive.
`-p` *directory* `--as-directory` *directory*	Specify the Genero Application Server directory.
`-a` *directory* `--asdir` *directory*	Specify the Genero Application Server directory.

Option	Description
`-x` *config_file* `--asxcf` *config_file*	Specify which configuration file to use when starting the Java™ dispatcher. If not specified, the default configuration file, `$FGLASDIR/etc/as.xcf`, is used.
`-f` *config_file* `--configuration-file` *config_file*	Specify which configuration file to use when starting the Java™ dispatcher. If not specified, the default configuration file, `$FGLASDIR/etc/as.xcf`, is used.
`--resource` *directory*	Specify the Genero archive resource directory. See Application deployment overview on page 218
`--trigger-component`	Genero archive trigger execution component, see TRIGGERS (for manifest) on page 224.
`--deploy-trigger` *command*	Genero archive deploy trigger command. See TRIGGERS (for manifest) on page 224
`--undeploy-trigger` *command*	Genero archive undeploy trigger command. See TRIGGERS (for manifest) on page 224
`--application` *config_file*	Genero archive application configuration file.
`--service` *config_file*	Genero archive service configuration file.

Example building a Web archive

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

Generates a Java™ Servlet called **java-j2eedispatch.war** using the Genero Application Server located at **C:\usr\gas\3.00** and with the configuration file located at **C:\conf\as.xcf**, and accessible via URL of type **http://host:port/java-j2eedispatch/demos.html**.

```
fglgar --war --asdir C:\usr\gas\3.00 --output connector
```

Generates a Java™ Servlet called **connector.war** using the Genero Application Server located at **C:\usr\gas\3.00** and accessible via URL of type **http://host:port/connector/demos.html**.

The gasadmin command

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

With this tool, you can:

- Display GAS version information, see Display GAS version.
- Specify the GAS application directory, see Specify GAS directory
- List all sessions of a specified dispatcher, see List sessions
- Stop dispatcher sessions, see Kill sessions
- Validate the GAS configuration, see Validating with the gasadmin tool on page 94
- Explode the GAS configuration, see Explode configuration file
- Ping dispatchers active sessions, see Ping active sessions
- Manage archive files, see Genero Archive procedures on page 226

Syntax

```
gasadmin [options]
```

Options

Table 56: gasadmin options

Option	Description
`-p` *directory* `--as-directory` *directory*	Specify the Genero Application Server directory.
`-f` *configuration-file* `--configuration-file` *configuration-file*	Specify the configuration file to use. If not specified, the default configuration file, *$FGLASDIR*`/etc/as.xcf`, is used.
`-E` *name=value* `--resource-overwrite` *name=value*	Define or overwrite a resource.
`-c` `--configuration-check`	Check the configuration file, then exit.
`-e` `--configuration-explode`	Explode the GAS configuration into files, one for each application.
`-t` `--configuration-explode-external`	Explode the given external configuration file in current directory.
`-r` `--configuration-expand-resources`	Expand resources and replace with real values. Used with `--configuration-explode` or `--configuration-explode-external`.
`-d` *dispatcher-name* `--dispatcher` *dispatcher-name*	Used by session-related options to select the target dispatcher.
`-K -d` *dispatcher-name* `--kill-all-sessions` `--kill-all-sessions --dispatcher` *dispatcher-name*	**Note:** The option `-d <dispatcher_name>` is optional. If omitted, the option kills all sessions of all dispatchers.
`-k` *session-id* `-d` *dispatcher-name* `--kill-session` *session-id* `--dispatcher` *dispatcher-name*	Terminate (kill) the requested session id of the specified dispatcher. **Important:** The option `-d <dispatcher_name>` is required, as the same session identifier can exist for the various dispatchers.
`-X` `-X -d` *dispatcher-name*	Ping all active sessions of the specified dispatcher.

Option	Description
`--ping-all-sessions` `--ping-all-sessions --dispatcher` *`dispatcher-name`*	**Note:** The option `-d <dispatcher_name>` is optional. If omitted, the option pings all active sessions of all dispatchers.
`-x` *`session-id`* `-d` *`dispatcher-name`* `--ping-session` *`session-id`* `--dispatcher` *`dispatcher-name`*	Ping the request session id of the specified dispatcher.

`-l`
`-l -d` *`dispatcher-name`*
`--list-sessions`
`--list-sessions --dispatcher` *`dispatcher-name`*
	List all sessions of the specified dispatcher.
	Note: The option `-d <dispatcher_name>` is optional. If omitted, the option lists all sessions of all dispatchers.

`-z` *`paths`* `--compress-resources` *`paths`*	Compress the resources located in specified paths. The path separator is a comma (,). See example.
`-C` `--session-cleanup`	Clear remaining Unix Domain sockets.
`-h` `--help`	Display help information.
`-V` `--version`	Display GAS version information. See example
`-q` `--quiet`	Silent mode.
`--deploy-archive` *archive file*	Unpack the specified archive content into the deployment directory.
`--undeploy-archive` *archive file*	Undeploy the specified archive.
`--enable-archive` *archive file*	Expose all services and applications contained in the specified archive.
`--disable-archive` *archive file*	Unexpose all services and applications contained in the specified archive.
`--list-archives`	List all archives deployed on the Genero Application Server.
`--clean-archives`	Clean up all undeployed archives.
`--xml-output`	List all archives and applications deployed on the Genero Application Server. Only compatible with archive options.
`-y`	Do not prompt for confirmation.

Option	Description
`--yes`	

Usage Examples

`gasadmin -t demo/Card`	Explode external configuration file demo/Card.xcf and expand its resources and its parent's resources into an XML file.
`gasadmin -r -t demo/Card`	Explode external configuration file demo/Card.xcf, replace its resources and its parent resources with real values, and expand them into individual XML files.
`gasadmin -X -f as1.xcf -d httpdispatch`	List all sessions having `as1.xcf` as configuration file for dispatcher *httpdispatch*.
`gasadmin -k d98290172c8f7c0d861db329f1ce6597 -f as1.xcf -d isapidispatch`	Kill the session with the id d98290172c8f7c0d861db329f1ce6597 that has `as1.xcf` as its configuration file, and *isapidispatch* as its dispatcher.
`gasadmin -z $FGLASDIR/app,$FGLASDIR/services,$FGLASDIR/web,$FGLASDIR/tpl`	Compress the resources located in the paths *$FGLASDIR*`/app`, *$FGLASDIR*`/services`, *$FGLASDIR*`/web` and *$FGLASDIR*`/tpl`
`gasadmin -V`	Display GAS version information.

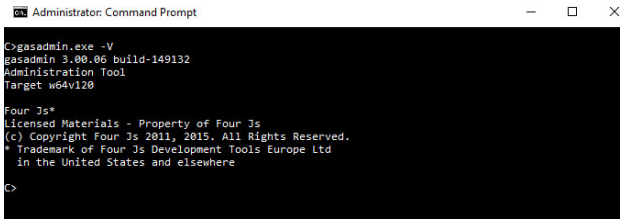


Figure 56: Sample GAS Version Information

Ghost Client and Testing Tools

This section provides an overview of the Genero `Ghost Client` (GGC) Java framework API and describes procedures that you can use for testing Genero applications.

The Genero `Ghost Client` is a Java framework tool that allows you to test different scenarios by emulating user interaction on applications running on a Genero server (i.e. Genero Application Server and runtime).

You can use the `Ghost Client` to automate, for example, the following tests:

- Unit tests
- Load tests
- Performance tests

Test scenarios can be developed in two ways:

- Tests can be written using both Java and BDL and be compiled to Java for testing by the `Ghost Client`.
- Test scenarios can also be generated from the behavior described in a log file recorded by the GDC or the GWC-JS clients. The Ghost Generator feature generates Java files automatically from the log file data, which allows you to replay the resulting scenarios with the `Ghost Client`, see Generating test scenarios from log file on page 281.

One of the key features of testing with the `Ghost Client` is that you do not need to modify the original application code to write test scenarios. The `Ghost Client` allows you to develop working test case scenarios that can be run as required to test the stability of your applications before release. The `Ghost Client` allows you to test applications that are targeted for different Genero clients, i.e. GWC-HTML5, GDC, GWC-JS, GMA, or GMI.

Installing Ghost Client

This topic provides information about how to install your `Ghost Client` API package.

About this task

The procedure in this topic shows you how to carry out an installation of `Ghost Client`.

Before you begin:

- Ensure Genero Ghost Client is supported for your operating system. For a list of supported operating systems, refer to the download page (available on the Four Js Web site) or contact your support center.
- To use Genero `Ghost Client` API, you need a Java Development Kit (JDK). Make sure that your JDK version is at least version 1.7 or greater.

Once the JDK has been installed, you are ready to install the `Ghost Client` as described in the next step.

1. Download the package appropriate for your operating system.
2. Unzip the GGC package to a directory in your server .

Now that you have installed GGC, your next task is to configure your environment for the Ghost Client as detailed in Configuring your environment for Ghost Client on page 273.

Configuring your environment for Ghost Client

This topic provides information about how to configure your environment to use the `Ghost Client`.

About this task

The procedure in this topic shows you how to configure your environment so that `Ghost Client` can use Java.

Before you begin:

To use Genero `Ghost Client`, you need to have Java Development Kit installed. Make sure that your JDK is installed. See Installing Ghost Client on page 273.

> **Important:** Before you can run scenarios with the GGC, you need to set up your PATH and CLASSPATH environment variables as described in the next steps.

1. Set your *PATH* environment variable to include the Java Development Kit `bin` directory.

   On Linux®/UNIX™/MAC®:

   ```
   export PATH= /path-to-jdk/bin:$PATH
   ```

   On Windows™:

   ```
   set PATH =  C:\path-to-jdk\bin;%PATH%
   ```

2. To be able to compile and run test scenarios from any directory on your disk, set your *CLASSPATH* environment variable to the absolute path to the `ggc.jar` and `fgl.jar` files.

   Note: You can also specify the path to jar libraries directly at the command line using the Java -classpath (-cp) option, see Example using Java -cp option below.

   On Windows®:

   ```
   set CLASSPATH =  C:\path-to-the-ggcdir\ggc.jar;C:\path-to-fgldir\fgl.jar
   ```

   On Linux®/UNIX™/Mac®:

   ```
   export CLASSPATH =/path-to-the-ggcdir/ggc.jar:/path-to-fgldir/fgl.jar
   ```

   Note: As the CLASSPATH now references the jars needed by the GGC, you can compile and run your scenarios as shown in the examples:.

   - Compiling:

     ```
     javac userWorkspace/path/to/generated/.*java
     ```
   - Running:

     ```
     java com.fourjs.ggc.Launcher  -s
     userWorkspace.path.to.generated.UserClass -u http://application/url
     ```

   An alternative to setting the *CLASSPATH* environment variable is to specify the absolute path to the `ggc.jar` file using the `java` -classpath (-cp) option at runtime. Below are some examples.

   On Linux®/UNIX™/Mac®:

   - Compiling:

     ```
     javac -cp /absolute/path/to/userWorkspace:/absolute/path/to/
     ggc.jar userWorkspace/path/to/generated/.*java
     ```
   - Running:

     ```
     java -cp /absolute/path/to/userWorkspace:/absolute/
     path/to/ggc.jar com.fourjs.ggc.Launcher -s
     userWorkspace.path.to.generated.UserClass -u http://application/
     url
     ```

   On Windows®:

   - Compiling:

     ```
     javac -cp C:\absolute\path\to\userWorkspace;C:\absolute\path\to
     \ggc.jar userWorkspace\path\to\generated\.*java
     ```
   - Running:

     ```
     java -cp C:\absolute\path\to\userWorkspace;C:
     \absolute\path\to\ggc.jar com.fourjs.ggc.Launcher -s
     userWorkspace.path.to.generated.UserClass -u http://application/
     url
     ```

   For more information on -classpath (-cp) option, see the Java documentation.

How Ghost Client works

This topic provides an overview of the Ghost Client infrastructure and describes how it works.

The Ghost Client infrastructure overview

The Ghost Client is a Java framework that provides you with the API structure of interfaces, classes and methods for building application tests. The Ghost Client infrastructure consists of three components:

Launcher	This is the Ghost Client Java program which starts a testing session for an application.
SessionManager	The SessionManager is the interface that manages the runtime and the test Scenario.
Scenario	The Scenario is the interface that contains the sequence of tests run during a given session.

Ghost Client interfaces and classes

The GGC's main public classes are described in the table below. The GhostRunner class, for example, is the entry point for test case scenarios.

Table 57: Ghost Client classes

Class	Description
Session	Exposes the application object to your BDL program so that it can be retrieved from the session.
Application	Exposes the application user interface to your BDL program so that form objects such as field values can be retrieved and set.
GhostRunner	Provides methods to control a Genero application running in a Genero Application Server. It contains methods for a set of possible actions that an end-user might do on the running application, e.g. set focus on a field, set a value in a field, etc.
Log	Implements the logging mechanism.

The complete details of the packages that make up the Ghost Client and the classes and interfaces it uses can be found in the /doc directory of your GGC package. For more information about specific functions along with detailed information about each class and interface, please see the help file by launching the /doc/index.html file in your browser.

Developing tests with Ghost Client

When you write tests to be run by the Ghost Client, you need to implement its two main interfaces: the SessionManager and the Scenario. The following describes generally the function each provides and the methods the GGC needs from their implementation:

Table 58: Ghost Client interfaces

Class	Description
SessionManager	Instantiates and manages Scenario instances according to incoming VM connections and new runtime launches.

Class	Description
Scenario	A Scenario describes the action sequence played during the GhostRunner session to simulate user actions on a Genero application.

- The SessionManager requires you to include a getScenario() method in your SessionManager class that instantiates the Scenario or sequence of scenarios required to run your tests.
- Your Scenario class requires two methods:
  - A play(GhostRunner runner) method, which describes the user interaction sequence to play; that is the sequence of tests it plays.
  - An InvokeFrontcall(String module, String name, String[] args) method, which describes how to handle the call to the front-end client when and if it is required by the application.

For more detailed information on developing tests with Ghost Client see Unit testing with Ghost Client on page 277.

Launching a test with Ghost Client

To launch a Ghost Client test, at the command line you run Java with the Launcher program providing the URL of your application and the path to the SessionManager as shown in the sample syntax below:

```
java com.fourjs.ggc.Launcher -u http://<host>:<port>/gas/ua/r/<group>/<myapp>
-s path.to.mySessionManager.mySessionManager.java <-options>
```

> **Note:** Depending on the user agent protocol in the URL you provide (e.g. /wa/r/, /ja/r/, /ua/r/), the Ghost Client will run the application tests behaving as either a GWC-HTML5, GDC, or version 3 client (i.e. GWC-JS, GMA, or GMI).

The Ghost Client allows you to run tests with different options, for example, you can specify that it launches the test for all protocols with the --all_mode switch. For more information on available options, see below.

Table 59: Ghost Client options

Option	Description
-u URL	Specify the URL of your application.
-s path.to.mySessionManager	Specify the SessionManager java file to use.
-t value	Specify the number of thread instances to launch to simulate the number of users using an application.
-td value	Define the delay in seconds between the launch of each thread instance.
--all_mode	Launch the test for all protocols, /wa/r/, /ja/r/, /ua/r/.
--log (-l) path/to/log_file	Specify a log file to use to generate a test set, see Generating test scenarios from log file on page 281
--write (-w) path/to/generate/java_files	Write Java files generated from a log file to specified path

Unit testing with Ghost Client

This topic describes the processes you can use for developing unit tests for your Genero applications.

What is unit testing?

The aim of unit testing is to test each feature of your application in isolation to make sure it works as expected. A good unit test should provide you with the correct responses to a given set of anticipated user input, showing that the feature is able to handle correct as well as incorrect input. As a developer approaching unit testing for the first time, the following is recommended as good practice when designing unit tests:

- It is recommended that each `4gl` application should have its own `Scenario`.
- Make a complete list of the application features, from the smallest (for example, displaying the **About...** screen), to the biggest and / or the most important ones.
- For each feature in this list, write a single test that will test one (and only one) feature.
- If a primary feature implies a secondary feature (for example, to register a new customer, you need to fill out a form), the secondary feature's test should take place before the primary feature's test in the test sequence.

Unit testing with Ghost Client

The `GGC` provides a simple framework to write and automate unit tests. You can develop these tests separate to your actual `4gl` application source code via the `SessionManager` and the `Scenario` interfaces. Typically, these are written as individual classes:

- The `SessionManager` is a Java class that manages the whole test session life cycle. You must include in this the `Scenario` instance creation that the main application (i.e. the first one launched) needs to get to run the tests. If child applications are run by the main application, the `SessionManager` will also get each `Scenario` and instantiate it, as shown in the example.

```
import com.fourjs.ggc.Scenario;
import com.fourjs.ggc.SessionManager;

public class mySessionManager
  implements SessionManager
{
 boolean mStarted = false;
 public Scenario getScenario()
    {
        if (!mStarted) {
            mStarted = true;
            return new FirstScenario(); << will be given to the first 4gl
 application, the mother
        } else {
            return new SecondScenario(); << will be given to the second
 4gl application, the child launched via run or run without waiting by the
 mother
        }
    }
}
```

- A `Scenario` is a Java class that tests different functions of your application. You can decide what tests are to be carried out by providing within the `play` method the required user interactions. Any interaction a user would normally do on an application, can be played by a `Ghost Client` API method. For example, if you want to have your application launch an action, you can get the `Ghost Client` to play the interaction by passing the action name in the `sendAction` method, as shown in the example.

Note: For more information about GhostRunner methods, please see the Scenario samples included in the /samples/java directory of your GGC package or see the help file located in the /doc directory.

```
/**
 * The Scenario to test the "ButtonEdit" application from FGLGWS demo
 */
public class ButtonEditScenario
  implements Scenario
{
 @Override
 public void play(GhostRunner runner)
 {
  Log.info("ButtonEditDelegatedScenario: started.");
  try {
   runner.sendAction("INPUT");
   // TODO need to check/do something
   // After that we close the application
   runner.sendAction("close");
   runner.close();
  } catch (GhostException e) {
   Log.error("ButtonEditDelegatedScenario: exception raised!", e);
  }
  Log.info("ButtonEditDelegatedScenario: done.");
 }
 @Override
 public String[] invokeFrontcall(String module, String name, String[]
 param)
 {
  return null;
 }
}
```

Note: Both Java and BDL can be used to write SessionManager and Scenario classes but these must be compiled to Java to be run by the GGC. In Genero Studio you can write tests using BDL and compile them to Java using the Ghost Client infrastructure through the Java Bridge. The Java Bridge allows you to import GGC classes, which opens up their methods for use. For more information about using BDL, please see the Genero Studio project, BDLSample.4pw, available in the samples/BDL directory.

Load testing with Ghost Client

This topic describes the processes you can use to implement load testing for your Genero applications.

What is load testing?

This testing strategy consists of simulating a specified number of user, all of them using your application at the same time and at normal human speed. You can use the GGC to test your application during development to see how it would behave in conditions in a production environment. This can help you identify, for example, server (e.g. how many servers, how much memory, etc.) and network requirements based on the number of anticipated users. Observations made under these test conditions will also show up the application's weaknesses and will allow you to fix them before release.

Load testing with Ghost Client

The GGC allows you to specify several options through the Launcher class for load testing:

- The thread_number option allows you to set the number of users to simulate, one thread representing one user.
- The thread_delay option allows you to set a delay between each user's connection. In a real life situation where, for example, a thousand users could potentially launch an application, the likelihood

of them all connecting at the exact same time is quite remote, so the thread delay option helps you to recreate a more real life situation.

You can specify the number of threads (users) and the thread delay using the command line switches when launching tests. As shown in the example below, the GGC will launch the IntegrityTestsSessionManager with 3 users, allowing a 3 second delay between each launch.

```
java com.fourjs.ggc.Launcher -s
com.fourjs.ggc.testcases.IntegrityTestsSessionManager -u http://
localhost:6394/ua/r/gwc-demo -t 3 -td 3
```

Developing Scenario for load testing

The Scenario for load testing needs to reflect how a real user would use an application. For example, the setDelay() method can be set to make the GhostRunner instance wait for a specified number of seconds between each action. This allows you to try to reproduce the human speed of interaction with an application, which is usually much slower than computer speed. Apart from that any interaction a user would normally do on an application can be played.

Performance testing

Performance testing is similar to load testing as it also consists of simulating a significant number of users but the aim is to determine when the system's performance will start to degrade. So, in this case you would specify the number of simulated user as high as possible so as to observe when the request/response delay begins to be affected by the number of users.

Exploring Ghost Client Java demos

This topic provides information about how to view the Ghost Client Java demos provided with the installation.

About this task

The procedure in this topic shows you how to use the Java demos provided with the Ghost Client installation to run integrity tests on the gwc-demo applications via the Ghost Client.

Before you begin:

- Make sure that the directory where your GGC package is installed has the following samples included in its /samples/java directory:
  - One SessionManager implementation named IntegrityTestsSessionManager.java that will manage the testing session
  - One Scenario implementation named GenericScenario.java that will instantiate real scenarios depending on the application launched by the GAS
  - Several simple Scenario implementations that will be invoked through the GenericScenario
- Make sure your GAS version is at least version 2.50.34 or greater.
- Make sure your environment is configured to run Java and the GGC, see Configuring your environment for Ghost Client on page 273.
- Make sure that the standalone dispatcher httpdispatch see Dispatcher: httpdispatch on page 263 is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser.

