User Guide
Version 2.11

Note: This documentation is for Genero 2.11. See the corresponding on-line documentation at the Web site **http://www.4js.com/online_documentation** for the latest updates. Please contact your nearest support center if you encounter problems or errors in the on-line documentation.

Table Of Contents

Genero Application Server Overview

Topics

What is the Genero Application Server?

Genero Application Server (GAS) is an engine that **delivers Genero applications**. It creates relationships between various front-ends (the Genero Desktop Client Active X, the Genero Java Client and the Genero Web Client) and the DVMs which run the applications. The GAS also manages a pool of DVMs for Web Services applications. *See also* Front-ends and Extensions.

This Application Server is interfaced with a Web Server to handle requests from the Internet. Communication between the Web Server and the GAS daemon is handled by the **Application Server Connector**.

GAS Daemon

With the support of the http protocol, the GAS daemon provides a **direct connection** for access to applications without using a Web Server. During the development phase, you can exclude the Web Server from your development architecture.

GAS simplifies the deployment phase by taking care of the connection to the applications. No software installation or configuration is needed on the end user's side; a simple **browser** is all that is required to access the program.

GAS Connector

The GAS connector sends requests from the Web Server to the correct GAS daemon. Several GAS daemons can be configured to load balance the requests. The configuration of GAS daemons is done in the connector.xcf file (the GAS connector configuration file), found in the script directory of the Web Server.

There are two kinds of connectors: **CGI** and **ISAPI**. While they use different technologies, they both play the same role and use the same GAS connector configuration file. The ISAPI connector is only available on Windows platforms.

Front-ends and Extensions

The Genero Application Server can serve your applications with these various front-ends and extensions.

- **Genero Desktop Client ActiveX**

  Genero Desktop Client Active X (GDC/AX) allows you to run the Genero Desktop Client via a Web site or http address. The first time you access the Client in Active X mode, the GDC/AX software installs itself on your client machine and creates a shortcut in Windows Start Menu. The GDC can then be executed from the web page or directly using the shortcut.

  For more information and installation instructions, refer to the *Genero Desktop Client Manual*.

- **Genero Web Client**

  The Genero Web Client (GWC) allows you to deliver Genero applications in a Web browser, without having to install any software on the client machine. Starting with 2.11, the GWC is included in the installation for the GAS. When working with the GWC, scan the GWC-specific sections in the Table of Contents.

  For a more detailed overview of the GWC, see What is the Genero Web Client?

- **Genero Web Services Extension**

  The Genero Web Services Extension (GWS) allows you to implement Web services. Web services are a standard way of communicating between applications over the Internet or Intranet. A web service can be a server that exposes services, or a client that consumes a service.
  For more information and installation instructions, refer to the *Genero Web Services Extension Manual*.

- **Genero Java Client**

  Genero Java Client (GJC) is a graphical front-end for delivering Genero applications. It is written in Java and can be run under any operating system which supports J2SE Java Runtime Environment (JRE) version 1.4.2 or higher. GJC can also be embedded into an HTML page thanks to an Applet, which will work on any browser that supports Java Plug-in technology with a 1.4.2 or higher J2SE Java Runtime Environment.

For more information and installation instructions, refer to the *Genero Java Client Manual*.

GAS Deployment Architecture

Topics

- GAS Architecture
- Connection Types
- High Availability
- Services Pool (GWS only!)