1. To run the test simulating a GDC 2.50 over HTTP client, type the command as shown below:

   ```
   java com.fourjs.ggc.Launcher -u http://localhost:6394/ja/r/gwc-demo -s
   com.fourjs.ggc.testcases.IntegrityTestsSessionManager
   ```

   Below is sample output logged by the GGC to the console during the running of integrity tests for CustomerOrderScenario.

   ```
   ...
   ```

```
[INFO] (tid:18) 12:5:26:926 GenericScenario.java:57
         > GenericScenario: ApplicationName=CustOrders /
 remainingScenario=null,null,null,fglp
ed,CustOrders,
[INFO] (tid:18) 12:5:26:927 GenericScenario.java:62              >
 GenericScenario: ApplicationName=[CustOrders] started.
[INFO] (tid:18) 12:5:26:927 CustomerOrderScenario.java:41        >
 CustomerOrderScenario: started.
[INFO] (tid:18) 12:5:27:40 CustomerOrderScenario.java:67         >
 CustomerOrderScenario: Creating a new customer ...
[INFO] (tid:18) 12:5:27:101 CustomerOrderScenario.java:93        >
 CustomerOrderScenario: Searching a given customer (with cust_id=3)  ...
[INFO] (tid:18) 12:5:27:116 CustomerOrderScenario.java:100       >
 CustomerOrderScenario: checking this customer orders  ...
[INFO] (tid:18) 12:5:27:121 CustomerOrderScenario.java:129       >
 CustomerOrderScenario: modifying one of his orders (order_id=305) ...
[INFO] (tid:18) 12:5:27:181 CustomerOrderScenario.java:151       >
 CustomerOrderScenario: Use the zoom to select the item.
[INFO] (tid:18) 12:5:27:218 CustomerOrderScenario.java:170       >
 CustomerOrderScenario: Check the new order is correct.
[INFO] (tid:18) 12:5:27:227 CustomerOrderScenario.java:182       >
 CustomerOrderScenario: done.
[INFO] (tid:18) 12:5:27:230 GenericScenario.java:71              >
 GenericScenario: ApplicationName=[CustOrders] done.
...
```

2. To run the test simulation on GDC, GWC-JS, GMA, or GMI clients, type the command shown below:

   ```
   java com.fourjs.ggc.Launcher -u http://localhost:6394/ua/r/gwc-demo -s
   com.fourjs.ggc.testcases.IntegrityTestsSessionManager
   ```

Exploring Ghost Client BDL demos

This topic provides information about how to view the BDL demos provided with the `Ghost Client` installation.

About this task

The procedure in this topic shows you how to use the BDL demos provided with the `Ghost Client` installation to test the integrity of the `gwc-demo` applications in simulation mode via the `Ghost Client`.

Before you begin:

- Make sure that the `samples/BDL` directory where your `GGC` package is installed has the `DemoTests.4gl` file included.
- Make sure your GAS version is at least version 2.50.34 or greater.
- To compile and run the test you need to have your environment configured to load the Java Virtual Machine, see Configuring your environment for Ghost Client on page 273.

1. To compile `DemoTests.4gl`, at the command line of the directory where the `GGC` is installed, type the following:

   ```
   cd samples/BDL ; fglcomp DemoTests.4gl
   ```

2. To run the test, at the command line of the directory where the `GGC` is installed, type the following:

   ```
   cd samples/BDL ; fglrun DemoTests http://localhost:6394
   ```

How to compile and run tests

This topic provides information about how to compile and run your own tests.

About this task

The procedure in this topic shows you how to compile and run tests you have developed.

Before you begin:

- It is assumed that you have built two new test java files:
  - One `SessionManager` implementation named, for example, `mySessionManager.java` that will manage the testing session
  - One `Scenario` implementation named, for example, `myScenario.java` that will instantiate real scenarios based on the application launched by the GAS
- It is assumed that your new test java files are in the package `path.to.myTests` and therefore you have placed them in the directory `path/to/myTests`.
- It is assumed your environment is configured to run Java and the `GGC`, see Configuring your environment for Ghost Client on page 273.
- Make sure your GAS version is at least version 2.50.34 or greater.

1. To compile your newly created tests, type the following:

   ```
   javac path/to/myTests/*.java
   ```

2. To run the tests, type the command shown below:

   ```
   java com.fourjs.ggc.Launcher -u http://localhost:6394/ua/r/myApplication -s
   path.to.myTests.mySessionManager
   ```

Generating test scenarios from log file

This topic provides information about how to generate test scenarios from a log of user actions recorded by the GDC or the GWC-JS.

About this task

The procedure in this topic shows you how to generate a Java file set of `SessionManager` and `Scenario` classes from a recorded log file found in your `GGC` package installation. It also shows you how to run integrity tests on the `gwc-demo` applications from the generated `Scenario`.

Before you begin:

- Make sure your GAS version is at least version 2.50.34 or greater.
- Make sure that the `/samples` directory where your `GGC` package is installed has the `StartMenu.log` and `DemoPanel.log` files included.
- It is assumed your environment is configured to run Java and the `GGC`, see Configuring your environment for Ghost Client on page 273.

1. To generate a Java file set using sample log files, type the command shown below:

   ```
   java com.fourjs.ggc.Launcher --write path/to/generate/DemoPanel --log
   samples/DemoPanel.log
   ```

   Note:

   - If some or all directories in the specified path, `path/to/generate/`, do not already exist, they will be created.
   - If the path specified is not absolute, `GGC` will consider its directory as root.
   - The classes generated will be of package `path.to.generate`.
   - If Java files with the same name already exist at the same location, an error will be raised.

Once the Ghost Generator feature has been run, the generated Java files need to be compiled as described in the next step.

2. To compile your newly generated tests, type the following:

   ```
   javac path/to/generate/*.java
   ```

Running the generated SessionManager

About this task:

Once you have compiled the files in the steps above, you can now run them to test the integrity of the `gwc-demo` applications from the generated scenarios as described in the next steps.

Before you begin:

- Make sure that the standalone dispatcher `httpdispatch` see Dispatcher: httpdispatch on page 263 is started and that you can access the GAS demos welcome page, `http://localhost:6394/demos.html`, from your browser.

1. To run the test simulation for GWC-JS clients, type the commands shown below:

```
java com.fourjs.ggc.Launcher -u http://localhost:6394/ua/r/gwc-demo -s
 path.to.generate.StartMenuSessionManager
java com.fourjs.ggc.Launcher -u http://localhost:6394/ua/r/gwc-demo -s
 path.to.generate.DemoPanelSessionManager
```

2. To run the test simulation for GWC-HTML5 clients, type the commands shown below:

```
java com.fourjs.ggc.Launcher -u http://localhost:6394/wa/r/gwc-demo -s
 path.to.generate.StartMenuSessionManager
java com.fourjs.ggc.Launcher -u http://localhost:6394/wa/r/gwc-demo -s
 path.to.generate.DemoPanelSessionManager
```

3. To run the test simulation for GDC clients, type the commands shown below:

```
java com.fourjs.ggc.Launcher -u http://localhost:6394/ja/r/gwc-demo -s
 path.to.generate.StartMenuSessionManager
java com.fourjs.ggc.Launcher -u http://localhost:6394/ja/r/gwc-demo -s
 path.to.generate.DemoPanelSessionManager
```

Automatic discovery of User Agent (adua.xrd)

`adua.xrd` is the configuration file used by the Genero Application Server (GAS) to determine which Output Map to use, based on the User Agent that submits the request for a specific application.

The `auda.xrd` file is located in the `$FGLASDIR/etc` directory.

> **Tip:** Under most circumstances, modification of this file is not necessary.

What is an Output Map?

When the application server needs to render an application, it relies on the application having one or more MAP components defined in its configuration. Each MAP component specifies a RENDERING engine to be used. These MAP elements are child elements of the OUTPUT element, thus the name "Output Map".

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

How an Output Map is chosen

The Output Map chosen for an application is based on the detected browser type, as specified in the `adua.xrd` file.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

The Genero Application Server first identifies the value of the RULE element for the application. The RULE element is defined in (or inherited by) the application configuration.

For example, if the value of the RULE element is UseGWC, then drop into that element. Once inside that element, the type of browser being used to display the application determines which Output Map is used to render the application.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 FOURJS_START_COPYRIGHT(D,2000)
 Property of Four Js*
 (c) Copyright Four Js 2000, 2013. All Rights Reserved.
 * Trademark of Four Js Development Tools Europe Ltd
   in the United States and elsewhere

 This file can be modified by licensees according to the
 product manual.
 FOURJS_END_COPYRIGHT
 -->
<RULE_LIST
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/xrd.xsd">
  <!-- Output Driver Determination (XRD - XML Rule Definition) -->
  <RULE Id="UseGDC">
    <TABLE Id="1" Key="User-Agent">
      <ROW>
        <IN>MSIE</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
      <ROW>
        <IN>GDC</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
    </TABLE>
  </RULE>
  <RULE Id="UseGWC">
    <TABLE Id="1" Key="User-Agent">
      <ROW>
        <IN>GDC</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
      <ROW>
        <ACTION Type="RESULT">DUA_HTML5</ACTION>
      </ROW>
    </TABLE>
  </RULE>
</RULE_LIST>
```

Based on this sample `adua.xrd` file:

If the RULE is UseGDC:

- For MSIE or GDC, the DUA_GDC Output Map is chosen.

If the RULE is UseGWC (for most desktop browsers):

- For the GDC, the DUA_GDC Output Map is chosen.
- Otherwise the DUA_HTML5 Output Map is chosen.

Modify the adua.xrd file to specify custom Output Maps

If you create a custom Output Map, you can modify the `adua.xrd` file to reference your new Output Map.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

For example, you create an Output Map named DUA_HTML5_CUSTOM. You would modify the `adua.xrd` file and identify those browsers where that application should use your custom Output Map. The decision is still based on the browser type that is displaying the application.

```
...
<ROW>
  <ACTION Type="RESULT">DUA_HTML5_CUSTOM</ACTION>
</ROW>
...
```

Specify the Output Map in the application URI

You can force an application to use a specific Output Map by providing the Output Map as an argument in the application URL.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

ADUA Syntax Diagrams

Define valid syntax for the `auda.xrd` file.

- RULE_LIST
- RULE
- TABLE
- ROW

Syntax

```
<RULE_LIST>
  <RULE Id="useId">
    <TABLE Id="numId" Key="keyType">
      <ROW>
```

```
        <IN>inType</IN>
        <OUT>outType</OUT>
        <ACTION Type="actionType">actionName</ACTION>
      </ROW>
      [<ROW> ...]
    </TABLE>
  </RULE>
  [<RULE> ...]
</RULE_LIST>
```

Example

```
<RULE_LIST>
  <RULE Id="UseGDC">
    <TABLE Id="1" Key="User-Agent">
      <ROW>
        <IN>MSIE</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
      <ROW>
        <IN>GDC</IN>
        <ACTION Type="RESULT">DUA_GDC</ACTION>
      </ROW>
    </TABLE>
  </RULE>
</RULE_LIST>
```

- RULE_LIST on page 285
- RULE on page 285
- TABLE on page 285
- ROW on page 286

RULE_LIST
The RULE_LIST element is the main element of an XRD (XML Rule Definition) used by the GAS.

The RULE_LIST element contains the following child element:

1. One or more RULE elements.

RULE
The RULE element defines a unique rule.

The RULE element must specify an Id attribute; this required attribute takes a string value. The identifier (Id) of the rule defines its name, as it is going to be used later, in files such as the GAS configuration file. Valid values for the Id attribute include:

- UseGDC
- UseGWC
- UseGWCMobile (deprecated)

The RULE element contains the following child element:

1. One or more TABLE elements. Each rule uses tables, which can be linked in order to have a complete process.

TABLE
The TABLE element must specify two attributes, an Id attribute and a Key attribute.

- The required Id attribute takes a string value. This attribute provides the table with a unique identifier (Id), which is necessary for linking tables.

- The required Key attribute takes a string value (NMToken). The Key attribute defines what is going to be analyzed. Currently, only two values are supported: **Accept** and **User-Agent**.

The TABLE element contains the following child element:

1. One or more ROW elements. Each table contains on or more rows. Rows are processed sequentially in order of appearance in the XRD file; therefore rows are not named.

ROW
The ROW element contains the required ACTION element, along with the optional IN and OUT elements.

The ROW element may contain the following child elements:

1. Zero or more IN elements (optional). The IN element takes a string value, and specifies a string or substring that must be in the HTTP header referenced by the TABLE key attribute.
2. Zero or more OUT elements (optional). The OUT element takes a string value, and specifies a string or substring that must not be in the HTTP header referenced by the TABLE key attribute (they must be OUT).
3. One ACTION element (required). The ACTION element must specify a Type attribute and takes a required string (NMToken) value. If the string matches the IN and OUT rules (i.e., the IN and OUT conditions are met), this element defines the action to perform. Valid values for this element type are:
   - GOTO_TABLE - Jumps to the specified table.
   - RESULT - Sends the result.

adua.xrd usage example

This topic explains an automatic user agent discovery configuration example.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Suppose you want to use the DUA_GDC output map for Internet Explorer browsers and DUA_XXX for all other user agents. To achieve this, you have to configure your application to support both Output Maps.

In your GAS configuration file (`as.xcf`), add an OUTPUT element with a rule identified by `UseAllOutputDriver`, defining the rendering and theme for two MAP elements identified by the DUA_GDC and DUA_XXX names:

```
<APPLICATION Id="test" Parent="defaultwa">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
  <OUTPUT Rule="UseAllOutputDriver">
    <MAP Id="DUA_XXX" Allowed="TRUE">
      <RENDERING Using="cpn.rendering.wa"/>
      <THEME Using="cpn.theme.default.xxx"/>
    </MAP>
    <MAP Id="DUA_GDC" Allowed="TRUE">
      <RENDERING Using="cpn.rendering.wa"/>
      <THEME Using="cpn.theme.default.gdc"/>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

In the `adua.xrd` file, define the `UseAllOutputDriver` rule to associate a User-Agent type to a MAP element defined in the as.xcf file:

```
<RULE Id="UseAllOutputDriver">
  <TABLE Id="1" Key="User-Agent">
    <ROW>
      <IN>MSIE</IN>
      <ACTION Type="RESULT">DUA_GDC</ACTION>
    </ROW>
    <ROW>
      <ACTION Type="RESULT">DUA_XXX</ACTION>
    </ROW>
  </TABLE>
</RULE>
```

With this example, if the User-Agent value contains `MSIE`, GDCAX will be used (identified by DUA_GDC); otherwise the mapping that corresponds to the DUA_XXX will be used.

GAS Predefined resources

Predefined resources for the Genero Application Server can be general or front-end specific.

- GAS predefined resources overview on page 287
- Common GAS predefined resources on page 288

GAS predefined resources overview

This topic describes the syntax used in the GAS configuration file for predefined resources (i.e. variables). .

While most resources are defined in the GAS configuration file, **predefined resources** can not be explicitly defined as they are automatically replaced depending on the path of the application you are executing. Therefore, in order that they are available for use, they are predefined as resources using an XML tagging mechanism, which is described in this topic.

Syntax 1

```
$(resource-name)
```

1. *resource-name* is the name of a resource defined in the GAS configuration file.

Syntax 2

```
<tag gwc:tpl-attribute="tpl-value" [...]>...</tag>
```

1. *tag* is an HTML tag.
2. *tpl-attribute* is an attribute.
3. *tpl-value* is the value of the attribute.

Usage example

For example, we can define the path to an application's web component directory as follows:

```
<WEB_COMPONENT_DIRECTORY>$(application.path)/webcomponents</
WEB_COMPONENT_DIRECTORY>
```

Common GAS predefined resources

These Genero Application Server predefined resources are available for any Front End .

These resources include:

Table 60: Common GAS predefined resources

Predefined Resource	Description
`application.id`	Application identifier in as.xcf or, in the case where an external application configuration file is used, the name of the .xcf file.
`application.group`	Application Group name.
`application.name`	Application name.
`application.path`	Handles the path of the application (the path defined in the /APPLICATION/EXECUTION/PATH element). For example, we can define the resource domain "Image" as follows: `<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.picture"> ... <PATH Id="Image" Type="APPSERVER"> $(res.path.pic);$(application.path) </PATH> ... </WEB_APPLICATION_PICTURE_COMPONENT>`
`connector.uri`	Will be replaced by the URL path to the connector. For more information on application URIs See Table 6: Explanation of URI syntax options on page 50. For example, if the URL is: `http://localhost/gas/ua/r/demo?Arg=val1&Arg=val2` The value of `connector.uri` will be `/gas`.
`server.version`	Will be replaced by GAS software version.
`configuration.filepath`	The absolute path of the configuration file used to start the GAS.

GAS Configuration Reference

These topics provide reference details for Genero Application Server configuration.

GAS configuration file

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

To create a new Genero Application Server configuration file, create a copy of the default and make modifications in the copy. Use the `-f` option to specify which configuration file to use when starting the Genero Application Server. If you do not specify a specific file, the default (`as.xcf`) is used.

The configuration file is an XML file. The root element in the Application Server configuration file is the `CONFIGURATION` element. A configuration file will have one `CONFIGURATION` element which serves as the global configuration element. There are no attributes available for the `CONFIGURATION` element. The `CONFIGURATION` element contains a single child element, the `APPLICATION_SERVER` element. The Application Server configuration starts with this element. See GAS configuration file hierarchy on page 290 for a full list of valid elements and their place in the file hierarchy.

When you make a change to the GAS configuration file, you must restart the dispatcher before the changes take effect. Depending on the dispatcher, you may or may not have the applications (e.g. proxies) stopped; some of the dispatchers can recover the application after the restart and continue where the application left off. For more information on restarting the GAS dispatcher, refer to the dispatcher-specific help topics:

- Restarting the standalone dispatcher
- Restarting the ISAPI dispatcher
- Restarting the FastCgi dispatcher
- Restarting the J2EE dispatcher on page 93

Application configuration files

Configure Genero Web applications and services using a subset of the elements provided for the Genero Application Server configuration.

The application configuration details can either be:

- added to the Genero Application Server configuration file.
- provided in application-specific configuration files. Each application would have its own configuration file, written in XML.

Elements in an application's configuration

The top-most element for application configuration is the `APPLICATION` element. The elements allowed, and how they are used, depend on whether the application is a web application (GWC) or a web service (GWS). For a full list of valid elements and their place in the file hierarchy, see the appropriate topic:

- GWC configuration file hierarchy on page 294
- GWS configuration file hierarchy on page 295

Use application-specific configuration files

When you add application configuration details to the Genero Application Server configuration file, you must restart the dispatcher before the changes take effect. You must also take care during upgrades, that the GAS configuration file is not overwritten and you lose your changes.

When you create a new application configuration file, or make changes to an existing application configuration file, you do not need to restart any dispatchers. All applications started after you save your changes use the changed settings. The files are easily archived apart from the GAS configuration file, and not overwritten by newer versions during upgrades.

For these reasons, it is recommended that you use application-specific configuration files.

Configuration file hierarchies

Configuration files define the initial configuration settings for the GAS and for individual applications. Each configuration file is made up of elements in a predefined hierarchy.

GAS configuration file hierarchy

A listing of elements valid in the Genero Application Server configuration file, shown hierarchically.

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

- CONFIGURATION
  - APPLICATION_SERVER
    - RESOURCE_LIST
      - PLATFORM_INDEPENDENT
        - RESOURCE
      - WNT
        - RESOURCE
      - UNX
        - RESOURCE
    - COMPONENT_LIST
      - WEB_APPLICATION_EXECUTION_COMPONENT
        - ENVIRONMENT_VARIABLE
        - PATH
        - DVM
        - WEB_COMPONENT_DIRECTORY on page 361
        - MODULE
        - PARAMETERS
          - PARAMETER
        - ACCESS_CONTROL
          - ALLOW_FROM
        - DELEGATE
      - SERVICE_APPLICATION_EXECUTION_COMPONENT
        - ENVIRONMENT_VARIABLE
        - PATH
        - DVM
        - MODULE
        - PARAMETERS
          - PARAMETER
        - ACCESS_CONTROL
          - ALLOW_FROM
        - DELEGATE
        - POOL
          - START
          - MIN_AVAILABLE
          - MAX_AVAILABLE

- INTERFACE_TO_DVM
  - ADDRESS
- LOG
  - OUTPUT
  - FORMAT
  - CATEGORIES_FILTER
  - RAW_DATA
- MONITOR
  - ALLOW_FROM
- FILE_TRANSFER
  - TIMEOUT
- APPLICATION_LIST
  - GROUP
  - APPLICATION
    - DESCRIPTION
      - SHORT
      - LONG
    - RESOURCE
    - EXECUTION
      - ENVIRONMENT_VARIABLE
      - PATH
      - DVM
      - MODULE
      - PARAMETERS
        - PARAMETER
      - ACCESS_CONTROL
        - ALLOW_FROM
      - DELEGATE
      - WEB_COMPONENT_DIRECTORY on page 361
    - UA_OUTPUT
      - PROXY
      - PUBLIC_IMAGEPATH
      - GWC-JS on page 319
      - TIMEOUT
        - USER_AGENT
        - REQUEST_RESULT
        - DVM_AVAILABLE
        - DVM_PINGTIMEOUT
    - OUTPUT
      - HTTP_HEADER
      - HTTP_COOKIES
        - HTTP_COOKIE
          - CONSTANT
          - VARIABLE
      - MAP

- PROXY
- PICTURE
  - PATH
- TIMEOUT
  - USER_AGENT
  - REQUEST_RESULT
  - DVM_AVAILABLE
  - DVM_PINGTIMEOUT
- RENDERING
  - OUTPUT_DRIVER
  - XML_DECLARATION
  - HTTP_RESPONSE_ENCODING
  - HTTP_REQUEST_ENCODING
  - MIME_TYPE
  - DOC_TYPE
    - NAME
    - EXTERNAL_ID
    - SYSTEM_ID
- THEME
  - BOOTSTRAP
  - TEMPLATE
  - SHORTCUT
  - SNIPPET

- SERVICE_LIST
  - GROUP
  - APPLICATION
    - DESCRIPTION
      - SHORT
      - LONG
    - RESOURCE
    - PROXY
    - EXECUTION
      - ENVIRONMENT_VARIABLE
      - PATH
      - DVM
      - MODULE
      - WEB_COMPONENT_DIRECTORY on page 361
      - PARAMETERS
        - PARAMETER
      - ACCESS_CONTROL
        - ALLOW_FROM
      - DELEGATE
      - POOL
        - START
        - MIN_AVAILABLE

GWC configuration file hierarchy
A listing of the elements valid for a Genero Web Client (GWC) application configuration file, shown hierarchically.