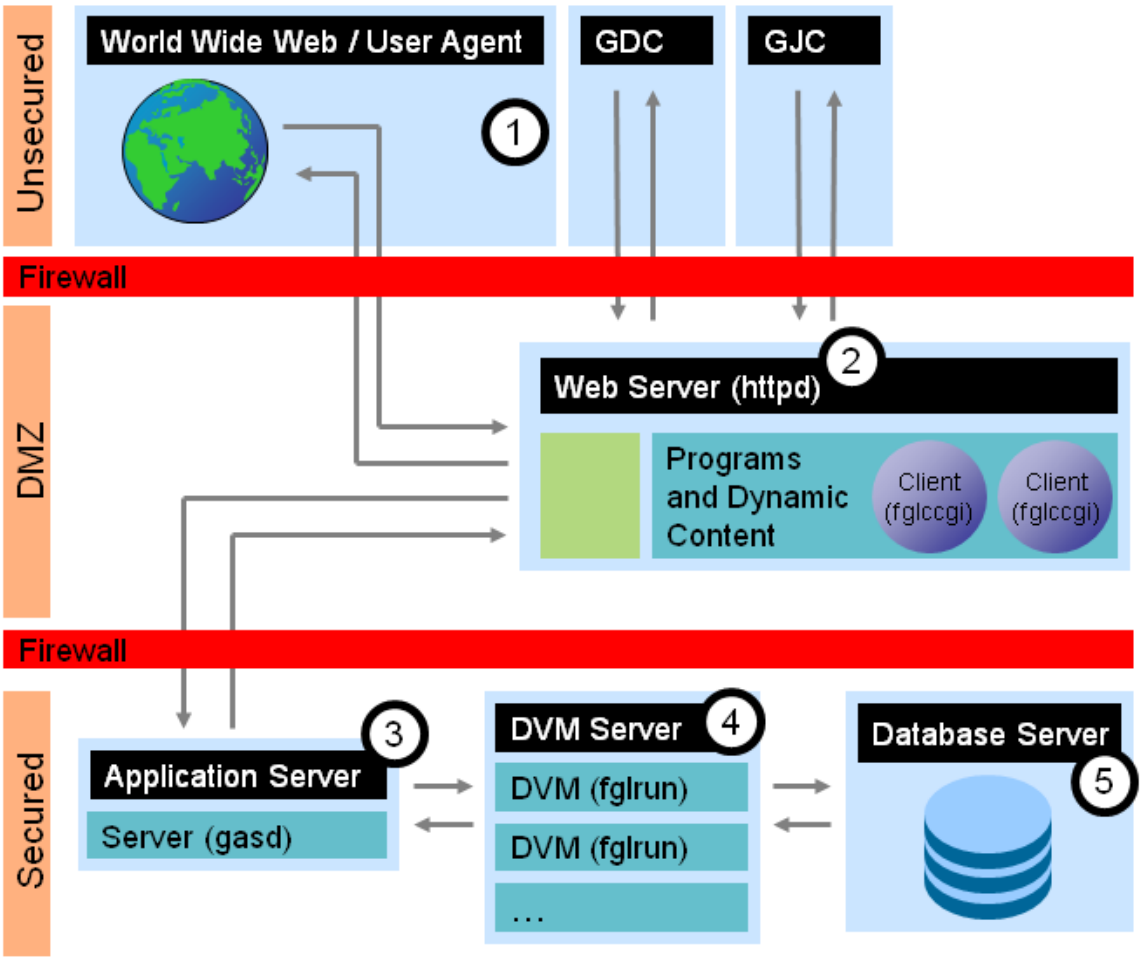
GAS Architecture

This section looks at the following topics:

- Components involved in the GAS solution - Detailing the path from the DVM to the browser and back.
- Third-party software requirements
- Component relationships - The relationship between the user agent, the GAS, the Dynamic Virtual Machine (DVM), and the files required by each.

Components involved in the GAS solution

The GAS works with a user agent/front end, a Web server, the Genero Web Client daemon (gasd), a Dynamic Virtual Machine (DVM), and a database server to provide Web applications to the user.



The components involved are shown in the diagram above.

- A *user agent* (1) or the *GDC* (1) or the *GJC* (1) initiates a request through a *Web server* (2).
- The Web server spans and communicates with the client *CGI Connector*, an executable named either **fglccgi** or **fglcisapi**. The GAS Connector configuration is specified in the file connector.xcf.
- The Connector handles communication with the *Application Server*, also called the GAS daemon (3). The GAS daemon is a process named **gasd**. The gasd configuration is set in the file as.xcf (default) or a user-specified configuration file. The gasd must be started and listening for requests from the GAS Connector.
- Upon receiving a request, the gasd selects the next available port (as defined in the gasd configuration file) and starts a DVM (4).
- The DVM runs the BDL program, which in turn interacts with the specified database (5).