This is a complete listing of all of the elements available for a Genero Web Client external application configuration file. It is a subset of the elements available for the Genero Application Server configuration file. The `APPLICATION` element is the root element for that application's configuration. For a Web application, the `APPLICATION` element is the child of the `APPLICATION_LIST` element in the Genero Application Server configuration file.

The elements are described in detail; select an element name to be taken to the topic discussing that element.

        - VARIABLE
    - MAP
        - PROXY
        - PICTURE
            - PATH
        - TIMEOUT
            - USER_AGENT
            - REQUEST_RESULT
            - DVM_AVAILABLE
            - DVM_PINGTIMEOUT
        - RENDERING
            - OUTPUT_DRIVER
            - XML_DECLARATION
            - HTTP_RESPONSE_ENCODING
            - HTTP_REQUEST_ENCODING
            - MIME_TYPE
            - DOC_TYPE
                - NAME
                - EXTERNAL_ID
                - SYSTEM_ID
        - THEME
            - BOOTSTRAP
            - TEMPLATE
            - SHORTCUT
            - SNIPPET

GWS configuration file hierarchy
A listing of available elements valid for a Genero Web Services (GWS) application configuration file, shown hierarchically.

This is a complete listing of all of the elements available for a Genero Web Services external application configuration file. It is a subset of the elements available for the Genero Application Server configuration file. The `APPLICATION` element is the root element for that application's configuration. For a Web service, the `APPLICATION` element is the child of the `SERVICE_LIST` element in the Genero Application Server configuration file.

The elements are described in detail; select an element name to be taken to the topic discussing that element.

- APPLICATION
    - DESCRIPTION
        - SHORT
        - LONG
    - RESOURCE
    - PROXY
    - EXECUTION
        - ENVIRONMENT_VARIABLE
        - PATH
        - DVM
        - MODULE

Configuration file elements

An alphabetical listing of the elements used across the Genero Application Server configuration files.

ACCESS_CONTROL

The `ACCESS_CONTROL` element specifies access from a list of IP allowed to access applications or services. Access can be globally denied or allowed by keywords (*NOBODY*, *ALL*) .

By default, an application or a service is **not** accessible by anyone. It needs to be explicitly configured with the `ALLOW_FROM` element.

Syntax

```
<ACCESS_CONTROL>
[ <ALLOW_FROM> ip_address </ALLOW_FROM> [...] ]
</ACCESS_CONTROL>
```

where *ip_address* is a valid IPv4 or IPv6 address. For IPv4 it can be a complete IP address or a network address (ending with a dot)..

> **Important:** Depending on the network configuration, it is not always possible to get the actual client IP address. If there is a proxy server between the client and the server, for example, the client IP address seen by the GAS may be the address from the proxy server.

Child elements

- Zero or more `ALLOW_FROM` on page 300 elements.

Example

```
<ACCESS_CONTROL>
```

```
  <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
  <ALLOW_FROM>10.</ALLOW_FROM>
  <ALLOW_FROM>192.168.</ALLOW_FROM>
  <ALLOW_FROM>fdbd:2768:c176:1::323a</ALLOW_FROM>
</ACCESS_CONTROL>
```

In this example, an application or a service is reachable from the localhost (127.0.0.1), and all IP addresses that begin with "192.168." or "10.". The consecutive colons (::) notation in "fdbd:2768:c176:1::323a" shows an example of a collapsed IPv6 address, where the colons represent four successive 16-bit blocks that contain zeros.

Example configuring access control for demo applications

The default deployment of the demo application is specified by the resource *res.access.control*, which is defined with the value *NOBODY* by default.

> **Note:** Access control rules will be ignored by the standalone dispatcher (httpdispatch).
>
> **Important:** The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.

To allow access from the localhost, in the GAS configuration file (default *%FGLASDIR%*/`etc/as.xcf`) you need to change the application element for `gwc-demo` from:

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

To:

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

Parent elements

This element is a child of one of the following elements:
SERVICE_APPLICATION_EXECUTION_COMPONENT on page 344,
WEB_APPLICATION_EXECUTION_COMPONENT on page 356,EXECUTION (for an application) on page 315, EXECUTION (for a service) on page 316

ADDRESS

The ADDRESS *element specifies* the name or IP address of the machine where the Genero Application Server runs. This address is used by the DVM to set (build) the **FGLSERVER** environment variable.

Usage examples

```
<ADDRESS>app_server</ADDRESS>
<ADDRESS>192.127.45.17</ADDRESS>
<ADDRESS>zeus</ADDRESS>
```

> **Note:** It is not recommended to use **localhost** or **127.0.0.1** because the license server requires the real address of the machine to check the licenses, and if your machine is not well configured, a bad address is returned to the license server that will then refuse to start a new DVM.

Parent elements

This element is a child of one of the following elements: INTERFACE_TO_DVM on page 325

ALLOW_FROM

The ALLOW_FROM element specifies access control rules to allow access from hosts. This can be specified by a host's IP address, or access can either be globally denied or allowed by the following keywords:

- *NOBODY*
- *ALL*

Syntax

```
<ALLOW_FROM>NOBODY</ALLOW_FROM>[...]
```

```
<ALLOW_FROM>ALL</ALLOW_FROM>[...]
```

```
<ALLOW_FROM>ip_address</ALLOW_FROM>[...]
```

where *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)..

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

where *res.access.control* is a resource defined with the value *NOBODY* by default. The default GAS deployment for demo applications and MONITOR on page 328 references this resource. For more information on configuring access control, see Example configuring access control for demo applications on page 299

> **Note:** Access control rules will be ignored by the standalone dispatcher (httpdispatch).
>
> **Important:** The standalone GAS is for development only, provided to simplify your development setup and configuration. For deployment and production systems, you must include a Web server.

How access control rules are applied

If more than one access control rule (i.e. ALLOW_FROM element) is specified, the following describes the order rules are applied:

1. *NOBODY* always gets the highest priority regardless of other rules, which means that nobody gets access.
2. *ALL* gets priority over specific IP addresses when the keyword *NOBODY* is **not** provided, which means that access is allowed to all.
3. If neither the keyword *NOBODY* nor *ALL* is provided, the remaining access control rules are applied.

Parent elements

This element is a child of one of the following elements: ACCESS_CONTROL on page 298, MONITOR on page 328

APPLICATION (for an application)

An APPLICATION element defines an application. Attributes for this element include:

- **Id** (required for applications defined within the Genero Application Server configuration file; optional for applications defined in an external application configuration file) - A string used to uniquely identify this application configuration element. The Id specified is compared to the application name in the request.
- **Parent** - A string that identifies the parent application, or the application from which this application will inherit its default configuration/settings.
- **Abstract** - Defines whether this application configuration element is an abstract application. It expects a boolean string; the valid values for this type are "TRUE" and "FALSE". An Abstract application can not instantiate Virtual Machines. Abstract configurations are used purely in the scope of future inheritance of the configuration for other Web applications. Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.
- **mode** - When set to "sticky", defines a Web service as a sticky Web service.

Child elements

The APPLICATION element may contain the following elements:

1. Zero or one DESCRIPTION element.
2. Zero or more RESOURCE elements.
3. Zero or one EXECUTION elements.
4. Zero or one UA_OUTPUT on page 354 elements. This element is used for all UI applications, with the exception of applications using GWC for HTML5.
5. Zero or one OUTPUT elements. This element is deprecated; only used for UI applications using GWC for HTML5.

Example

```
<APPLICATION Id="gwc-demo" Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
  <UA_OUTPUT> </UA_OUTPUT>
</APPLICATION>
```

For more information, see Configuring applications on GAS on page 95.

Parent elements

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

When used in the GAS configuration file, this element is a child of one of the following elements: APPLICATION_LIST on page 302.

APPLICATION (for a service)

An APPLICATION element defines an application. For each Web Service you wish to make accessible through the Genero Application Server, you must create an APPLICATION element. Attributes for this element include:

- **Id** (required for Web services applications defined within the Genero Application Server configuration file; optional for applications defined in an external application configuration file) - A string to uniquely identify this Web service application configuration element. The Id specified is compared to the application name in the request.
- **Parent** - A string that identifies the parent application, or the application from which this application will inherit its default configuration/settings.

- **Abstract** - Defines whether this application configuration element is an abstract application. It expects a boolean string; the valid values for this type are "TRUE" and "FALSE". An Abstract application can not instantiate Virtual Machines. Abstract configurations are used purely in the scope of future inheritance of the configuration for other Web services applications. Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.
- **mode** - When set to "sticky", defines a Web service as a sticky Web service.

  Note: With the release of Genero 2.0, Web services are named applications as they host several Web services in one DVM.

Child elements

When you define a Web service application, you can specify the following elements:

1. Zero or one `DESCRIPTION` element.
2. Zero or more `RESOURCE` elements.
3. Zero or one `EXECUTION` element.
4. Zero or more `TIMEOUT` elements.

Example

```
<APPLICATION Id="webapp" Parent="abswebapp">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>webapp.42r</MODULE>
  </EXECUTION>
  <TIMEOUT> </TIMEOUT>
</APPLICATION>
```

Parent elements

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

When used in the GAS configuration file, this element is a child of one of the following elements:
`SERVICE_LIST` on page 346

APPLICATION_LIST

For each application to be serviced by the Genero Application Server, you must provide the details for that application in either the Genero Application Server configuration file or in an external application server configuration file.

For information about the general process of defining applications and groups, refer to the Configuring applications on GAS on page 95 section of this manual.

The `APPLICATION_LIST` element provides a list of groups and Web applications (for Web applications defined within the Genero Application Server configuration file).

Syntax

```
<APPLICATION_LIST>
  [ <GROUP element.> ] [...]
  [ <APPLICATION ....> ] [...]
  ...
</APPLICATION_LIST>
```

Child elements

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

1. Zero or more GROUP elements.
2. Zero or more APPLICATION elements.

Example

```
<APPLICATION_LIST>
  <GROUP Id="appgroup">/home/appgroup</GROUP>
  <APPLICATION Id="gwc-demo" Parent="defaultgwc">
   <EXECUTION>
     <PATH>$(res.path.fgldir.demo)</PATH>
     <MODULE>demo.42r</MODULE>
   </EXECUTION>
 </APPLICATION>
</APPLICATION_LIST>
```

Parent elements

This element is a child of one of the following elements: APPLICATION_SERVER on page 303

APPLICATION_SERVER

This element acts as a parent container for all Genero Application Server configuration elements.

The APPLICATION_SERVER element does not support any attributes.

Syntax

```
<APPLICATION_SERVER>
  <RESOURCE_LIST>...</RESOURCE_LIST>
  <COMPONENT_LIST>...</COMPONENT_LIST>
  <INTERFACE_TO_CONNECTOR>...</INTERFACE_TO_CONNECTOR>
  <INTERFACE_TO_DVM>...</INTERFACE_TO_DVM>
  <LOG>...</LOG>
  <MONITOR>...</MONITOR>
  <FILE_TRANSFER>...</FILE_TRANSFER>
  <APPLICATION_LIST>...</APPLICATION_LIST>
  <SERVICE_LIST>...</SERVICE_LIST>
</APPLICATION_SERVER>
```

Child elements

The APPLICATION_SERVER element contains the following child elements:

- One RESOURCE_LIST element, containing a list of resources.
- One COMPONENT_LIST element, containing a list of components.
- One INTERFACE_TO_CONNECTOR element, specifying the interface between the Genero Application Server (GAS) and the GAS Connector. The connector is either the ISAPI, FastCGI, or J2EE extension, or the user agent through direct connection.
- One INTERFACE_TO_DVM element, specifying the interface to the Dynamic Virtual Machine.
- Zero or more LOG elements, specifying the type of information that is logged and where it is logged to.
- Zero or one MONITOR element, specifies from which machines the monitor URL is accessible.
- Zero or more FILE_TRANSFER elements, specifying the directory where files are stored while being transferred between the front-end machine and the DVM.
- Zero or one APPLICATION_LIST element, containing a list of applications.
- Zero or one SERVICE_LIST element, containing a list of Web Services.

Example

```
<CONFIGURATION
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfas.xsd">
  <APPLICATION_SERVER>
    ...
  </APPLICATION_SERVER>
</CONFIGURATION>
```

Parent elements

This element is a child of one of the following elements: CONFIGURATION on page 310

AUTO_LOGOUT

Selects the AUTO_LOGOUT mechanism to be used for an application. It takes the attribute Using, where you can reference a predefined AUTO_LOGOUT_COMPONENT on page 305 (to inherit the auto logout parameters of that component).

If the AUTO_LOGOUT is set, the application will get a log out message after the time in seconds specified by the TIMEOUT (for auto logout) on page 353 element has elapsed.

- For GWC-JS applications, you will get a log out page.
- For GDC applications, you will get a pop-up window.

By default the ending page or pop-up window shows the following message when auto logout occurs : You have been logged out.

> **Note:** The auto logout feature is not supported on the legacy Genero Web Client applications using the HTML5 theme.

Syntax

```
<AUTO_LOGOUT Using="auto_logout_component_Id"></AUTO_LOGOUT>
```

Usage example - using an auto logout component

When your applications reference the default application defaultwa, they inherit its AUTO_LOGOUT settings, which use the default AUTO_LOGOUT_COMPONENT identified by the **Id** value "cpn.wa.autologout", as shown in the examples from the GAS configuration file.

```
<APPLICATION Id="defaultwa" Abstract="TRUE">
   <!-- This is the "default" application.
    It is not used directly: it is used for defining a "root" application.
 -->
   <EXECUTION Using="cpn.wa.execution.local"/>
   <AUTO_LOGOUT Using="cpn.wa.autologout"/>
   ...
</APPLICATION>
```

```
<COMPONENT_LIST>
   ...
   <AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
      <TIMEOUT>0</TIMEOUT>
      <!--COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND-->
   </AUTO_LOGOUT_COMPONENT>
   ...
</COMPONENT_LIST>
```

> **Note:** The default timeout duration set to 0 seconds means the auto logout is ignored and applications keep running.

Usage example - defining the auto logout directly
If you want to set auto logout directly in your application configuration file (`xcf`), you can add an `AUTO_LOGOUT` element.

```
<APPLICATION Parent="defaultwa">
    <EXECUTION>
    <PATH>my_app_dir</PATH>
    <MODULE>my_module</MODULE>
 </EXECUTION/>
    <AUTO_LOGOUT>
       <TIMEOUT>30</TIMEOUT>
       <!--COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND-->
    </AUTO_LOGOUT>
    ...
 </APPLICATION>
```

Child elements

There are no child elements.

Parent elements

This element is a child of the following element: `APPLICATION (for an application)` on page 301

Auto logout and child applications

As long as the parent application or any of its child applications have user activity, the auto logout is not triggered.

> **Important:** If an application has child applications running when the `AUTO_LOGOUT` is triggered, the following describes the logout process:
>
> - An auto logout ending page or pop-up window will be generated and returned for all child applications.
> - The proxy will stay alive until the last child application has shown its ending page before closing the `fglrun` application.
>
> **Note:** The order that applications close can not be determined by the `AUTO_LOGOUT` process as it depends mainly on when the auto logout is initiated in the front-end client. If the order your applications close at auto logout is important, you will need to handle this in your application's code.

AUTO_LOGOUT_COMPONENT

The `AUTO_LOGOUT_COMPONENT` element creates an application auto logout component, which defines a mechanism for triggering and handling auto-logout events. It takes an attribute **Id**, which specifies the unique identifier for a set of auto-logout definitions. It is this unique identifier that is referenced by an application, providing it with a base set of auto-logout values.

Syntax

```
<AUTO_LOGOUT_COMPONENT Id="component_unique_identifier">
    <TIMEOUT>timeoutSeconds</TIMEOUT>
        <!--COMMAND is an optional configuration element -->
    <COMMAND Timeout="timeoutSeconds"> commandScript </COMMAND>
</AUTO_LOGOUT_COMPONENT>
```

Child elements

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

1. Zero or one TIMEOUT (for auto logout) on page 353 element.
2. Zero or one COMMAND (for auto logout) on page 309 element (Optional).

Usage example default AUTO_LOGOUT_COMPONENT in the GAS configuration file

```
<AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
    <TIMEOUT>0</TIMEOUT>
</AUTO_LOGOUT_COMPONENT>
```

In this example, the **Id** value - **cpn.wa.autologout** - defines the default AUTO_LOGOUT_COMPONENT in the GAS configuration file. When your applications reference the default application, defaultwa, they inherit the settings defined by its AUTO_LOGOUT on page 304 element as shown in the example:

```
<AUTO_LOGOUT Using="cpn.wa.autologout"/>
```

Usage example 2

```
<AUTO_LOGOUT_COMPONENT Id="cpn.wa.autologout">
    <TIMEOUT>30</TIMEOUT>
    <COMMAND Timeout="20">auto-logout-allowed.sh</COMMAND>
</AUTO_LOGOUT_COMPONENT>
```

In this example:

- The TIMEOUT element is set for 30 seconds. When no user activity is detected, the DVM waits for this timeout period to elapse before the auto logout task is performed.
- The COMMAND element's *Timeout* attribute is set to 20 seconds to allow a command to be run that checks if the auto logout is allowed, see COMMAND (for auto logout) on page 309

Parent elements
This element is a child of one of the following elements: COMPONENT_LIST on page 308.

BOOTSTRAP (GWC-HTML5)

Selects the bootstrap template to be used in this theme.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Syntax

```
<BOOTSTRAP Id="bootstrapId"> bootstrapPath </BOOTSTRAP>
```

Syntax notes

1. *bootstrapId* is the unique identifier for this element.
2. *bootstrapPath* is the path to the bootstrap template to be used in this theme.

Usage example

```
<WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.html5.gwc">
```

```
  <BOOTSTRAP Id="_default">$(res.path.tpl.html5)/bootstrap.xhtml</BOOTSTRAP>
  <TEMPLATE Id="_default">$(res.path.tpl.html5)/main.xhtml</TEMPLATE>
  <SNIPPET Id="UIFrame">$(res.path.tpl.html5)/UIFrame.xhtml</SNIPPET>
  ...
</WEB_APPLICATION_THEME_COMPONENT>
```

Parent elements
This element is a child of one of the following elements: THEME on page 351, WEB_APPLICATION_THEME_COMPONENT on page 359

CATEGORIES_FILTER

The CATEGORIES_FILTER element specifies the type of log messages to be captured. Specify multiple values by listing multiple filter names, separated by spaces.

This element is optional. If the CATEGORIES_FILTER element is omitted, no categorized messages are logged.

Table 61: Category filters

Filter Names	Description
DEBUG	Log internal information for debugging. **Important:** Do not set unless requested by your support center.
WARNING	Log warning messages.
ERROR	Log error information.
MUTEX	Log multithreading information.
SESSION	Log session management information.
GAS	Log information about the Genero Application Server version, build and package.
LOG	Display log parameters.
CONFIGURATION	Log configuration management information.
DEPRECATED	Log warnings if a deprecated function is used in a template.
PROCESS	Log Dynamic Virtual Machine data, error, any process handled by the Genero Application Server.
VM	Log communications with the Dynamic Virtual Machine.
FT	Log information about file transfer.
ACCESS	Log summary information about the requests handled by the server.
HTTP	Log http requests (interaction with the User Agent).
SOCKET	Log information concerning sockets (creation, connection, communication, and so on).
TASK	Extracts the time of all the things done by the GAS (a technical attribute).
TEMPLATE	Information about TEMPLATE template processing.
TIMER	Information about timers creation, destruction and expiration.

Filter Names	Description
WA	Context of the Web applications.
ALL	Activate all categories excepted DEBUG.

Usage example

This example would be valid for the CATEGORIES_FILTER element:

```
<CATEGORIES_FILTER>ERROR WARNING</CATEGORIES_FILTER>
```

Generate a detailed daily log file

If you encounter an issue, you can send a detailed log file to your support center.