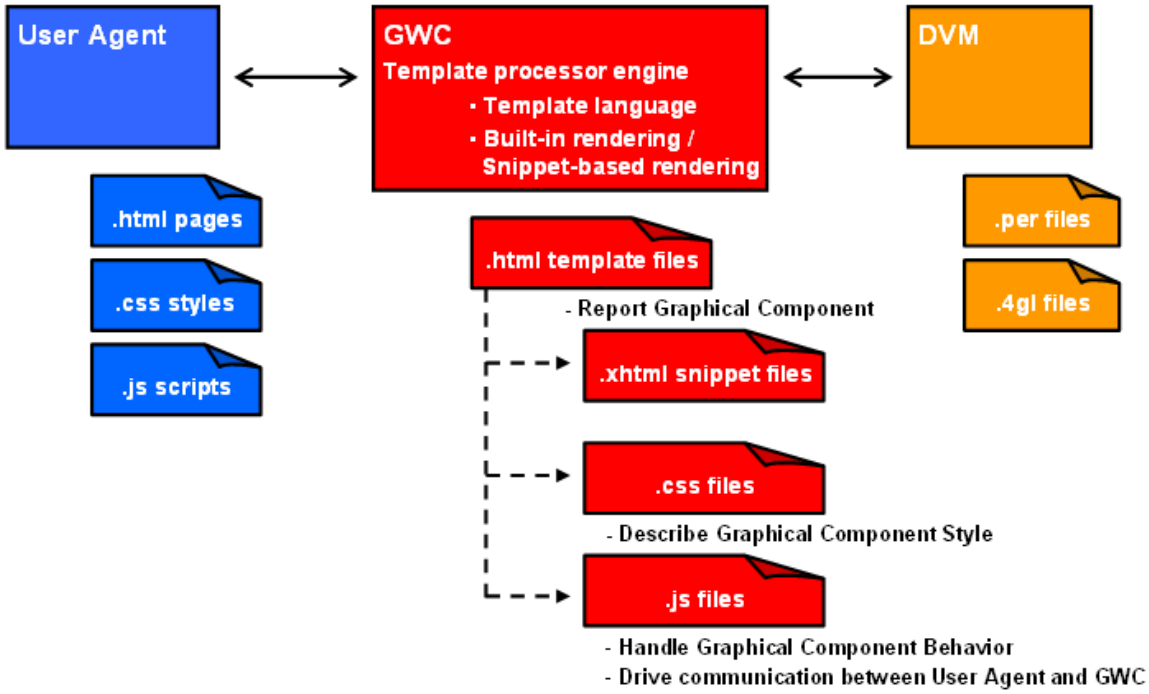
Communication is bi-directional, with information flowing back to the user agent.

In development, it is typical to have the user agent (browser) connect directly to the application server, bypassing the Web server and Connector. For production, it is recommended that you include the Web server in your GAS solution.

Third-party software requirements

The user agent, Web server, Genero BDL, and database server are not included. For information about supported third-party software, refer to System Requirements (in the section **Installation**).

Component Relationships



The diagram above provides another look at connections between a user agent, the client front end, and a DVM. It also identifies which files are needed by each engine.

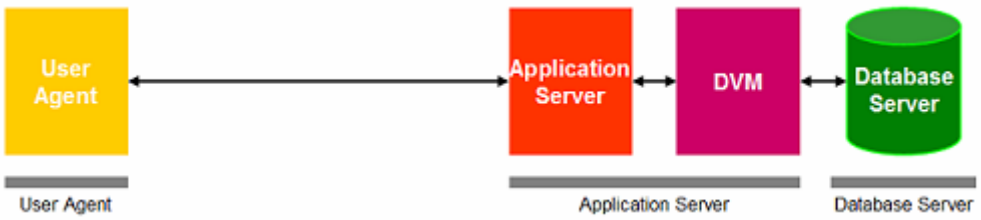
Connection Types

When running an application, there are two methods of connecting to the application server:

- Connect directly to the application server
- Connect to the application server through a Web server.

Connect directly to the application server

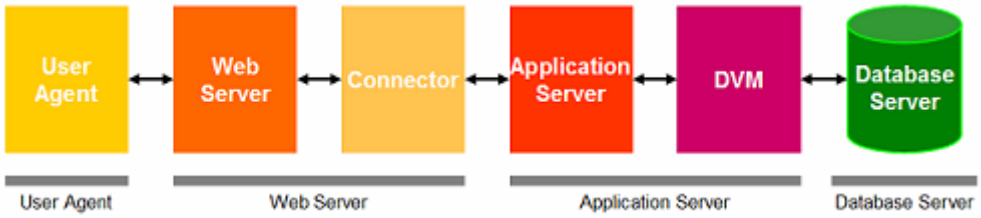
Direct connection allows the user agent to connect directly to the Application Server, without using a Web Server. Direct connection is provided to simplify the architecture of development environments. It is not recommended for production environments.



Notes:

- Connecting directly to the application server is the typical connection method used in development environments.
- A direct connection is always much faster than connecting through a Web server, as it removes the routing of the request through the Web server and GAS Connector.

Connect to the application server through a Web server



When you connect through a Web Server, the GAS Connector routes requests from the Web server to an application server. Connectors are available in two forms:

- As a Common Gateway Interface (CGI) executable, usable on any CGI 1.1 Web servers.
- As an Internet Information Server (IIS) plug-in, usable on any IIS web server (version 5.x or greater).

Notes

- To use the HTTPS protocol, you must connect through a Web Server. Native support of HTTPS by the application server is not supported at this time
- Two types of Connectors are available:
    - Common Gateway Interface (CGI) executables, which are usable on any CGI 1.1 web servers. The CGI connector executables are named `fglccgi` under Unix systems and `fglccgi.exe` under Windows systems.