To create the detailed daily log file, use the ALL pseudo category as the category filter.

> **Important:** Only use the default categories (GAS ACCESS PROCESS DEPRECATED ERROR WARNING) when in production. Categories such as VM or ALL are for debugging and should only be set for a short period of time, as they generate many log entries. Non-standard categories should be used with care. The support center will tell you when you should set ALL for your categories filter.

Example

```
<LOG>
  <OUTPUT Type="DAILYFILE">/work/tmp/gas</OUTPUT>
  <FORMAT>date time relative-time process-id thread-id contexts
   event-type event-params<FORMAT>
  <CATEGORIES_FILTER>ALL</CATEGORIES_FILTER>
</LOG>
```

This example generates various log files in the /work/tmp/gas directory, depending on the dispatcher and the application or service run.

Parent elements

This element is a child of one of the following elements: LOG on page 326

COMPONENT_LIST

Within the Genero Application Server configuration file, you can define various application components. Components are sets of preset variables, and are used by applications that share common configurations. Within the COMPONENT_LIST element, you specify components to be available for use by applications. When defining an application, you can then reference the component by its unique Id.

Syntax

```
<CONFIGURATION>
  <APPLICATION_SERVER>
    ...
    <COMPONENT_LIST>
      EXECUTION (for an application) on
 page 315 [...]
      TIMEOUT (for an application) on
 page 352 [...]

  autoLogoutComponent [...]
```

```
   serviceTimeoutComponent [...]

   pictureComponent [...]

   renderingComponent [...]

   httpCookiesComponent [...]
       themeComponent [...]
     </COMPONENT_LIST>
     ...
   </APPLICATION_SERVER>
</CONFIGURATION>
```

Syntax notes

1. The COMPONENT_LIST element does not support any attributes.
2. Components are grouped by type. Each component type is discussed in its own section of this manual.

Child elements

A COMPONENT_LIST element may contain the following child elements:

- Zero to many WEB_APPLICATION_EXECUTION_COMPONENT elements.
- Zero to many SERVICE_APPLICATION_EXECUTION_COMPONENT elements.
- Zero to many WEB_APPLICATION_TIMEOUT_COMPONENT elements.
- Zero to many AUTO_LOGOUT_COMPONENT on page 305elements.
- Zero to many SERVICE_APPLICATION_TIMEOUT_COMPONENT elements.
- Zero to many WEB_APPLICATION_PICTURE_COMPONENT elements.
- Zero to many WEB_APPLICATION_RENDERING_COMPONENT elements.
- Zero to many WEB_APPLICATION_HTTP_COOKIES_COMPONENT elements.
- Zero to many WEB_APPLICATION_THEME_COMPONENT elements.

Parent elements

This element is a child of one of the following elements: APPLICATION_SERVER on page 303

COMMAND (for auto logout)

The COMMAND is an optional configuration element of AUTO_LOGOUT_COMPONENT on page 305 that provides a mechanism for the Genero Application Server to override an application's auto logout.

Syntax

```
<COMMAND Timeout="timeoutSeconds"> commandScript </COMMAND>
```

Syntax notes

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

Usage example

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

The command determines whether the auto logout takes place based on the value of an exit code it returns, or whether it times out:

1. If the command's exit code is zero, the auto logout is ignored and the application keeps running.

2. If the command's exit code is **not** zero, the auto logout is performed.

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

Usage Scenario

A typical use of the `COMMAND` option might be to check when the auto logout process is allowed. The external command could be used, for example, to only allow auto logout between the hours of 21.00 pm and 08.00 am, i.e. there would be no auto logout during work hours.

Parent elements

This element is a child of the following element: `AUTO_LOGOUT_COMPONENT` on page 305

CONFIGURATION

This element is the starting point for the Genero Application Server configuration.

The `CONFIGURATION` element does not support any attributes.

Syntax

```
<CONFIGURATION>
  <APPLICATION_SERVER>...</APPLICATION_SERVER>
</CONFIGURATION>
```

Child elements

The `CONFIGURATION` element contains a single child element:

- One `APPLICATION_SERVER` element.

Example

```
<CONFIGURATION
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.50/cfas.xsd">
  <APPLICATION_SERVER>...</APPLICATION_SERVER>
</CONFIGURATION>
```

Parent elements

The `CONFIGURATION` element is the top-most element of the GAS configuration file. It has no parent element.

CONSTANT

Zero or one `CONSTANT`. Define a constant and optionally its value.

Syntax

```
<CONSTANT Id="cstId"> cst </CONSTANT>
```

Syntax notes

1. *cstId* is the constant name
2. *cst* is the constant value.

Example

```
<CONSTANT Id="constant1">A value</CONSTANT>
```

Parent elements

This element is a child of one of the following elements: HTTP_COOKIE on page 321

DELEGATE

The DELEGATE element specifies the Genero REST service in charge of all starting requests of the form / wa/r, /ja/r and /ws/r. It takes an attribute called service where you define the group and name of the Genero REST service.

Syntax

```
<DELEGATE service="serName">
 [ <AnyTag1> anyValue </AnyTag1> ]
 [ <AnyTag2> anyValue </AnyTag2> ]
</DELEGATE>
```

All child elements are optional and are passed as parameters to the REST service if present. See How to implement delegation on page 105.

When working with a single sign-on solution, child elements of the DELEGATE element will be specific to the identity provider (IdP). You will need to add the appropriate tags to work with your IdP. These tags should be documented by your IdP.

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_EXECUTION_COMPONENT on page 356, SERVICE_APPLICATION_EXECUTION_COMPONENT on page 344, EXECUTION (for an application) on page 315, EXECUTION (for a service) on page 316

DESCRIPTION

The DESCRIPTION element allows a short and a long description to be associated with an application definition.

Syntax

```
<DESCRIPTION>
[ <SHORT> shortDescription </
SHORT> ]
[ <LONG> longDescription </LONG> ]
</DESCRIPTION>
```

Child elements

The DESCRIPTION element may contain the following child elements:

1. Zero or one SHORT element.
2. Zero or one LONG element.

Example

```
<DESCRIPTION>
  <SHORT>A short description</SHORT>
  <LONG>A long description</LONG>
</DESCRIPTION>
```

Parent elements

This element is a child of one of the following elements: APPLICATION (for an application) on page 301, APPLICATION (for a service) on page 301

DOC_TYPE

The DOC_TYPE element can be be used to override the document type declaration (DTD) in the main template.

Syntax

```
<DOC_TYPE>
  [ <NAME> name </NAME>]
  [ <EXTERNAL_ID> externalId </EXTERNAL_ID>]
  [ <SYSTEM_ID> systemId </SYSTEM_ID>]
</DOC_TYPE>
```

Child elements

The DOC_TYPE element may contain the following child elements:

- One NAME element.
- Zero or one EXTERNAL_ID element.
- Zero or one SYSTEM_ID element.

Usage example

```
<DOC_TYPE>
  <NAME>HTML</NAME>
  <EXTERNAL_ID>-//W3C//DTD HTML 4.01//EN</EXTERNAL_ID>
  <SYSTEM_ID>http://www.w3.org/TR/html4/strict.dtd</SYSTEM_ID>
</DOC_TYPE>
```

In this example, the DTD in the main template would be overwritten with:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
```

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_RENDERING_COMPONENT on page 358, RENDERING on page 339

DOCUMENT_ROOT

The DOCUMENT_ROOT element specifies root directories that determine filesystem paths to serve files. The default document root directory (*res.path.docroot*) is *$FGLASDIR*/web. This directory is the default GWC-JS bootstrap template path which is searched for CSS, JavaScript™, demos.html, and other files that are specifically served by the Application Server, see GWC-JS on page 319.

> **Note:** When a Web server is included in the solution architecture, the GAS's document root directory is separate to the Web server's, which has its own document root - often called htdocs for the Apache Web server or C:\Inetpub\wwwroot for IIS. Files located in the Web server document root are served by the Web server, while files located in the GAS DOCUMENT_ROOT are served by the GAS, i.e. the dispatcher.

For the GAS, other root directories can be specified as required. For example, the resource path *res.path.docroot.user*, which is not defined in the GAS configuration file by default, can be set as required for files generated at runtime, such as reports generated by the Genero Report Engine (GRE).

Syntax

```
<DOCUMENT_ROOT> docroot; docrootuser </DOCUMENT_ROOT>
```

It allows for multiple document root paths to be specified, the separator used between resource paths is a semi-colon, '`;`'.

The `DOCUMENT_ROOT` element does not support any attributes or have any child elements.

Usage example

```
<DOCUMENT_ROOT>/usr/fgl2c/as/web</DOCUMENT_ROOT>
```

In this usage example, if you have the `demos.html` file in this directory and wish to access the file, use the URL: `http://<app_server>:<port>/demos.html` (where the file is on the host where the GAS resides) or `http://<web_server>/gas/demos.html` (where the file is on the Web server host), see File serving URIs on page 53.

Parent elements

This element is a child of one of the following elements: `INTERFACE_TO_CONNECTOR` on page 324

DVM_AVAILABLE

The `DVM_AVAILABLE` timeout specifies how long (in seconds) the Genero Application Server allows for the DVM to start. This parameter provides a delay for the DVM to start and be available.

The `DVM_AVAILABLE` timeout provides a mechanism for the Genero Application Server to handle the failure of the DVM to start. If the DVM has not started by the time the `DVM_AVAILABLE` timeout expires, the Genero Application Server sends an error message to the front end client and logs the message: "DVM_AVAILABLE timeout expired."

This element is optional. If the element is not specified, the default time out is 100 seconds.

Web applications

When used as a child of the `WEB_APPLICATION_TIMEOUT_COMPONENT` element, the `DVM_AVAILABLE` timeout is only applicable when you start an application or you launch sub process in interactive mode (IN LINE MODE). If you run the sub process in background (IN FORM MODE), the `DVM_AVAILABLE` timeout is not applicable.

Web services

For a Web service, there is no notion of a front-end client. An error message is therefore not sent. In addition, a Web service is not expected to perform RUNs commands with new DVMs connecting by themselves.

Usage example

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

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

Parent elements

This element is a child of one of the following elements: `WEB_APPLICATION_TIMEOUT_COMPONENT` on page 360, `SERVICE_APPLICATION_EXECUTION_COMPONENT` on page 344, `TIMEOUT (for a service)` on page 353, `TIMEOUT (for an application)` on page 352

DVM_PINGTIMEOUT

Time out (in seconds) before the Genero Application Server sends a PING (keep-alive) message to a running DVM.

The DVM Ping timeout specifies how long (in seconds) the GAS waits before send a PING (keep-alive) message to a running DVM.

A running DVM waits at a maximum `gui.protocol.pingTimeout` seconds for a front-end PING message when there is no user activity. After this timeout period elapses, the program stops with an error.

A correct configuration requires that the `DVM_PINGTIMEOUT` be set lower than the DVM `gui.protocol.pingTimeout` setting. By default, the DVM `gui.protocol.pingTimeout` is defined as 600 seconds and the GAS `DVM_PINGTIMEOUT` is defined as 300 seconds.

Usage example

```
<DVM_PINGTIMEOUT>300</DVM_PINGTIMEOUT>
```

In this example, the DVM PING timeout is set to 300 seconds.

Parent elements

This element is a child of one of the following elements: `WEB_APPLICATION_TIMEOUT_COMPONENT` on `page 360`, TIMEOUT (for an application) on page 352

DVM

The `DVM` element specifies the name of the Dynamic Virtual Machine you want to use to start and run the application. This value is typically **fglrun** for UNIX™ Systems (UNX) and **fglrun.exe** for Windows™ Systems (WNT).

Usage example

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

Parent elements

This element is a child of one of the following elements: `WEB_APPLICATION_EXECUTION_COMPONENT` `on page 356`, `SERVICE_APPLICATION_EXECUTION_COMPONENT` `on page 344`, `EXECUTION (for an application)` `on page 315`, `EXECUTION (for a service)` `on page 316`

ENVIRONMENT_VARIABLE

The `ENVIRONMENT_VARIABLE` element provides the value to be set for an environment variable. It takes an attribute **Id**, which specifies the name of the environment variable. Prior to starting the application, the environment variable is set using this information.

Usage example

```
<ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
```

In this example, the environment variable **FGLGUI** is set to **1**.

See also:

- Application environment on page 42

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_EXECUTION_COMPONENT on page 356, SERVICE_APPLICATION_EXECUTION_COMPONENT on page 344, EXECUTION (for an application) on page 315, EXECUTION (for a service) on page 316

ERROR_DOCUMENT

> **Important:** As of version 2.2x, this feature is no longer available. This should now be configured and handled by the web server.

The ERROR_DOCUMENT element contains the action to do in case of error. It takes an attribute **Code**, which specifies the HTTP status code. The element contains the action to perform in case of error. It can specify this action in one of three ways:

- Free text.
- An URL that is relative to DOCUMENT_ROOT and that begins with a backslash (/).
- An absolute URL that begins with "http".

Usage example

```
<ERROR_DOCUMENT Code="404">/demos.html</ERROR_DOCUMENT>
```

Parent elements

This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

EXECUTION (for an application)

The EXECUTION element sets the runtime environment for the application by specifying the parameters for executing a Web application. You can reference a predefined WEB_APPLICATION_EXECUTION_COMPONENT on page 356 to inherit the runtime environment settings of that component by including the **Using** attribute, specifying the unique identifier for that execution component, and/or you can set individual execution elements specific to the application.

The attribute AllowUrlParameters defines whether parameters provided on the command line should be ignored ("FALSE", default value) or provided to the DVM ("TRUE").

The attribute AllowUnsafeSession defines whether safe session management is active ("FALSE", default value) or deactivated ("TRUE"). Safe session management is a transparent session check based on session cookies, introduced to secure the application session tracking. You should deactivate (set to "TRUE") only if you encounter issues when migrating to version 2.50.20 or greater.

Settings defined locally within the EXECUTION element override settings defined in included execution components.

Child elements

The EXECUTION element may contain the following child elements:

> **Important:** Element order. If child elements are present, they must be set in the order listed below.

1. Zero to many ENVIRONMENT_VARIABLE elements.
2. Zero or one PATH element.
3. Zero or one DVM element.
4. Zero or one MODULE element.
5. Zero or one PARAMETERS element.
6. Zero or one ACCESS_CONTROL element.
7. Zero or one DELEGATE element.
8. Zero or one WEB_COMPONENT_DIRECTORY on page 361 element.

For more information on defining execution elements, see Web Application Execution Component and Service Application Execution Component.

Usage examples

```
<EXECUTION Using="cpn.wa.execution.local" />
```

```
<EXECUTION Using="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI">1</ENVIRONMENT_VARIABLE>
</EXECUTION>
```

```
<EXECUTION AllowUrlParameters="TRUE">
  <PATH>/home/examples/BuiltIn/Arguments</PATH>
</EXECUTION>
```

Parent elements

This element is a child of one of the following elements: APPLICATION (for an application) on page 301

EXECUTION (for a service)

The EXECUTION element sets the runtime environment for the application by specifying the parameters for executing a Web application. You can reference a predefined SERVICE_APPLICATION_EXECUTION_COMPONENT to inherit the runtime environment settings of that component by including the **Using** attribute, specifying the unique identifier for that execution component, and/or you can set individual execution elements specific to the application.

Settings defined locally within the EXECUTION element override settings defined in included execution components.

Child elements

Possible execution elements include:

1. Zero or more ENVIRONMENT_VARIABLE elements.
2. Zero or one PATH element.
3. Zero or one DVM element.
4. Zero or one MODULE element.
5. Zero or one PARAMETERS element.
6. Zero or one ACCESS_CONTROL element.
7. Zero or one DELEGATE element.
8. Zero or one WEB_COMPONENT_DIRECTORY on page 361 element.
9. Zero or one POOL on page 337/xref> element.

Usage examples

```
<EXECUTION Using="cpn.ws.execution.local" />
```

```
<EXECUTION Using="cpn.ws.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI>1</ENVIRONMENT_VARIABLE>
</EXECUTION>
```

Parent elements

This element is a child of one of the following elements: APPLICATION (for a service) on page 301

EXTERNAL_ID

The EXTERNAL_ID element is the public identifier of the DTD.

Usage example

```
<EXTERNAL_ID>-//W3C//DTD HTML 4.01//EN</EXTERNAL_ID>
```

Parent elements

This element is a child of one of the following elements: DOC_TYPE on page 312

FILE_TRANSFER

The FILE_TRANSFER element sets up parameters relevant for applications involved in transferring files between the front end and the application server host. The FILE_TRANSFER element specifies when the transferred files should be deleted after application end. The files are removed from the temporary directory at the dispatcher shut down and after application end, when the timeout specified between the TIMEOUT tags expires.

Syntax

```
<FILE_TRANSFER>
  <TIMEOUT>duration</TIMEOUT>
</FILE_TRANSFER>
```

Child elements

The FILE_TRANSFER element must contain the following child element:

1. One TIMEOUT element.

Example

```
<FILE_TRANSFER>
  <TIMEOUT>600</TIMEOUT>
</FILE_TRANSFER>
```

Parent elements

This element is a child of one of the following elements: APPLICATION_SERVER on page 303

FORMAT

The FORMAT element specifies the format of each log message.

The FORMAT element takes an optional attribute **Type** (whose value can only be TEXT) and a list of field identifiers.

The field identifiers indicate the line format. The fields are separated by spaces. The order of the fields determines the order of the log output.

Table 62: Log message field identifiers

Type	Description
category	Name of the category filter of the event.

Type	Description
`component`	Name of the component that generate the event.
`date`	Date of the event.
`time`	Time of the event.
`relative-time`	Time elapsed since the dispatcher has started.
`process-id`	Process identifier.
`thread-id`	Thread identifier.
`location`	The file name and line number where the event occurred. Replaces `line` and `file` Type attributes from earlier versions.
`contexts`	Internal data representing the successive context the process went through to reach the logged event. These data is formatted as type=value pairs separated by semicolons.
`event-type`	Event name.
`event-params`	Event details. XML fragment representing the structured information attached to the event. This XML fragment will be on a single line. If used this field should be the last one as it can contain data with spaces.

When parsing the content of log files:

- A log message ends with a LF/CR character.
- `event-type` values are enclosed in double quotes (").
- `event-params` values may contain any characters. This field should be put at the end of the log format string.
- All other field values do not contain spaces.
- A missing value is replaced by a hyphen (-).
- Within `event-params` values, well-known non-printable characters are replaced by standard C escape sequences; other non-printable characters use a hexadecimal encoding.
- Single event-params values, such as raw text, have the following structure: `Data Size=`*`size`* `Content=`*`value`*, where *size* is the size of the value.
- `event-params` contains the escaped log message.

Example

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

Parent elements
This element is a child of one of the following elements: `LOG on page 326`

GROUP (for an application)

The `GROUP` element allows you to specify a directory where external application configuration files are located. Once a `GROUP` has been declared, an administrator can add an external application configuration file into the specified directory, and the Genero Application Server will be able to locate and use that file without having to be restarted.

It takes an attribute **Id**, which specifies the unique identifier for this group. When calling an application defined by an external application configuration file, you must provide the group **Id** and the name of the

external application configuration file name (without the extension), which is typically the name of the application.

Syntax

```
<GROUP Id="groupName"> path </GROUP>
```

Note:

1. *groupName* is a string that uniquely identifies the group.
2. *path* is the directory path where the external application configuration files are to be placed.

Usage examples

```
<GROUP id="tut-demo">$(res.path.as.demo)/tut/app</GROUP>
<GROUP id="mygroup">/home/myuser/config</GROUP>
```

Parent elements

This element is a child of one of the following elements: APPLICATION_LIST on page 302

GROUP (for a service)

The GROUP element allows you to specify a directory where external Web service application configuration files are located. Once a GROUP has been declared, an administrator can add an external Web service application configuration file into the specified directory, and the Genero Application Server will be able to locate and use that file without having to be restarted.

It takes an attribute **Id**, which specifies the unique identifier for this group. When calling an application defined by an external application configuration file, you must provide the group **Id** and the name of the external Web service application configuration file name (without the extension), which is typically the name of the Web service application.

Syntax

```
<GROUP Id="groupName"> path </GROUP>
```

Syntax notes

1. *groupName* is a string that uniquely identifies the group.
2. *path* is the directory path where the external Web service application configuration files are to be placed.

Usage example

```
<GROUP id="mygroup">/home/myuser/config</GROUP>
```

Parent elements

This element is a child of one of the following elements: SERVICE_LIST on page 346

GWC-JS

The GWC-JS element references the GWC-JS bootstrap directory specified by a path in the GWC_JS_LOOKUP_PATH element.

The bootstrap template path refers to the directory that contains a collection of files known as the Bootstrap. Bootstrap is an HTML, CSS, and JavaScript framework for developing responsive, mobile-first projects on the web.

By default, a single bootstrap directory is provided at installation, and it is referenced in the `defaultwa` abstract application configuration in the Genero Application Server configuration file:

```
...
<RESOURCE Id="res.path.docroot" Source="INTERNAL">$(res.path.as)/web</
RESOURCE>
...
<APPLICATION Id="defaultwa" Abstract="TRUE">
<!-- This is the "default" application.
 It is not used directly: it is used for defining a "root" application. -->
  <EXECUTION Using="cpn.wa.execution.local"/>

  <UA_OUTPUT>
    <PROXY>$(res.uaproxy.cmd)</PROXY>
    <PUBLIC_IMAGEPATH>.</PUBLIC_IMAGEPATH>
    <TIMEOUT Using="cpn.wa.timeout"/>
    <GWC-JS>gwc-js</GWC-JS>
  </UA_OUTPUT>
...
```

Given these details, we can state that the bootstrap template for an application using the `UA_OUTPUT` element will, by default, use the files located in `$(res.path.docroot)/gwc-js`. When we replace the resource with its value, it becomes `$(res.path.as)/web/gwc-js`, where *$(res.path.as)* refers to the Genero Application Server installation directory. You can continue to replace each resource with the resource value provided by the configuration file, until we have the absolute path shown without any resources.

Usage example: Project Directory

```
<GWC-JS>gwc-js-custom</GWC-JS>
```

In this example, you create a new set of bootstrap files to customize the look and feel of a Web application in a separate directory to the default bootstrap, for example, *$FGLASDIR*`/web/gwc-js-custom`. The GAS searches for the GWC-JS directory at the path defined by the `GWC_JS_LOOKUP_PATH` on page 320.

Child elements

There are no child elements.

Parent elements

This element is a child of one of the following elements: UA_OUTPUT on page 354

GWC_JS_LOOKUP_PATH

The `GWC_JS_LOOKUP_PATH` element specifies paths to look for the GWC-JS directory.

By configuring the `GWC_JS_LOOKUP_PATH` element, you can specify the location of your custom Genero Web Client for JavaScript (GWC-JS) front end.

The GAS provides a dedicated URL as the document root where your customized GWC-JS directory is located (see `ua/w/` in Application URIs on page 49). The dispatcher searches the paths specified in `GWC_JS_LOOKUP_PATH` to locate the directory specified by the GWC-JS on page 319 element. When the directory is found, the requested file is sent if it exists. If the file does not exist, no additional look up is performed. This is to avoid the risk of mixing up files in other GWC-JS versions that may be located in various directories in these paths.

Syntax

```
<GWC_JS_LOOKUP_PATH>docroot;gwc_js_userdir</GWC_JS_LOOKUP_PATH>
```

It allows for multiple GWC-JS paths to be specified, the separator used between resource paths is a semi-colon, ';'.

Usage example

```
<GWC-JS-LOOKUP-PATH>$(res.path.docroot);$(res.path.gwcjs.user)</GWC-JS-
LOOKUP-PATH>
```

In this example, the GAS first searches for the GWC-JS directory at the default location defined by the resource *res.path.docroot* , which is *$FGLASDIR/*web. The second resource, *res.path.gwcjs.user*, which is not defined in the GAS configuration file by default, can be set as required at runtime.

Usage example setting GWC_JS_LOOKUP_PATH at runtime

```
httpdispatch -E res.path.gwcjs.user=<directory-of-your-choice>
```

In this example, you set the location of your customized GWC-JS files with the dispatcher switch (-E) by creating or overwriting the *res.path.gwcjs.user* resource with the path to your customized directory at runtime.

Child elements

There are no child elements.

Parent elements

This element is a child of the following element: INTERFACE_TO_CONNECTOR on page 324

HTTP_COOKIE

The HTTP_COOKIE element contains any HTTP cookie definitions for an application.

The main goal of cookies is to keep a state, through session variables, between two runs of an application by the same user. The number of cookies associated with an application should be constant.

Syntax

```
<HTTP_COOKIE Id="cid" [Expires="endTime" | Domain="mydomain" |
 Secure="TRUE|FALSE" | HttpOnly=""]>
  <VARIABLE Id="varId">val</VARIABLE> [...]
  <CONSTANT Id="cstId">cst</CONSTANT> [...]
<HTTP_COOKIE> [...]
```

Syntax notes

The HTTP_COOKIE element takes a mandatory Id attribute and four optional attributes: Expires, Domain, Secure and HttpOnly.

1. *cid* is the cookie name.
2. The Expires attribute specifies the cookie expiration date. *endTime* is cookie expiration in "Wdy, DD-Mon-YYYY HH:MM:SS GMT" format. You can set a relative date with "+X" or "X", where X represent a number of seconds. "X" will fix the cookie date only at the creation time and "+X" will regenerate a new date for the cookie on each HTTP request. The Expires attribute is optional.
3. The Domain attribute restricts the cookie to a specified domain. *mydomain* is the domain name the cookie is restricted to. The Domain attribute is optional.

4. When set to TRUE, the `Secure` attribute restricts the cookie to secured connections (HTTPS) only. Valid values are `TRUE` or `FALSE`. The `Secure` attribute is optional.
5. When set to TRUE, `HttpOnly` attribute disables the cookie access from client-side scripting languages, such as JavaScript™, running in a browser. Valid values are `TRUE` or `FALSE`. The `HttpOnly` attribute is optional.
6. *varId* is the variable name and *val* its value.
7. *cstId* is the constant name and *cst* its value.

Child elements

The `HTTP_COOKIE` element may contain the following child elements, defined by a mandatory identifier and an optional value.:

1. Zero to many `CONSTANT on page 310` elements
2. Zero to many `VARIABLE on page 355` elements

Example

```
<!-- secure persistent cookie with default variable value and constant value
 -->
<HTTP_COOKIES>
  <HTTP_COOKIE Id="cookie3" Expires="Wdy, DD-Mon-YYYY HH:MM:SS GMT"
   Domain="www.domain.com" Secure="TRUE" HttpOnly="TRUE">
    <VARIABLE Id="var7" />
    <VARIABLE Id="var8">Initial value</VARIABLE>
    <CONSTANT Id="constant1">A value</CONSTANT>
  </HTTP_COOKIE>
</HTTP_COOKIES>
```

For more information on HTTP cookies, refer to *Session Variables and Cookies*.

Parent elements

This element is a child of one of the following elements: `HTTP_COOKIES on page 322`, `WEB_APPLICATION_HTTP_COOKIES_COMPONENT on page 357`

HTTP_COOKIES

The `HTTP_COOKIES` element contains child elements that define persistent session variables or constants.

Child elements

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

1. Zero or more `HTTP_COOKIE on page 321` elements.

Syntax

```
<HTTP_COOKIES>
  <HTTP_COOKIE ...>...</HTTP_COOKIE>
  [...]
</HTTP_COOKIES>
```

Example

```
<!-- secure persistent cookie with default variable value and constant value
 -->
<HTTP_COOKIES>
  <HTTP_COOKIE Id="cookie3" Expires="Wdy, DD-Mon-YYYY HH:MM:SS GMT"
   Domain="www.domain.com" Secure="TRUE" HttpOnly="TRUE">
```

```
    <VARIABLE Id="var7" />
    <VARIABLE Id="var8">Initial value</VARIABLE>
    <CONSTANT Id="constant1">A value</CONSTANT>
  </HTTP_COOKIE>
</HTTP_COOKIES>
```

For more information on HTTP cookies, refer to *Session Variables and Cookies*.

Parent elements
This element is a child of one of the following elements: `OUTPUT (under APPLICATION)` on page 331

HTTP_HEADER

The `HTTP_HEADER` element specifies the request HTTP header for request response.

Usage example

```
<HTTP_HEADER Id="Cache-Control">no-cache</HTTP_HEADER>
```

Parent elements
This element is a child of one of the following elements: `OUTPUT (under APPLICATION)` on page 331

HTTP_REQUEST_ENCODING

- When "Source" attribute is "INLINE": use content of `HTTP_REQUEST_ENCODING` to determine encoding used in http request.
- When "Source" attribute is "REQUEST": use "_charset_" form field or Content-Type http request header sent by User-Agent to determine encoding used in http request. Use UTF-8 if "_charset_" form field or Content-Type http request are missing.

Usage example

```
<HTTP_REQUEST_ENCODING Source="INLINE">ISO-8859-1</HTTP_REQUEST_ENCODING>
```

or

```
<HTTP_REQUEST_ENCODING Source="REQUEST"/>
```

Parent elements
This element is a child of one of the following elements: `WEB_APPLICATION_RENDERING_COMPONENT` on page 358, `RENDERING` on page 339

HTTP_RESPONSE_ENCODING

- When "Source" attribute is "INLINE": use content of `HTTP_RESPONSE_ENCODING` and force HTTP response encoding to this value.
- When "Source" attribute is "REQUEST": use Accept-Charset HTTP request header sent by User-Agent to determine HTTP response encoding. Use UTF-8 if Accept-Charset is missing.

Usage example

```
<HTTP_RESPONSE_ENCODING Source="INLINE">ISO-8859-1</HTTP_RESPONSE_ENCODING>
```

or

```
<HTTP_RESPONSE_ENCODING Source="REQUEST"/>
```

Parent elements
This element is a child of one of the following elements: WEB_APPLICATION_RENDERING_COMPONENT on page 358, RENDERING on page 339

INTERFACE_TO_CONNECTOR
The INTERFACE_TO_CONNECTOR element specifies the interface to the Genero Application Server (GAS) Connector. It specifies the connection between the GAS and the GAS Connector (located with the Web Server).

The GAS Connector is either:

- The FastCGI extension
- The ISAPI extension
- The Java Servlet extension

The INTERFACE_TO_CONNECTOR element defines the port on which the Genero Application Server listens for incoming requests. This applies for the httpdispatcher, (the stand-alone GAS, see Dispatcher: httpdispatch on page 263) as well as the GAS Connectors.

Syntax

```
<INTERFACE_TO_CONNECTOR>
  <ROOT_URL_PREFIX>URL_Web_Server_behind_proxy</ROOT_URL_PREFIX>
  <TCP_BASE_PORT> base </TCP_BASE_PORT>
  <TCP_PORT_OFFSET> offset </TCP_PORT_OFFSET>
  <DOCUMENT_ROOT> root </DOCUMENT_ROOT>
  <GWC_JS_LOOKUP_PATH> path;path </GWC_JS_LOOKUP_PATH>
  <TEMPORARY_DIRECTORY> dir </TEMPORARY_DIRECTORY>
  <SESSION_DIRECTORY> dir </SESSION_DIRECTORY>
  <REPORT_VIEWER_DIRECTORY>dir</REPORT_VIEWER_DIRECTORY>
  <SOCKET_FAMILY> dir </SOCKET_FAMILY>
  <SOCKET_PATH> dir </SOCKET_PATH>
  [ <ERROR_DOCUMENT Code="code"> path </ERROR_DOCUMENT> ]
</INTERFACE_TO_CONNECTOR>
```

Child elements

The INTERFACE_TO_CONNECTOR element may contain the following child elements.

1. Zero or more ROOT_URL_PREFIX on page 344 element.
2. One TCP_BASE_PORT on page 350 element.
3. One TCP_PORT_OFFSET on page 350 element.
4. One DOCUMENT_ROOT on page 312 element.
5. One GWC_JS_LOOKUP_PATH on page 320 element.
6. One TEMPORARY_DIRECTORY on page 351.
7. One SESSION_DIRECTORY on page 346 element.
8. One REPORT_VIEWER_DIRECTORY on page 340 element.
9. One SOCKET_FAMILY on page 348 element.
10. One SOCKET_PATH on page 349 element.
11. Zero or more ERROR_DOCUMENT on page 315 element.

Example

```
<INTERFACE_TO_CONNECTOR>
  <ROOT_URL_PREFIX></ROOT_URL_PREFIX>
  <TCP_BASE_PORT>6300</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  <GWC_JS_LOOKUP_PATH>$(res.path.gwcjs.user);$(res.path.docroot)</
GWC_JS_LOOKUP_PATH>
  <TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>
  <SESSION_DIRECTORY>$(res.appdata.path)/session</SESSION_DIRECTORY>
  <REPORT_VIEWER_DIRECTORY>$(res.gredir)/viewer</REPORT_VIEWER_DIRECTORY>
  <SOCKET_FAMILY>$(res.dispatcher.socket.family)</SOCKET_FAMILY>
  <SOCKET_PATH>$(res.dispatcher.socket.path)</SOCKET_PATH>
  <SESSION_DIRECTORY>/var/tmp</SESSION_DIRECTORY>
</INTERFACE_TO_CONNECTOR>
```

In this example, the application server is listening on port 6394 (TCP_BASE_PORT + TCP_PORT_OFFSET), the application server web site root is specified as the resource $(res.path.docroot).

To have several instances of the Genero Application Server run concurrently on the same host, you create several application server configuration files with different offsets. Once started, each application server listens at the offset specified.

> **Important:** If you create multiple application configuration files (one for each instance of the application server), take care to ensure that the port values are unique for each application server started. If two application server configuration files both specify the same TCP_BASE_PORT and TCP_BASE_OFFSET, a port conflict exists. The second application server will not start, an error message displays (Application Server startup.....[fail]) and the message "Address already in use" is written to the log file.

Parent elements

This element is a child of one of the following elements: APPLICATION_SERVER on page 303

INTERFACE_TO_DVM

The INTERFACE_TO_DVM element in the Genero Application Server (GAS) configuration file specifies the connection between the GAS and the Dynamic Virtual Machine (DVM). This element specifies the address of the host where the GAS dispatcher runs.

There can only be one INTERFACE_TO_DVM element specified in a given Genero Application Server configuration file.

Syntax

```
<INTERFACE_TO_DVM>
  <ADDRESS> address </ADDRESS>
</INTERFACE_TO_DVM>
```

Child elements

The INTERFACE_TO_DVM element contains a child element:

1. One ADDRESS element.

Example

```
<INTERFACE_TO_DVM>
  <ADDRESS>app_server</ADDRESS>
</INTERFACE_TO_DVM>
```

Parent elements
This element is a child of one of the following elements: APPLICATION_SERVER on page 303

KEEP_ALIVE
Time out (in seconds) before shutting down a proxy that is no longer serving requests.

The proxy KEEP_ALIVE timeout is used to shut down the proxy that hasn't served any requests during the specified time. The KEEP_ALIVE element specifies how long (in seconds) the proxy waits with no request to process before determining whether to shut down.

If the KEEP_ALIVE entry is missing, the proxy will never shutdown. It also prevents the DVMs from running indefinitely in the unlikely event that the GAS dispatcher or the web server crashes.

Usage example

```
<KEEP_ALIVE>10</KEEP_ALIVE>
```

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

Parent elements
This element is a child of one of the following elements: SERVICE_APPLICATION_TIMEOUT_COMPONENT on page 345, TIMEOUT (for a service) on page 353

LONG

The LONG element contains the long description.

Usage example

```
<LONG>A long description</LONG>
```

Parent elements
This element is a child of one of the following elements: DESCRIPTION on page 311

LOG
The Genero Application Server creates log files for each of its dispatchers, proxies, and DVMs started. The LOG element and its child elements specify where the log files are created, the format of the log messages, the type of information logged, and the maximum size of a single log message.

It is possible to specify multiple LOG elements. For example, you may need to have separate log files capturing different categories of messages.

Syntax

```
<LOG>
  <OUTPUT Type="DAILYFILE|CONSOLE|CONSOLE,DAILYFILE">path/filename</OUTPUT>
  <FORMAT [ Type="TEXT" ]>fields-id</FORMAT>
  <CATEGORIES_FILTER>FILTERNAME [...]</CATEGORIES_FILTER>
  [ <RAW_DATA MaxLength="length" /> ]
</LOG>
```

Child elements

The LOG element may contain the following child elements:

1. One OUTPUT element.
2. One FORMAT element.
3. One CATEGORIES_FILTER element.

4. Zero or one RAW_DATA element.

Example

```
<LOG>
  <OUTPUT Type="DAILYFILE">$(res.log.output.path)</OUTPUT>
  <FORMAT Type="TEXT">time event-type event-params</FORMAT>
  <CATEGORIES_FILTER>GAS ACCESS PROCESS DEPRECATED ERROR WARNING</
CATEGORIES_FILTER>
  <RAW_DATA MaxLength="-1" />
</LOG>
```

To support more than one GAS process using the same log configuration, if a required log file cannot be opened, the GAS process ID will be added to the log file name.

Parent elements

This element is a child of one of the following elements: APPLICATION_SERVER on page 303

MAP

The MAP element is a combination of a rendering mechanism and a theme.

The MAP element takes a required attribute **Id**, which specifies the unique identifier for this component. The Id can be any value, but it is based on the result the user gets from adua.xrd. The default values include DUA_GDC, DUA_UA, (and DUA_HTML5, which is deprecated). The list can be extended by custom choices.

The MAP element may also specify optional attributes:

- **Allowed**, which specifies whether this map will be used in this context or not. Possible values are TRUE (allowed) or FALSE (not allowed). Default value is FALSE or the parent definition when inherited. For example, this can be used to forbid some applications to use PDA if they were not designed to.
- **ForwardDVMStderr**, which specifies whether the DVM standard error will be forwarded to the Genero Desktop Client. This attribute is ignored for GWC applications. Default value is FALSE or the parent definition when inherited.

Child elements

The MAP element may contain the following child elements:

1. Zero or one PROXY element.
2. Zero or one TIMEOUT element.
3. Zero or one PICTURE element.
4. Zero or one RENDERING element.
5. Zero or one THEME elements.

Parent elements

This element is a child of one of the following elements: OUTPUT (under APPLICATION) on page 331

MAX_AVAILABLE

The MAX_AVAILABLE element specifies the maximum number of available DVMs to be attached to a Web Service.

Constraints

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

To control the number of DVMs - and in effect the number of licenses - used by each application, you can split your licenses using MAX_AVAILABLE. By using a different fglprofile for each application, you can specify X licenses for application 1, Y licenses for application 2.

Parent elements
This element is a child of one of the following elements: POOL on page 337

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.

The value must be equal or greater than 1.

Parent elements
This element is a child of one of the following elements: POOL on page 337

MONITOR

The MONITOR element in the Genero Application Server configuration file specifies from which machines the monitor URL is accessible. By default, the monitor page is **not** accessible and needs to be configured.

> **Important:** Depending on the network configuration, the monitor is not always able to get the actual client IP address. If there is a proxy server between the client and the server, for example, the client IP address seen by the GAS may be the address from the proxy server.

Syntax

```
<MONITOR>
  <ALLOW_FROM>ip_address</ALLOW_FROM>[...]
</MONITOR>
```

where *ip_address* is a valid IPv4 or IPv6 address. For IPv4 it can be a complete IP address or a network address (ending with a dot)..

Child elements

- Zero or more ALLOW_FROM on page 300 elements.

Example configuring monitor access

In the default deployment monitoring is specified by the resource *res.access.control*, which is defined with the value *NOBODY* by default.

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

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

To allow monitoring from hosts, in the GAS configuration file (default *%FGLASDIR%*/etc/as.xcf) you need to change the application element for MONITOR from:

```
<MONITOR>
   <ALLOW_FROM>$(res.access.control)</ALLOW_FROM>
    <!--
        <ALLOW_FROM>192.168.</ALLOW_FROM>
        <ALLOW_FROM>10.</ALLOW_FROM>
        <ALLOW_FROM>193.111.222.123</ALLOW_FROM>
     -->
</MONITOR>
```

To (for example):

```
<MONITOR>
  <ALLOW_FROM>127.0.0.1</ALLOW_FROM>
  <ALLOW_FROM>192.168.</ALLOW_FROM>
  <ALLOW_FROM>10.</ALLOW_FROM>
  <ALLOW_FROM>193.111.222.123</ALLOW_FROM>
</MONITOR>
```

In this example, the GAS monitor is reachable from localhost (127.0.0.1), 193.111.222.123 and all IP that begin with "192.168." or "10.". For more details on the monitor usage, see Monitoring Genero Application Server

Parent elements
This element is a child of one of the following elements: APPLICATION_SERVER on page 303

MIME_TYPE

The MIME_TYPE element sets the MIME type in HTTP Content-Type response header. If not specified, the default MIME_TYPE of "text/html" is used by the output driver.

Usage example

```
<MIME_TYPE>text/html</MIME_TYPE>
<MIME_TYPE>text/xml</MIME_TYPE>
```

Parent elements
This element is a child of one of the following elements: WEB_APPLICATION_RENDERING_COMPONENT on page 358, RENDERING on page 339

MIN_AVAILABLE

The MIN_AVAILABLE element specifies the minimum number of available DVMs to be attached to a Web Service. It can be either less than or greater than the value specified by START. If START > MIN_AVAILABLE, the number of DVMs can decrease to reach MIN_AVAILABLE.

Constraint

0 <= MIN_AVAILABLE <= MAX_AVAILABLE

Parent elements
This element is a child of one of the following elements: POOL on page 337

MODULE

The MODULE element specifies the application module name (the name of the **.42r** module you want to run). If omitted, the GAS uses the name of the requested application.

While this element can be specified as part of an execution component, it is typically defined at the application level.

Usage example

```
<MODULE>Edit</MODULE>
```

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_EXECUTION_COMPONENT on page 356, SERVICE_APPLICATION_EXECUTION_COMPONENT on page 344, EXECUTION (for an application) on page 315, EXECUTION (for a service) on page 316

NAME

The NAME element is the name of the DTD.

Usage example

```
<NAME>HTML</NAME>
```

Parent elements

This element is a child of one of the following elements: DOC_TYPE on page 312

OUTPUT (under LOG)

The OUTPUT element specifies where the log messages are sent or written.

The OUTPUT element takes an attribute of **Type** and an optional path value. If the path value is not specified, the default of $FGLASDIR/log is used.

> **Note:** The default Genero Application Server configuration file provides a value for the OUTPUT element, specifying a directory other than $FGLASDIR/log. This directory can vary depending on the operating system. Check the configuration file to identify the directory specified for your installation.

Table 63: Valid values for the Type attribute of the OUTPUT element

Type	Description
DAILYFILE	Log files are written to disk. New log files are created each day. A directory is created each day with the naming convention naming *YYYYMMDD*. All log files created by the GAS during the day are stored in this directory. If a path is specified, the daily directory is created and stored under the log directory in the specified directory. If a path is not specified, the log files are created in the default log directory. DVM logs are redirected to files when DAILYFILE is set for the log output type.
CONSOLE	Log messages are sent to standard output.

Examples

```
<OUTPUT Type="DAILYFILE"/>
<OUTPUT Type="DAILYFILE">$(res.as.dir)/logdirs</OUTPUT>
<OUTPUT Type="CONSOLE"/>
<OUTPUT Type="CONSOLE,DAILYFILE"/>
```

In the first example, the daily log file is written to the default logging directory.

In the second example, the daily directory would be created in `$(res.as.dir)/logdirs/log`.

In the third example, log messages are sent to standard output.

In the fourth example, log messages are both written to a log file in the default logging directory and sent to standard output.

Parent elements
This element is a child of one of the following elements: `LOG on page 326`

OUTPUT (under APPLICATION)
Starting with Genero 3.00, the `OUTPUT` element is deprecated and used by GWC for HTML5 applications only.

> **Note:**
>
> As GWC for HTML5 is deprecated, the `OUTPUT` element is also deprecated. UI applications that are not using GWC for HTML5 use the UA_OUTPUT on page 354 element instead.

The `OUTPUT` element specifies the output parameters for a Web application, listing all maps required to make the defined Web application usable with different browsers/Front Ends.

It takes an optional attribute **Rule**, to assist with automatic discovery of the User Agent. For more information, see Automatic Discovery of User Agent.

Child elements

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

1. Zero or more HTTP_HEADER elements.
2. Zero or one HTTP_COOKIES elements.
3. Zero or more MAP elements.

Parent elements
This element is a child of one of the following elements: `APPLICATION (for an application) on page 301`

OUTPUT_DRIVER

The `OUTPUT_DRIVER` element specifies the output driver to be used. Valid values include:

- `GWC2` - The GWC2 output driver specifies that the snippet-based rendering engine be used to render the application for GWC front-end clients.
- `JFE36` - This value is for use with GDC Front End clients.

Usage example

```
<OUTPUT_DRIVER>GWC2</OUTPUT_DRIVER>
<OUTPUT_DRIVER>JFE36</OUTPUT_DRIVER>
```

Parent elements
This element is a child of one of the following elements: `WEB_APPLICATION_RENDERING_COMPONENT on page 358`, `RENDERING on page 339`

PARAMETERS

The `PARAMETERS` element specifies the parameters to provide on the DVM command line. Each parameter is contained in its own `PARAMETER` element.

If allowed, parameters can also be set in the application URL. For example, `/ua/r/myapp?Arg=val1&Arg=Val2` provides two parameters. To enable URL parameters for Web applications, set the

`AllowUrlParameters` attribute in the EXECUTION (for an application) on page 315 tag to `TRUE`. The default is `FALSE`. If the DVM already has parameters set by the command line, the parameters in the URL are added to the end of the command line.

Important: This attribute is not supported and should be removed for Web services applications.

Syntax

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

Child elements

- Zero or more `PARAMETER` elements.

Usage examples

The following example provides two parameters:

- Hello world!
- Again

```
<PARAMETERS>
  <PARAMETER>Hello world!</PARAMETER>
  <PARAMETER>Again</PARAMETER>
</PARAMETERS>
```

If URL parameters are allowed, these parameters are listed after the ones defined in the `PARAMETERS` element of the configuration file.

Parent elements

This element is a child of one of the following elements: `WEB_APPLICATION_EXECUTION_COMPONENT` on page 356, `SERVICE_APPLICATION_EXECUTION_COMPONENT` on page 344, `EXECUTION (for an application)` on page 315, `EXECUTION (for a service)` on page 316

PATH (under EXECUTION)

The `PATH` element specifies the current working directory for the application module.

Usage example

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

Parent elements

This element is a child of one of the following elements: `WEB_APPLICATION_EXECUTION_COMPONENT` on page 356, `SERVICE_APPLICATION_EXECUTION_COMPONENT` on page 344, `EXECUTION (for an application)` on page 315, `EXECUTION (for a service)` on page 316

PATH (under PICTURE)

Starting with Genero 3.00, the `PATH (under PICTURE)` element is deprecated and used by GWC for HTML5 applications only.

Note:

UI applications that are using GWC-JS, see the PUBLIC_IMAGEPATH on page 338 under the UA_OUTPUT on page 354 element instead.

The PATH element specifies where to look for an image or resource. The exact contents of the PATH element depends on whether the Type attribute is set to WEBSERVER or APPSERVER.

Parent elements
This element is a child of one of the following elements: WEB_APPLICATION_PICTURE_COMPONENT on page 357, PICTURE on page 335
PATH with Type WEBSERVER
Starting with Genero 3.00, the PATH with Type WEBSERVER attribute is deprecated and used by GWC for HTML5 applications only.

> **Note:**
>
> UI applications that are using GWC-JS, see the PUBLIC_IMAGEPATH on page 338 under the UA_OUTPUT on page 354 element instead.

When you specify the Type as WEBSERVER, the PATH value provides the URL for the directory where the resources reside on the Web server side. This URL typically consists of the Web server directory resource combined with a directory alias.

Syntax

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
  <PATH Id="pathID" Type="WEBSERVER">$(connector.uri)/path</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

1. *resID* is the unique identifier for this picture component.
2. *pathID* defines what is located in the specified path, such as "Image" or "Resource". The Id attribute for the PICTURE element is optional; if not specified, it defaults to "Image".
3. The TYPE attribute is optional; if not specified, it defaults to WEBSERVER.
4. *$(connector.uri)* is the resource for the Web server directory. If you are using a direct connection to the GAS, the resource *$(connector.uri)* is empty. If you connect through an Apache web server, *$(connector.uri)* is replaced by **/gas/**, assuming your URL is

   ```
   http://WebServer/gas/ua/r/AppID
   ```

   If *$(connector.uri)* is not specified in the picture path, the web server is searched for the images.
5. *path* is relative to the DOCUMENT_ROOT on page 312 path set in the configuration to map to the physical directory that stores the image files. You need to specify a valid path according to the DOCUMENT_ROOT path in your configuration.

Example

```
<!-- A Legacy Picture Component -->
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.picture.appserver">
<PATH Id="Image" Type="WEBSERVER" >$(connector.uri)/pic</PATH>
<PATH Id="Resource" Type="WEBSERVER" >$(connector.uri)/fjs</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

> **Note:** If DOCUMENT_ROOT is set to **/usr/local/genero/web**, our example maps the Image path to **/usr/local/genero/web/pic** and the Resource path to **/usr/local/genero/web/fjs**.
>
> **Note:** The front-end clients use *$(pictures.uri)* in their templates to access the pictures. This corresponds to the picture component path: *$(connector.uri)*/pic.
>
> **Tip:** When creating your HTML pages, use the absolute path to HTML objects like images or JavaScript files; for example, use /fjs/pic/accept.png rather than ../pic/accept.png, because URLs containing **..** will be rejected by the GAS for security reasons.

PATH with Type APPSERVER

Starting with Genero 3.00, the `PATH with Type APPSERVER` attribute is deprecated and used by GWC for HTML5 applications only.

> **Note:**
>
> UI applications that are using GWC-JS, see the PUBLIC_IMAGEPATH on page 338 under the UA_OUTPUT on page 354 element instead.

When you specify the `Type` of APPSERVER, the `PATH` value is then an enumeration of disk locations where the resource is potentially located. The server will then build specific URL for each resources and then catch these URL to search through the given paths for the resource on disk and deliver it.

Syntax

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
  <PATH Id="pathID" Type="APPSERVER" [ ExtensionFilter="ext" ]
   [ DVMFallbackAllowed="TRUE|FALSE" ]>pathlist</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Syntax notes

1. *resID* is the unique identifier for this picture component definition.
2. *pathID* defines what is located in the specified path, such as "Image" or "Resource". The `Id` attribute for the PICTURE element is optional; if not specified, it defaults to "Image".
3. **ExtensionFilter** is an optional attribute that filters access to application resources like images. Its value is a list of extensions, separated by semi-colons. For example: .png;.gif;.jpeg;.jpg;.bmp;.ico;.js;.css. The extensions are **case sensitive**. You might want to allow .png and not .PNG. If ExtensionFilter is not defined, access to any resources of the application is granted. ExtensionFilter is also used to filter image extensions that are asked to the DVM.By default, any applications which inherits **defaultgwc** configuration have restricted access to resources (see defaultgwc image component references in as.xcf)
4. **DVMFallbackAllowed** is an optional attribute that allows images and resources to be asked to the DVM if they are not found at the locations pointed by *pathlist*. By default, asking images to the DVM is allowed.
5. *pathlist* is a list of directory paths separated by semi-colons.

Example

```
<!-- A GAS multi-location Picture Component -->
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.gwc.html5.picture">
  <PATH Id="Resource" Type="WEBSERVER">$(connector.uri)/fjs</PATH>
  <PATH Id="Image" Type="APPSERVER"
 ExtensionFilter="$(res.image.extensions)">
   $(res.path.tpl.html5)/img;$(res.path.pic);$(application.path)</PATH>
  <PATH Id="SetHtml5" Type="APPSERVER"
   ExtensionFilter="$(res.web.extensions);.less;.svg"
 DVMFallbackAllowed="FALSE">
   $(res.path.tpl.html5);$(res.path.tpl.common)</PATH>
  <PATH Id="WebComponents" Type="APPSERVER"
   ExtensionFilter="$(res.web.components.extensions)"
 DVMFallbackAllowed="FALSE">
   $(res.path.docroot)</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

PICTURE

Starting with Genero 3.00, the PICTURE element is deprecated and used by GWC for HTML5 applications only.

> **Note:**
>
> UI applications that are using GWC-JS, see the PUBLIC_IMAGEPATH on page 338 under the UA_OUTPUT on page 354 element instead.

The PICTURE element specifies the picture parameters required by this Web application. It takes an attribute **Using**, where you can reference a predefined WEB_APPLICATION_PICTURE_COMPONENT (to inherit the picture parameters of that component), or you can specify the path to the image directory by including a PATH element.

Usage example

```
<PICTURE Using="cpn.gwc.html5.picture" />
```

```
<PICTURE>
  <PATH>$(connector.uri)/iiug/images</PATH>
</PICTURE>
```

Each application output map can define a PICTURE element. This element will be used to look for images and resources handled by URLs generated by the resourceURI() SBRE function or the legacy imageURI() SBRE function.

There are several ways to define a PICTURE element:

1. The PICTURE element can simply refer to a previously defined *picture component*:

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
    <PICTURE Using="cpn.gwc.picture.appserver"/>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
    <PICTURE Using="cpn.gwc.picture.webserver"/>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

2. The PICTURE element can define its own Picture component based on the Web Server behavior:

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HMTL5">
      <PICTURE>
        <PATH Id="Image" Type="WEBSERVER">$(connector.uri)/card/img</PATH>
      </PICTURE>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

3. The PICTURE element can define its own Picture component based on the AppServer behavior:

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
      <PICTURE>
        <PATH Id="Image" Type="APPSERVER">
         $(res.path.demo.app)/card/img;$(defaultPicturePath)
        </PATH>
      </PICTURE>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

4. For each application output map, you can define several PATH element using different IDs inside a PICTURE element. This allows the use of a mixed mechanism of resources referencing:

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.21/
cfextwa.xsd">
  <EXECUTION>
    <PATH>$(res.path.demo.app)/card/src</PATH>
    <MODULE>card.42r</MODULE>
  </EXECUTION>
  <OUTPUT>
    <MAP Id="DUA_HTML5">
     <PICTURE Using="cpn.gwc.picture.webserver">
        <PATH Id="AppServerImage" Type="APPSERVER">
         $(res.path.demo.app)/card/img;$(defaultPicturePath)</PATH>
        <PATH Id="WebServerImage" Type="WEBSERVER">
         $(connector.uri)/mypics</PATH>
      </PICTURE>
    </MAP>
```

```
  </OUTPUT>
</APPLICATION>
```

Parent elements

This element is a child of one of the following elements: MAP on page 327

PLATFORM_INDEPENDENT

This element contains a collection of platform-independent RESOURCE elements.

The PLATFORM_INDEPENDENT element contains a list of platform-independent resources, available for both UNIX™ and Windows™ platforms.

Child elements

A PLATFORM_INDEPENDENT element main contain the following child elements:

- Zero to many RESOURCE elements.

Parent elements

This element is a child of one of the following elements: RESOURCE_LIST on page 343

POOL

The POOL element sets the limitations regarding the number of Virtual Machines (DVMs) that are attached to a Web Service. You specify four values within a POOL element:

- The number of DVMs to start when the GAS starts
- The minimum number of DVMs to have alive while the GAS is running
- The maximum number of DVMs to have alive while the GAS is running.
- The maximum number of requests a DVM can handle before being stopped by the pool.

Note: The POOL element is only available for Web Services.

Syntax

```
<POOL>
 <START> startValue </START>
 <MIN_AVAILABLE> minValue </MIN_AVAILABLE>
 <MAX_AVAILABLE> maxValue </MAX_AVAILABLE>
 <MAX_REQUESTS_PER_DVM> maxRequests </MAX_REQUESTS_PER_DVM>
</POOL>
```

Child elements

The POOL element may contain the following child elements:

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

Example

```
<POOL>
  <START>5</START>
  <MIN_AVAILABLE>3</MIN_AVAILABLE>
  <MAX_AVAILABLE>10</MAX_AVAILABLE>
  <MAX_REQUESTS_PER_DVM>1</MAX_REQUESTS_PER_DVM>
```

```
</POOL>
```

In this example, 5 DVMs are started to service the Web service when the GAS starts; the number can fall as low as 3 DVMs or rise as high as 10 DVMs. For more information on setting service pool elements, see the Service Pool section of the GAS Architecture topic.

Parent elements
This element is a child of one of the following elements:
SERVICE_APPLICATION_EXECUTION_COMPONENT on page 344, EXECUTION (for a service) on page 316,

PROXY (for an application)

Specifies the executable you want to use for this (set of) application(s). Usually, the value is `fglrun` for UNIX™ Systems (UNX) and `fglrun.exe` for Windows NT™ / 2000 / XP (WNT).

Usage examples

```
<PROXY>fglrun</PROXY>
<PROXY>fglrun.exe</PROXY>
```

Parent elements
This element is a child of one of the following elements: UA_OUTPUT on page 354, MAP on page 327

PROXY (for a service)

Specifies the executable you want to use for this (set of) application(s). Usually, the value is `fglrun` for UNIX™ Systems (UNX) and `fglrun.exe` for Windows NT™ / 2000 / XP (WNT).

Usage examples

```
<PROXY>fglrun</PROXY>
<PROXY>fglrun.exe</PROXY>
```

Parent elements
This element is a child of one of the following elements: APPLICATION (for a service) on page 301

PUBLIC_IMAGEPATH

The `PUBLIC_IMAGEPATH` element defines a path relative to the root path *`appdata/`*`public`, where `fglrun` looks for resources for common images used by applications. The *"appdata"* resource, as it is commonly known, is set by the resource *$(res.appdata.path)* in the GAS configuration file, see GAS directories on page 38.

> **Note:** UI applications that are using GWC for HTML5, see the PATH (under PICTURE) on page 332 element instead.

Syntax

```
<PUBLIC_IMAGEPATH>path</PUBLIC_IMAGEPATH>
```

Where *path* is a path relative to the root path *`appdata/`*`public`.

Usage example 1

```
    ...
```

```
      <UA_OUTPUT>
         <PROXY></PROXY>
         <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
         <GWC-JS>gwc-js</GWC-JS>
         <TIMEOUT Using="cpn.wa.timeout"/>
      </UA_OUTPUT>
      ...
```

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

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

Usage example 2

```
      ...
      <UA_OUTPUT>
         <PROXY></PROXY>
         <PUBLIC_IMAGEPATH>myapp/newpictures</PUBLIC_IMAGEPATH>
         <GWC-JS>gwc-js</GWC-JS>
         <TIMEOUT Using="cpn.wa.timeout"/>
      </UA_OUTPUT>
      ...
```

In this example, the value of the `PUBLIC_IMAGEPATH` is set to "myapp/newpictures", so images are therefore sought in the *appdata*`/public/myapp/newpictures` directory.

Child elements

There are no child elements.

Parent elements

This element is a child of one of the following elements: UA_OUTPUT on page 354

RAW_DATA

The `RAW_DATA` element limits the size of a single log message.

Log messages can include the complete html response, and can therefore be very large (an entire html page). The optional `RAW_DATA` element specifies the maximum number of characters in any single log message. It takes an attribute **MaxLength**, the number of characters after which the log message is truncated. The value provided for **MaxLength** must be a non-negative integer.

If the `RAW_DATA` element is omitted, data is logged in its entirety.

Example

```
<RAW_DATA MaxLength="100" />
```

Parent elements

This element is a child of one of the following elements: `LOG` on page 326

RENDERING

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

The RENDERING element defines the rendering to be applied to this Web application.
It takes an optional attribute **Using**, in which the unique identifier of a predefined WEB_APPLICATION_RENDERING_COMPONENT element can be specified.

The RENDERING element may contain an OUTPUT_DRIVER on page 331 child element, specifying the output driver to be used. If an output driver is defined here, it overrides any output driver settings inherited via the Usage attribute and its specified rendering component.

Child elements

The RENDERING element may contain the following child elements:

1. Zero or one OUTPUT_DRIVER element.
2. Zero or one XML_DECLARATION element.
3. Zero or one HTTP_RESPONSE_ENCODING element.
4. Zero or one HTTP_REQUEST_ENCODING element.
5. Zero or one MIME_TYPE element.
6. Zero or one DOC_TYPE element.

Usage examples

```
<RENDERING Using="cpn.rendering.wa" />
```

```
<RENDERING>
  <OUTPUT_DRIVER>JFE36</OUTPUT_DRIVER>
</RENDERING>
```

Parent elements

This element is a child of one of the following elements: MAP on page 327

REPORT_VIEWER_DIRECTORY

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

Starting with Genero 3.00, the REPORT_VIEWER_DIRECTORY allows you to configure the location of the Genero Web Report Viewer, where report viewer files may be accessed. A corresponding report viewer URL prefix, /ua/report/viewer , is provided to the GRE to load the report viewer implementation, see Application URIs on page 49.

Syntax

```
<REPORT_VIEWER_DIRECTORY>path</REPORT_VIEWER_DIRECTORY>
```

The REPORT_VIEWER_DIRECTORY element does not support any attributes or have any child elements.

Usage example

```
    <REPORT_VIEWER_DIRECTORY>$(res.gredir)/viewer</REPORT_VIEWER_DIRECTORY>
   ...
```

Parent elements
This element is a child of the following element: INTERFACE_TO_CONNECTOR on page 324

REQUEST_RESULT (for an application)
Time out (in seconds) for pending transactions.

The Request Result timeout is used to inform the user when a transaction is taking longer than expected. The REQUEST_RESULT element specifies the number of seconds to wait for the DVM to give an answer to the Application Server, after which the Application Server sends a "transaction pending" page to the Front End client to inform the user that this transaction is taking longer than expected. This is also known as sending a keepalive response. The default transaction pending page automatically submits a new request to wait for the DVM to complete its processing.

Under normal operations, the Front End client sends a GET request to the Genero Application Server immediately after receiving a response. Meanwhile, the Genero Application Server stores data sent by the DVM for the application in its buffer, waiting for a GET request from the client. When the GET request is received by the Genero Application Server, if the server has data sent by the DVM in its buffer, the stored data is sent back to the Front End client. If the DVM does not have data to send, the Genero Application Server waits and, if the DVM is still processing the request after the specified REQUEST_RESULT timeout expires, it sends the keepalive response to the Front End client and resets the REQUEST_RESULT timer.

> **Tip:** The number of seconds specified for the REQUEST_RESULT timeout should be less that the cgi timeout. By default, the Apache Web server has the cgi timeout default to 300 seconds. Therefore, the REQUEST_RESULT timeout has an initial default setting of 60 seconds.

Usage example

```
<REQUEST_RESULT>60</REQUEST_RESULT>
```

In this usage example, the Request Result timeout is set to 60 seconds.

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

- WEB_APPLICATION_TIMEOUT_COMPONENT on page 360
- TIMEOUT

REQUEST_RESULT (for a service)
Time out (in seconds) for pending transactions.

The Request Result timeout is used to inform the user when a Web service transaction is taking longer than expected. The REQUEST_RESULT element specifies the number of seconds to wait for the DVM to give an answer to the Application Server, after which the Application Server sends a HTTP 400 error page to the Front End client to inform the user that the request has taken too long to fulfill.

The Front End client cannot recover from a HTTP 400 error page, and any Web service client application must send a new request.

> **Tip:** The number of seconds specified for the REQUEST_RESULT timeout should be less that the cgi timeout. By default, the Apache Web server has the cgi timeout default of 300 seconds. If setting the REQUEST_RESULT timeout, it should be set to less than that, for example, at 60 seconds.

Usage example

```
<REQUEST_RESULT>60</REQUEST_RESULT>
```

In this usage example, the Request Result timeout is set to 60 seconds. It tells the GWS proxy to release DVM in charge of a service that has not responded in the given time frame.

Note: If Request Result timeout is not set (the default), the GWS proxy never releases the DVM and will wait until DVM responds to the request.

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

- SERVICE_APPLICATION_TIMEOUT_COMPONENT on page 345
- TIMEOUT

RESOURCE
This element defines a variable you can define once and use elsewhere in your Genero Application Server and external application configuration files.

Resources allow you to create resources, or variables, for use within the configuration files and templates. Resources can be defined in the general RESOURCE_LIST element or within individual application or Web service configurations.

A RESOURCE element defines a resource, or variable, that can be used in configuration files and template files. It takes two attributes, an **Id** attribute and a **Source** attribute. The Id attribute is the identifier of the resource itself, while the Source attribute tells the Application Server where to find the value of the resource.

Syntax

```
<RESOURCE Id="resId" Source="{ INTERNAL | ENVIRON } " > resData </RESOURCE>
```

Syntax notes

1. *resId* is the resource identifier
2. *resData* is the resource data. Its use depends on the value of the Source attribute.
   - If Source is INTERNAL, *resData* is the value of the resource.
   - If Source is ENVIRON, *resData* is the name of an environment variable.
3. Resources are used in the configuration files or in the template files using the syntax:

```
$(resId)
```

Usage examples

A resource defined inline.

```
<RESOURCE Id="res.dvm.wa" Source="INTERNAL">$(res.fgldir)/bin/fglrun.exe</
RESOURCE>
```

A resource defined as the value of an environment variable. In this example, the resource res.os contains the value of the environment variable OS. For example, on a Windows™ system, the environment variable OS could have the value Windows_NT.

```
<RESOURCE Id="res.os" Source="ENVIRON">OS</RESOURCE>
```

Parent elements
This element is a child of one of the following elements: PLATFORM_INDEPENDENT on page 337, UNX on page 355, WNT on page 361

RESOURCE (for a service)

The RESOURCE element defines a resource available for this application. For more information, see *Resources*.

Parent elements

This element is a child of one of the following elements: APPLICATION (for a service) on page 301

RESOURCE (for an application)

The RESOURCE element defines a resource available for this application. For more information on defining resources, see RESOURCE on page 342.

Parent elements

This element is a child of one of the following elements: APPLICATION (for an application) on page 301

RESOURCE_LIST

This element contains all RESOURCE elements, organized by operating system.

The RESOURCE_LIST element of the Genero Application Server configuration file allows you to define RESOURCE elements, which can then be referenced in your configuration files and template files. A resource is a type of variable. By defining and using resources, when the value of the resource needs updating, it becomes possible to modify the resource in one location, and the new value is carried through the various configuration and template files that reference the resource.

A resource is defined as platform-independent or platform-dependent, based on the section (parent element) in which the resource is defined.

Syntax

```
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT> [ resource ] [...] </PLATFORM_INDEPENDENT>
  <WNT> resource [...] </WNT>
  <UNX> resource [...] </UNX>
</RESOURCE_LIST>
```

Child elements

A RESOURCE_LIST element contains the following child elements:

- One PLATFORM_INDEPENDENT element, containing a list of platform-independent resources.
- One WNT element, containing a list of WNT-specific resources.
- One UNX element, containing a list of UNIX-specific resources.

Example

```
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT>
    <RESOURCE Id="res.fglgui" Source="INTERNAL">1</RESOURCE>
    ...
  </PLATFORM_INDEPENDENT>
  <WNT>
    <RESOURCE Id="res.dvm.wa" Source="INTERNAL">
     $(res.fgldir)\bin\fglrun.exe</RESOURCE>
    ...
  </WNT>
  <UNX>
```

```
    <RESOURCE Id="res.dvm.wa" Source="INTERNAL">
     $(res.fgldir)/bin/fglrun.exe</RESOURCE>
    ...
  </UNX>
</RESOURCE_LIST>
```

For more information on defining a resource, see RESOURCE on page 342.

Parent elements
This element is a child of one of the following elements: APPLICATION_SERVER on page 303

ROOT_URL_PREFIX

The ROOT_URL_PREFIX element specifies the URL to access the Web server when a reverse proxy server is used between the client and the GAS. The ROOT_URL_PREFIX will override the URLs generated by the Web server and will construct them using this prefix value instead.

> **Note:** The reverse proxy server works on behalf of the Application server. The Web client is not aware of the proxy and does not know or see what server it is being forwarded to behind the proxy. In this case, the Web server uses the ROOT_URL_PREFIX, which provides the correct interface to the client.

Usage example

```
<ROOT_URL_PREFIX>http://serverB:8080/java-j2eedispatch</ROOT_URL_PREFIX>
```

- Where a reverse proxy server (e.g. server A) is forwarding requests to the GAS on *serverB*.
- Where "java-j2eedispatch" specifies the *connector.uri* part of the URI; typically this is the same as *connector.uri* acknowledged by the GAS for the dispatcher specific to that Web server.

> **Note:** If ROOT_URL_PREFIX is defined and is empty, it behaves as if not defined.

Parent elements
This element is a child of the following element: INTERFACE_TO_CONNECTOR on page 324

SERVICE_APPLICATION_EXECUTION_COMPONENT

The SERVICE_APPLICATION_EXECUTION_COMPONENT creates a Web service execution component, which defines a set of execution parameters that are used when starting the Web service. It takes an attribute **Id**, which specifies the unique identifier for this set of execution definitions. It is this unique identifier that is referenced by a Web service, providing that Web service with a base set of execution parameters.

Syntax

```
<SERVICE_APPLICATION_EXECUTION_COMPONENT Id="compId">
 <ENVIRONMENT_VARIABLE Id="envId" > env </ENVIRONMENT_VARIABLE> [...]
 [ <PATH> path </PATH> ]
 [ <DVM> dvm </DVM> ]
 [ <MODULE> module </MODULE> ]
 [ <PARAMETERS> parameterSettings </PARAMETERS> ]
 [ <ACCESS_CONTROL> accessSettings </ACCESS_CONTROL> ]
 [ <DELEGATE> delegateSettings </DELEGATE> ]
 [ <WEB_COMPONENT_DIRECTORY>webcomponentsSettings</WEB_COMPONENT_DIRECTORY>
 [ <POOL> poolSettings </POOL> ]
</SERVICE_APPLICATION_EXECUTION_COMPONENT>
```

Child elements

The SERVICE_APPLICATION_EXECUTION_COMPONENT element may contain the following child elements:

1. Zero or more ENVIRONMENT_VARIABLE elements.
2. Zero or one PATH element.
3. Zero or one DVM element.
4. Zero or one MODULE element.
5. Zero or one PARAMETERS element.
6. Zero or one ACCESS_CONTROL element.
7. Zero or one DELEGATE element.
8. Zero or one WEB_COMPONENT_DIRECTORY on page 361 element.
9. Zero or one POOL element.

Example

```
<SERVICE_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="INFORMIXDIR">$(res.informixdir)</
ENVIRONMENT_VARIABLE>
  <ENVIRONMENT_VARIABLE Id="INFORMIXSERVER">$(res.informixserver)
   </ENVIRONMENT_VARIABLE>
  ...
  <DVM>$(res.dvm.wa)</DVM>
</SERVICE_APPLICATION_EXECUTION_COMPONENT>
```

Parent elements

This element is a child of one of the following elements: COMPONENT_LIST on page 308

SERVICE_APPLICATION_TIMEOUT_COMPONENT

The SERVICE_APPLICATION_TIMEOUT_COMPONENT element creates a Web service application timeout component, which define a set of timeout values to be used when configuring a Web service. It takes an attribute **Id**, which specifies the unique identifier for this set of timeout definitions. It is this unique identifier that is referenced by a Web service, providing that Web service with a base set of timeout values.

The GAS handles the Web Services Server side. It takes care of the DVM requested by a Web Services client.

Syntax

```
<SERVICE_APPLICATION_TIMEOUT_COMPONENT Id="sTimeOutID">
  <DVM_AVAILABLE>dvmTimeOut</DVM_AVAILABLE>
  <KEEP_ALIVE>dvmKeepAliveTimeOut</KEEP_ALIVE>
  <REQUEST_RESULT>requestTimeOut</REQUEST_RESULT>
</SERVICE_APPLICATION_TIMEOUT_COMPONENT>
```

Child elements

The SERVICE_APPLICATION_TIMEOUT_COMPONENT element may contain the following child elements.

1. Zero or one DVM_AVAILABLE element.
2. Zero or one KEEP_ALIVE element.
3. Zero or one REQUEST_RESULT (for a service) on page 341 element.

Usage example

```
<SERVICE_APPLICATION_TIMEOUT_COMPONENT Id="cpn.ws.timeout.set1">
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
  <KEEP_ALIVE>360</KEEP_ALIVE>
  <REQUEST_RESULT>60</REQUEST_RESULT>
</SERVICE_TIMEOUT_COMPONENT>
```

In this example, the **Id** value - **cpn.ws.timeout.set1** - can be referenced when defining a Web service. When a Web service references a component by its **Id** value, it inherits the settings defined by that component.

Parent elements

This element is a child of one of the following elements: COMPONENT_LIST on page 308

SERVICE_LIST

The SERVICE_LIST element provides a list of groups and Web services applications (for those Web services applications defined within the Genero Application Server configuration file).

Syntax

```
<SERVICE_LIST>
  [ <GROUP ...> [...] ]
  [ <APPLICATION ...> [...] ]
</SERVICE_LIST>
```

Child elements

The SERVICE_LIST element may contain the following child elements:

1. Zero or more GROUP elements.
2. Zero or more APPLICATION elements.

Example

```
<CONFIGURATION>
  <APPLICATION SERVER>
    ...
    <SERVICE_LIST>
      ...
    </SERVICE_LIST>
  </APPLICATION_SERVER>
</CONFIGURATION>
```

> **Important:** You must include the SERVICE_LIST element, even if the Genero Application Server does not have any Web Services to define. In this situation, you simply specify an empty SERVICE_LIST element.

Parent elements

This element is a child of one of the following elements: APPLICATION_SERVER on page 303

SESSION_DIRECTORY

The SESSION_DIRECTORY element specifies where to store the session files of all applications and services started by the GAS. This element is optional.

Usage example

```
<SESSION_DIRECTORY>/var/tmp</SESSION_DIRECTORY>
```

In this example, the session files are stored in sub directories of /var/tmp/session.

By default, the SESSION_DIRECTORY entry is not set and the session files are stored in $FGLASDIR.

Parent elements

This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

SHORT

The SHORT element contains the short description.

Usage example

```
<SHORT>A short description</SHORT>
```

Parent elements

This element is a child of one of the following elements: DESCRIPTION on page 311.

SHORTCUT

Selects the template to be used to generate the GDC shortcut for this application.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Syntax

```
<SHORTCUT Id="shortcutId"> shortcutPath </SHORTCUT>
```

Syntax notes

1. *shortcutId* is the unique identifier for this element.
2. *shortcutPath* is the path to the shortcut template to be used in this theme.

Usage example

```
<WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.default.gdc">
  <TEMPLATE Id="_default">$(res.theme.default.gdc.template)</TEMPLATE>
  <SHORTCUT Id="_default">$(res.theme.default.gdc.shortcut.path)/gdc-
http.gdc
  </SHORTCUT>
</WEB_APPLICATION_THEME_COMPONENT>
```

Parent elements

This element is a child of one of the following elements: THEME on page 351, WEB_APPLICATION_THEME_COMPONENT on page 359

SNIPPET

The SNIPPET element associates a snippet object identifier with a template to be used by the Genero Web Client snippet-based rendering engine when rendering forms that include the specified object.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Syntax

```
<SNIPPET Id="snipId" Style="mystyle"> snippath </SNIPPET>
```

Syntax notes

1. *snipId* is the object identifier
2. *mystyle* is the value of the STYLE attribute
3. *snippath* is the path to the template snippet file

Usage Example

```
<SNIPPET Id="Edit" Style="FileUpload">
 $(res.path.tpl.html5)/FileUpload.xhtml</SNIPPET>
```

Code Example

```
EDIT f01 = formonly.f01, STYLE="FileUpload";
```

For more information, refer to the *Genero Web Client User Guide*.

Parent elements

This element is a child of one of the following elements: THEME on page 351, WEB_APPLICATION_THEME_COMPONENT on page 359

SOCKET_FAMILY

By default, the Genero Application Server uses UNIX™ domain sockets to communicate between the dispatcher and the proxies on UNIX™ systems. The Genero Application Server uses TCP sockets to communicate between the dispatcher and proxies on Windows™ and with the J2EE dispatcher.

The SOCKET_FAMILY element specified whether UNIX™ domain sockets or TCP sockets are to be used on UNIX™ systems. The configured default is UNIX™. To use TCP sockets, change the value to TCP.

> **Caution:** Forcing TCP sockets on UNIX™ should only be done for debugging purposes, at the request of support.

On Windows™, it is configured to TCP, but it is not used. It must exist, however, as it is required by the schema validating the file.

Usage example

```
<RESOURCE_LIST>
  <WNT>
```

```
    <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">TCP</
RESOURCE>
    <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">C:\temp</
RESOURCE>
  </WNT>
  <UNX>
      <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">UNIX</
RESOURCE>
      <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">/tmp</
RESOURCE>
  </UNX>
...
<INTERFACE_TO_CONNECTOR>
  <TCP_BASE_PORT>$(res.ic.base.port)</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>$(res.ic.port.offset)</TCP_PORT_OFFSET>
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  <TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>
  <SOCKET_FAMILY>$(res.dispatcher.socket.family)</SOCKET_FAMILY>
  <SOCKET_PATH>$(res.dispatcher.socket.path)</SOCKET_PATH>
</INTERFACE_TO_CONNECTOR>
```

Parent elements
This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

SOCKET_PATH

The SOCKET_PATH element defines the directory where UNIX™ domain sockets will be created and stored. By default, it is configured as `/tmp` for UNIX™ systems, `C:\temp` for Windows™)

Usage example

```
<RESOURCE_LIST>
  <WNT>
    <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">TCP</
RESOURCE>
    <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">C:\temp</
RESOURCE>
  </WNT>
  <UNX>
      <RESOURCE Id="res.dispatcher.socket.family" Source="INTERNAL">UNIX</
RESOURCE>
      <RESOURCE Id="res.dispatcher.socket.path" Source="INTERNAL">/tmp</
RESOURCE>
  </UNX>
...
<INTERFACE_TO_CONNECTOR>
  <TCP_BASE_PORT>$(res.ic.base.port)</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>$(res.ic.port.offset)</TCP_PORT_OFFSET>
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  <TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>
  <SOCKET_FAMILY>$(res.dispatcher.socket.family)</SOCKET_FAMILY>
  <SOCKET_PATH>$(res.dispatcher.socket.path)</SOCKET_PATH>
</INTERFACE_TO_CONNECTOR>
```

Parent elements
This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

START

The START element specifies the number of DVMs to start for this Web Service when the GAS starts.

Constraint

START <= MAX_AVAILABLE

Parent elements
This element is a child of one of the following elements: POOL on page 337

SYSTEM_ID

The SYSTEM_ID element is the system identifier of the DTD.

Usage example

```
<SYSTEM_ID>http://www.w3.org/TR/html4/strict.dtd</SYSTEM_ID>
```

Parent elements
This element is a child of one of the following elements: DOC_TYPE on page 312

TCP_BASE_PORT

The TCP_BASE_PORT element specifies the *base* value of the port the Genero Application Server is listening to.

The true port that the Genero Application Server is listening to is the port specified by TCP_BASE_PORT + TCP_PORT_OFFSET.

Usage example

```
<TCP_BASE_PORT>6420</TCP_BASE_PORT>
```

Parent elements
This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

TCP_PORT_OFFSET

The TCP_PORT_OFFSET element specifies the *offset* value of the port the GWC is listening to.

The true port that the Genero Application Server is listening to is the port specified by TCP_BASE_PORT + TCP_PORT_OFFSET.

Usage example

```
<TCP_PORT_OFFSET>75</TCP_PORT_OFFSET>
```

Parent elements
This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

TEMPLATE

TEMPLATE element identifies the main template associated with the application. Its identifier is "_default".

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

Syntax

```
<TEMPLATE Id="_default"> templatePath </TEMPLATE>
```

Syntax notes

1. *templatePath* is the path to the template file

Usage example

```
<TEMPLATE Id="_default">$(res.path.tpl.html5)/main.xhtml</TEMPLATE>
```

Parent elements

This element is a child of one of the following elements: THEME on page 351, WEB_APPLICATION_THEME_COMPONENT on page 359

TEMPORARY_DIRECTORY

The TEMPORARY_DIRECTORY element specifies where to store the transferred files.

Usage example

```
<TEMPORARY_DIRECTORY>/var/tmp</TEMPORARY_DIRECTORY>
```

In this example, the transferred files are stored in /var/tmp.

Parent elements

This element is a child of one of the following elements: INTERFACE_TO_CONNECTOR on page 324

THEME

Attention: As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.

For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

The THEME element defines the theme to be applied to the application. It takes an optional attribute **Using**, in which the unique identifier of a predefined WEB_APPLICATION_THEME_COMPONENT element can be specified.

The THEME element may contain BOOTSTRAP (GWC-HTML5) on page 306, TEMPLATE on page 350, SHORTCUT on page 347, and SNIPPET on page 348 child elements, specifying the various templates and snippets to be used. You can specify multiple theme elements within an application, as different themes can be called by different windows and/or forms. If a template defined in this THEME element has the same unique identifier as a template inherited via a WEB_APPLICATION_THEME_COMPONENT setting, the local THEME element is used. In other words, templates defined explicitly for the application override any templates defined in the WEB_APPLICATION_THEME_COMPONENT that have the same template identifier.

Child elements

The THEME may contain the following elements:

1. Zero or one BOOTSTRAP element.
2. Zero or one TEMPLATE element.
3. Zero or one SHORTCUT element.
4. Zero or more SNIPPET elements.

Usage examples

```
<THEME Using="cpn.theme.default.gwc" />
```

```
<THEME Using="cpn.theme.default.gwc">
  <TEMPLATE Id="_default">/templatedir/deftemp.html</TEMPLATE>
</THEME>
```

For more information, see *Defining a Theme Component*.

Parent elements

This element is a child of one of the following elements: MAP on page 327

TIMEOUT (for a file transfer)

The TIMEOUT element specifies the number of seconds that transferred files are available after application end and before they are deleted. The duration is always specified in seconds.

Usage example

```
<TIMEOUT>600</TIMEOUT>
```

By default, the timeout duration is set to 600 seconds.

Parent elements

This element is a child of one of the following elements: FILE_TRANSFER on page 317

TIMEOUT (for an application)

The TIMEOUT element sets the timeouts for the application. You can reference a predefined WEB_APPLICATION_TIMEOUT_COMPONENT to inherit the timeout settings of that component by including the **Using** attribute, specifying the unique identifier for that timeout component, and/or you can set individual timeout elements specific to the application.

Settings defined locally within the TIMEOUT element override settings defined in included timeout components.

Child elements

The TIMEOUT element may contain the following child elements:

1. Zero or one USER_AGENT element.
2. Zero or one REQUEST_RESULT (for an application) on page 341 element.
3. Zero or one DVM_AVAILABLE element.
4. Zero or one DVM_PINGTIMEOUT element.

For more information, see WEB_APPLICATION_TIMEOUT_COMPONENT.

Usage examples

```
<TIMEOUT Using="cpn.wa.timeout.set1" />
```

```
<TIMEOUT>

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

Parent elements

This element is a child of one of the following elements: UA_OUTPUT on page 354, MAP on page 327

TIMEOUT (for a service)

The TIMEOUT element sets the timeouts for the Web services application. You can reference a predefined SERVICE_APPLICATION_TIMEOUT_COMPONENT to inherit the timeout settings of that component by including the **Using** attribute, specifying the unique identifier for that timeout component, and/or you can set individual timeout elements specific to the application.

Settings defined locally within the TIMEOUT element override those settings defined in a referenced SERVICE_APPLICATION_TIMEOUT_COMPONENT.

Child elements

Possible timeout elements include:

1. Zero or one DVM_AVAILABLE element.
2. Zero or one KEEP_ALIVE element.
3. Zero or one REQUEST_RESULT (for a service) on page 341 element.

For more information on setting timeout parameters, refer to TIMEOUT (for an application) on page 352.

Usage examples

```
<TIMEOUT Using="cpn.ws.timeout.set1" />
```

```
<TIMEOUT>
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
  <KEEP_ALIVE>240</KEEP_ALIVE>
  <REQUEST_RESULT>60</REQUEST_RESULT>
</TIMEOUT>
```

Parent elements

This element is a child of one of the following elements: APPLICATION (for a service) on page 301

TIMEOUT (for auto logout)

The TIMEOUT element specifies how long (in seconds) the DVM waits from when it detects an application has no user activity before it triggers an auto logout event.

After this timeout period elapses, the front end client program gets a log out message, see AUTO_LOGOUT on page 304.

Syntax

```
<TIMEOUT>timeoutSeconds</TIMEOUT>
```

Usage example

```
<TIMEOUT>0</TIMEOUT>
```

A timeout duration set to 0 seconds means the auto logout is ignored and the application keeps running. A correct configuration requires that the TIMEOUT be set as required to enable auto logout when there is no user activity.

Parent elements

This element is a child of the following element: AUTO_LOGOUT_COMPONENT on page 305

UA_OUTPUT

The UA_OUTPUT element specifies the configuration parameters for a UA proxy rendered application.

Starting with Genero 3.00, all UI applications delivered by the Genero Application Server are rendered by the UA proxy, to include the Genero Desktop Client (GDC), Genero Web Client (GWC) for JavaScript, Genero Mobile for Android and Genero Mobile for iOS. These applications are configured in part by the UA_OUTPUT element.

Note: Applications delivered by GWC for HTML5 continue to use the OUTPUT element instead.

Syntax

```
<UA_OUTPUT>
  <PROXY>executable-name</PROXY>
  <PUBLIC_IMAGEPATH>image-path</PUBLIC_IMAGEPATH>
  <GWC-JS>gwc-js-directory</GWC-JS>
  <TIMEOUT>...</TIMEOUT>
</UA_OUTPUT>
```

Child elements

The UA_OUTPUT element may contain the following child elements:

1. One PROXY (for an application) on page 338 element.
2. One PUBLIC_IMAGEPATH on page 338 element.
3. Zero or one GWC-JS on page 319 element.
4. Zero or one TIMEOUT (for an application) on page 352 element.

Usage example

```
<UA_OUTPUT>
  <PROXY>$(res.uaproxy.cmd)</PROXY>
  <PUBLIC_IMAGEPATH>$(res.public.resources)</PUBLIC_IMAGEPATH>
  <GWC-JS>gwc-js</GWC-JS>
  <TIMEOUT Using="cpn.wa.timeout"/>
</UA_OUTPUT>
```

Parent elements

This element is a child of one of the following elements: APPLICATION (for an application) on page 301

UNX

This element contains a collection of UNIX™-specific RESOURCE elements.

The UNX element contains a list of those resources that are only available on UNIX™ operating systems. There is no difference between UNIX™ systems like Linux®, AIX®, HP-UX, Solaris, and so on.

Child elements

A UNX element may contain the following child elements:

- Zero to many RESOURCE elements.

Parent elements

This element is a child of one of the following elements: RESOURCE_LIST on page 343

USER_AGENT

The USER_AGENT configuration parameter specifies the number of seconds the Genero Application Server is to wait for a client request before assuming that the Front End client has died or that there has been a network failure.

Under normal operation, the Front End client sends a GET request to the Genero Application Server immediately after receiving a response. The client-side front end (CSF) will also send keep-alive requests (/wa/ka) to keep the application alive in the case of user inactivity. If the client has not sent a request to the Genero Application Server before the USER_AGENT timeout expires, the Genero Application Server assumes that the front end client has died and asks the application's DVM to shut down.

With the Genero Desktop Client Active X, the USER_AGENT timeout usually does not expire. When the user closes the application, the DVM handling that application is properly shut down.

The USER_AGENT timeout proves to be particularly useful with the Genero Web Client. As with the other front end clients, 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 client has died. This can occur with the Genero Web Client when a user closes the browser instead of properly exiting the application; the Front End client has no mechanism to tell the Genero Application Server that the user has closed his browser. With the USER_AGENT timeout, the Genero Application Server closes the socket to the DVM, which causes the DVM to shut down.

Usage example

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

In this example, the User Agent timeout is set to 300 seconds.

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_TIMEOUT_COMPONENT on page 360, TIMEOUT (for an application) on page 352

VARIABLE

Zero or one VARIABLE. Define a session variable and optionally the initial value.

Syntax

```
<VARIABLE Id="varId"> val </VARIABLE>
```

Syntax notes

1. *varId* is the variable name.

2. *val* is the variable value.

Example

```
  <VARIABLE Id="var7" />
  <VARIABLE Id="var8">Initial value</VARIABLE>
```

Parent elements

This element is a child of one of the following elements: HTTP_COOKIE on page 321

WEB_APPLICATION_EXECUTION_COMPONENT

The WEB_APPLICATION_EXECUTION_COMPONENT defines a set of execution parameters that are used when starting the Web application. It takes an attribute **Id**, which specifies the unique identifier for this set of execution definitions. It is this unique identifier that is referenced by an application, providing that application with its set of execution parameters. The attribute **AllowUrlParameters** defines whether the parameters provided in the request query string should be ignored ("FALSE", default value) or provided to the DVM ("TRUE").

Syntax

```
<WEB_APPLICATION_EXECUTION_COMPONENT Id="compId"
 [ AllowUrlParameters="allowParam" ] >
 [ <ENVIRONMENT_VARIABLE Id="envId" > env </ENVIRONMENT_VARIABLE> [...] ]
 [ <PATH> path </PATH> ]
 [ <DVM> dvm </DVM> ]
 [ <MODULE> module </MODULE> ]
 [ <PARAMETERS> parameterSettings </PARAMETERS> ]
 [ <ACCESS_CONTROL> accessSettings </ACCESS_CONTROL> ]
 [ <DELEGATE> delegateSettings </DELEGATE> ]
 [ <WEB_COMPONENT_DIRECTORY> webComponent </WEB_COMPONENT_DIRECTORY> ]
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

Child elements

The WEB_APPLICATION_EXECUTION_COMPONENT element may contain the following child elements:

1. Zero or more ENVIRONMENT_VARIABLE elements.
2. Zero or one PATH element.
3. Zero or one DVM element.
4. Zero or one MODULE element.
5. Zero or one PARAMETERS element.
6. Zero or one ACCESS_CONTROL element.
7. Zero or one DELEGATE element.
8. Zero or one WEB_COMPONENT_DIRECTORY on page 361 element.

Example

```
<WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
    <ENVIRONMENT_VARIABLE Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="INFORMIXDIR">$(res.informixdir)</
ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE Id="INFORMIXSERVER">$(res.informixserver)</
ENVIRONMENT_VARIABLE>
    ...
    <DVM>$(res.dvm.wa)</DVM>
```

```
    ...
   <WEB_COMPONENT_DIRECTORY>$(connector.uri)/ua/i/$(application.path)/
webcomponents</WEB_COMPONENT_DIRECTORY>
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

Parent elements
This element is a child of one of the following elements: COMPONENT_LIST on page 308

WEB_APPLICATION_HTTP_COOKIES_COMPONENT
This element defines persistent session variables or constants.

The WEB_APPLICATION_HTTP_COOKIES_COMPONENT element defines persistent session variables or constants. It takes an Id attribute, which specifies the unique identifier for this component. It is this unique identifier that is referenced by an application configuration.

Syntax

```
<WEB_APPLICATION_HTTP_COOKIES_COMPONENT Id="compId">
  <HTTP_COOKIE Id="cid" [Expires="endTime" | Domain="mydomain" |
   Secure="TRUE|FALSE" | HttpOnly=""]>
    <VARIABLE Id="varId">val</VARIABLE> [...]
    <CONSTANT Id="cstId">cst</CONSTANT> [...]
  <HTTP_COOKIE> [...]
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Syntax notes

1. *compId* is the unique identifier for this component (required).

Child elements

The WEB_APPLICATION_HTTP_COOKIES_COMPONENT element may contain the following child elements:

1. Zero or one HTTP_COOKIE elements.

Parent elements
This element is a child of one of the following elements: COMPONENT_LIST on page 308

WEB_APPLICATION_PICTURE_COMPONENT
This element defines how images are served by the GAS.

The WEB_APPLICATION_PICTURE_COMPONENT element specifies the directory from which images are served. It takes an Id attribute, which specifies the unique identifier for this Picture component. It is this unique identifier that is referenced by an application configuration, providing that application with the location of its image directory or directories.

Syntax

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
  <PATH Id="pathID" Type="pathType">Path</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Syntax notes

1. *resID* is the unique identifier for this picture component.

Child elements

The WEB_APPLICATION_PICTURE_COMPONENT may contain the following child elements:

1. Zero to many PATH elements.

Parent elements

This element is a child of one of the following elements: COMPONENT_LIST on page 308

WEB_APPLICATION_RENDERING_COMPONENT

A rendering component defines how an application is rendered for delivery via the Web to the front-end client.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

The WEB_APPLICATION_RENDERING_COMPONENT element specifies the output driver and other elements that determine how an application is to be rendered for a Web application (GWC-HTML5). It takes an attribute **Id**, which specifies the unique identifier for this Rendering component. It is this unique identifier that is referenced by an application, specifying the output driver that the application should use.

Syntax

```
<WEB_APPLICATION_RENDERING_COMPONENT Id="compId">
  [<OUTPUT_DRIVER> outputDriver </OUTPUT_DRIVER>]
  [<XML_DECLARATION> xmlDec </XML_DECLARATION>]
  [<HTTP_RESPONSE_ENCODING Source="REQUEST|INLINE">
    httpResponseEnc
   </HTTP_RESPONSE_ENCODING>]
  [<HTTP_REQUEST_ENCODING Source="REQUEST|INLINE">
    httpRequestEnc
    </HTTP_REQUEST_ENCODING>]
  [<MIME_TYPE> mimetype </MIME_TYPE>]
  [<DOC_TYPE> doctype </DOC_TYPE>]
</WEB_APPLICATION_RENDERING_COMPONENT>
```

Child elements

The WEB_APPLICATION_RENDERING_COMPONENT may contain the following child elements:

1. Zero or one OUTPUT_DRIVER element.
2. Zero or one XML_DECLARATION element.
3. Zero or one HTTP_RESPONSE_ENCODING element.
4. Zero or one HTTP_REQUEST_ENCODING element.
5. Zero or one MIME_TYPE element.
6. Zero or one DOC_TYPE element.

Example

```
<WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.gwc">
  <OUTPUT_DRIVER>GWC</OUTPUT_DRIVER>
  <HTTP_RESPONSE_ENCODING Source="INLINE">ISO-8859-1</
HTTP_RESPONSE_ENCODING>
  <HTTP_REQUEST_ENCODING Source="INLINE">ISO-8859-1</HTTP_REQUEST_ENCODING>
```

```
  <MIME_TYPE>text/html</MIME_TYPE>
</WEB_APPLICATION_RENDERING_COMPONENT>
```

```
<WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.gwc2">
  <OUTPUT_DRIVER>GWC2</OUTPUT_DRIVER>
  <HTTP_RESPONSE_ENCODING Source="REQUEST"/>
  <HTTP_REQUEST_ENCODING Source="REQUEST"/>
  <MIME_TYPE>application/xml</MIME_TYPE>
</WEB_APPLICATION_RENDERING_COMPONENT>
```

Parent elements

This element is a child of one of the following elements: COMPONENT_LIST on page 308

WEB_APPLICATION_THEME_COMPONENT

This element specifies the theme (set of templates) that the application or set of applications can use.

> **Attention:** As of Genero version 3.00, the Snippet-Based Rendering Engine (SBRE) and all themes using template paths are deprecated. Output maps (e.g. DUA_HTML5) are no longer used to specify output theme, as the *wa* protocol did previously. For information on how templates and snippets were used by the Front End clients, please refer to the *Genero Application Server 2.50 User Guide*.
>
> For new development, use GWC for JavaScript, see Genero Web Client for JavaScript (GWC-JS) on page 173.

A theme component defines the theme that an application or set of applications can use. A theme is made up of one main template and multiple template snippets that will drive the rendering of snippet-based rendering engine (SBRE) components. The same snippet identifier can appear more than one time, but each one must be associated with an unique style name, so that each template snippet is uniquely identified by the identifier/style couple. If no template snippet exists for the specified style, the default one is used.

The WEB_APPLICATION_THEME_COMPONENT element specifies the theme (set of templates) that the application or set of applications can use. It takes an attribute Id, which specifies the unique identifier for this Template component. It is this unique identifier that is referenced by a Web application to make accessible the list of templates defined therein.

Syntax

```
<WEB_APPLICATION_THEME_COMPONENT Id="compId" >
   [ <BOOTSTRAP Id="_default" > bootstrapPath </BOOTSTRAP > ]
   [ <TEMPLATE Id="_default" > templatePath </TEMPLATE > ]
   [ <SHORTCUT Id="_default" > shortcutPath </SHORTCUT > ]
   [ <SNIPPET Id="snipId" [ Style="stylename" ] > snippath </SNIPPET> ]
 [...]
</WEB_APPLICATION_THEME_COMPONENT>
```

Child elements

The WEB_APPLICATION_THEME_COMPONENT may contain the following elements:

1. Zero or one BOOTSTRAP element.
2. Zero or one TEMPLATE element.
3. Zero or one SHORTCUT element.
4. Zero or more SNIPPET elements.

Example

```
<WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.html5.gwc">
  <BOOTSTRAP Id="_default">$(res.path.tpl.html5)/bootstrap.xhtml</BOOTSTRAP>
  <TEMPLATE Id="_default">$(res.path.tpl.html5)/main.xhtml</TEMPLATE>
  <SNIPPET Id="UIFrame">$(res.path.tpl.html5)/UIFrame.xhtml</SNIPPET>
  ...
</WEB_APPLICATION_THEME_COMPONENT>
```

Parent elements

This element is a child of one of the following elements: COMPONENT_LIST on page 308

WEB_APPLICATION_TIMEOUT_COMPONENT

The WEB_APPLICATION_TIMEOUT_COMPONENT element creates a Web application timeout component, which define a set of timeout values to be used when configuring a Web application. It takes an attribute **Id**, which specifies the unique identifier for this set of timeout definitions. It is this unique identifier that is referenced with an application's configuration, providing that application with a set of timeout values.

Why are Web application timeouts necessary? When a Front End client connects to a DVM via the Genero Application Server (GAS), the connection between the Front End client and the GAS is not persistent (although the connection between the GAS and the DVM is persistent). The Genero Application Server needs the timeout settings to determine whether these components have remained alive and that communication can continue between the two.

The Front End client can send two types of requests to the DVM: a POST request when sending data to the DVM and a GET request when asking whether there is data to retrieve. The Genero Application Server, however, cannot send a request to the Front End client because the Front End client does not have a public address.

As a result, a request is always initiated by the Front End client and the server response is done with the same connection. Between requests, the Genero Application Server stores data sent from the DVM in its buffer and keeps it for the next GET request from the Front End client.

Syntax

```
<WEB_APPLICATION_TIMEOUT_COMPONENT Id="appTimeOutID">
 [ <USER_AGENT> uaTimeOut </USER_AGENT> ]
 [ <REQUEST_RESULT> requestTimeOut </REQUEST_RESULT>]
 [ <DVM_AVAILABLE> dvmTimeOut </DVM_AVAILABLE> ]
 [ <DVM_PINGTIMEOUT> dvmPingTimeOut </DVM_PINGTIMEOUT> ]
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

Child elements

The WEB_APPLICATION_TIMEOUT_COMPONENT element may contain the following child elements:

1. Zero or one USER_AGENT element.
2. Zero or one REQUEST_RESULT element.
3. Zero or one DVM_AVAILABLE element.
4. Zero or one DVM_PINGTIMEOUT element.

Example

```
<WEB_APPLICATION_TIMEOUT_COMPONENT Id="cpn.wa.timeout.set1">
  <USER_AGENT>300</USER_AGENT>
  <REQUEST_RESULT>60</REQUEST_RESULT>
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
  <DVM_PINGTIMEOUT>300</DVM_PINGTIMEOUT>
```

```
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

In this example, the **Id** value - **cpn.wa.timeout.set1** - can be referenced when defining an application. When an application references a component by its **Id** value, it inherits the settings defined by that component.

Parent elements

This element is a child of one of the following elements: COMPONENT_LIST on page 308

WEB_COMPONENT_DIRECTORY

The WEB_COMPONENT_DIRECTORY element specifies the path where Web components for an application are located.

Starting with Genero 3.00, this element added to the URL, builds the path used to find a Web component. It defines paths from where Web components are served via the GAS. The WEB_COMPONENT_DIRECTORY configuration only applies to Genero Desktop Client (GDC) applications delivered via GAS, and Genero Web Client for JavaScript (GWC-JS).

> **Note:** For your legacy GWC-HTML5 applications, the WEB_COMPONENT_DIRECTORY configuration entry is ignored. Web components in this case are still required to be located in *$(FGLASDIR)/*web/components as before. For more details on web component usage, see the *Genero Business Development Language User Guide*.

Syntax

```
<WEB_COMPONENT_DIRECTORY>path;path</WEB_COMPONENT_DIRECTORY>
```

It allows for multiple paths to be specified, the separator used between resource paths is a semi-colon, ';'.

Usage example

```
   <WEB_APPLICATION_EXECUTION_COMPONENT Id="cpn.wa.execution.local">
     ...
      <DELEGATE service="MyGroup/MyDelegateService"> ... </DELEGATE>
      <WEB_COMPONENT_DIRECTORY>$(application.path)/webcomponents;
$(my.web.components)/static-files/webcomponents</WEB_COMPONENT_DIRECTORY>
   </WEB_APPLICATION_EXECUTION_COMPONENT>
```

> **Important:** Element order. If the WEB_COMPONENT_DIRECTORY element is present, it must be set in the correct order within the parent element, see EXECUTION (for an application) on page 315, EXECUTION (for a service) on page 316, or WEB_APPLICATION_EXECUTION_COMPONENT on page 356.

Child elements

There are no child elements.

Parent elements

This element is a child of one of the following elements: EXECUTION (for an application) on page 315, EXECUTION (for a service) on page 316, WEB_APPLICATION_EXECUTION_COMPONENT on page 356

WNT

This element contains a collection of Windows™-specific RESOURCE elements.

The WNT element contains a list of Windows NT™ resources: those resources are only available on the Windows™ operating systems.

Child elements

A WNT element main contain the following child elements:

- Zero to many RESOURCE elements.

Parent elements

This element is a child of one of the following elements: RESOURCE_LIST on page 343

XML_DECLARATION

The XML_DECLARATION element specifies if the xml declaration is present or not in the document response. This element only works with OUTPUT_DRIVER=GWC2.

Usage example

```
<XML_DECLARATION>TRUE</XML_DECLARATION>
```

Parent elements

This element is a child of one of the following elements: WEB_APPLICATION_RENDERING_COMPONENT on page 358, RENDERING on page 339

Glossary and Acronyms

In this section, many terms and acronyms used throughout this document are briefly defined.

For more details on the various web technology terms and acronyms found on this page, visit www.w3.org.

Table 64: Terminology

Term	Description
CSS	Cascading style sheets. CSS is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents.
DTD	Document Type Definition. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.
DUA	Driver User Agent.
DVM	The Dynamic Virtual Machine or Runtime System that is installed on the Application Server and executes the application program.
GAD	Genero ADministration Application. This application is accessed by clicking on the Administration tab on the Genero Application Server Welcome Page (demos.html).
GAS	Genero Application Server. Defined by the computer system that houses the Dynamic Virtual Machine (DVM).
GDC	Genero Desktop Client.
GDCAX	Genero Desktop Client / Active X.
GWC	Genero Web Client. A client technology that renders the application in an HTML Graphical User Interface (browser).
GWS	Genero Web Services. A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows™ applications can talk with UNIX™ applications. See also SOA.
HTML	Hyper Text Markup Language. An HTML file is a text file containing markup tags. The markup tags tell the Web browser how to display the page.
JavaScript™	JavaScript™ is a scripting language designed to add interactivity to HTML pages. A JavaScript™ consists of lines of executable computer code that can be embedded directly into HTML pages. It is an interpreted language, meaning that the scripts execute without preliminary compilation. Most browsers support JavaScript™, and anyone can use JavaScript™ without purchasing a license.
SOA	Service-Oriented Architecture. In SOA, autonomous, loosely-coupled and coarse-grained services with well-defined interfaces provide business functionality and can be discovered and accessed through a supportive infrastructure. This allows internal and external system integration as well as the flexible reuse of application logic through the composition of services to support an end-to-end business process.
SUA	Sending User Agent.

Term	Description
User Agent	A User Agent is a client agent. It can be a browser or the Genero Desktop Client.
Web Server	A computer that delivers (serves up) Web pages. Every web server has an IP address and possibly a domain name. For example, if you enter the URL http://www.mycompany.com in your browser, this sends a request to the server whose domain name is mycompany.com. The server then fetches the home page and sends it to your browser. Any computer can be turned into a web server by installing server software and connecting the machine to the Internet.
WSDL	Web Services Description Language. WSDL is an XML-based language for describing Web services and how to access them.
XML	Short for Extensible Markup Language, a specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. For more information, please refer to the W3C web site at www.w3.org.
XML Schema	XML Schema is an XML-based alternative to a DTD . An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).
XHTML	EXtensible HyperText Markup Language. XHTML is aimed to replace HTML. XHTML is almost identical to HTML 4.01. XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application.
XPath	XPath is a language for navigating in XML documents.
XSD	See XML Schema.
XSL	XML Style Sheets. XML does not use predefined tags (you can use any tag names you wish), and the meaning of these tags are not well understood. For example, a <table> element could mean an HTML table, a piece of furniture, or something else - and a browser does not know how to display it. XSL describes how the XML document should be displayed.
XSLT	XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.

Legal Notices

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software developed by CollabNet (http://www.Collab.Net/).

This product includes software developed by the University of California, Berkeley and its contributors.

This product includes software developed or owned by Caldera International, Inc