For example, if you installed the connector in the cgi-bin directory of your web server, you'll access your application using the URL:

```
http://server:port/cgi-bin/fglccgi/wa/r/application
```

or for Windows systems:

```
http://server:port/cgi-bin/fglccgi.exe/wa/r/application
```

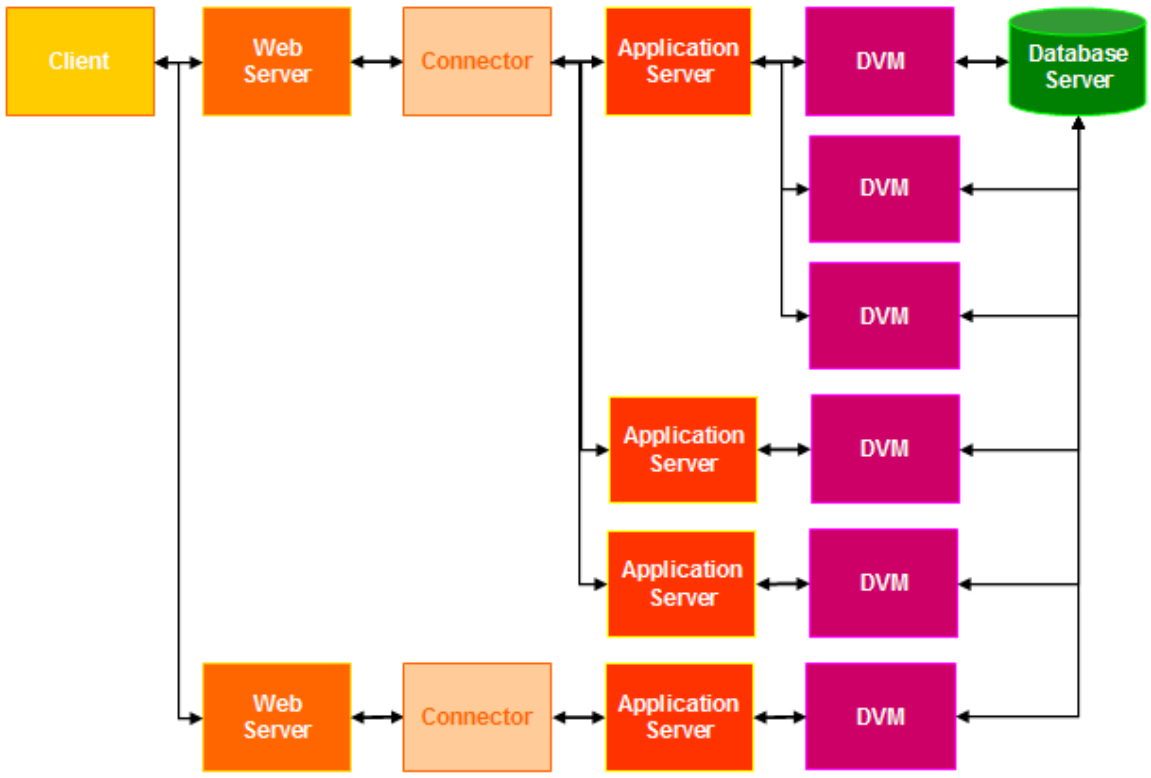
- Internet Information Server (IIS) plug-in, usable on any IIS web server since version 5.

  The IIS connector is named `fglcisapi.dll`. To access to your application through this connector, you use the syntax:

  ```
  http://server:port/scripts/fglcisapi.dll/wa/r/application
  ```

High Availability

The following diagram illustrates the possible path of an application request.



The Client sends a request to the Web Server. There are two methods that the client might use to identify which Web Server to send the request to. It can make a DNS request, and the DNS request can specify which Web Server to send the request to. The DNS database can be updated while monitoring the Web Servers for availability. The other solution is to use hardware (layer switch) that dispatches the load and monitors the system.

The Web Server starts the GAS Connector (which is on the same physical machine as the Web Server). For the initial connection, the GAS Connector reads from its configuration file (`connector.xcf`) and randomly chooses from the list of available Application Servers. If the selected Application Server does not respond, it attempts to connect to the next available Application Server, and so on until a successful connection is established.

The Application Server attaches to an available Dynamic Virtual Machine (DVM) to handle the request. If no DVM is available to process the request, the Application Server attempts to create a new DVM to process the request.

The Virtual Machine connects to the Database (or Database Cluster or Replica).

Services Pool (GWS Only!)

When you define a Web Service application, you specify execution parameters that determine the number of DVMs available at any one time to service a request for that Web Service. These parameters are defined in the Application Server configuration file.

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 (and inherit the runtime environment settings defined for that component) or the individual execution elements can be explicitly set for the application.

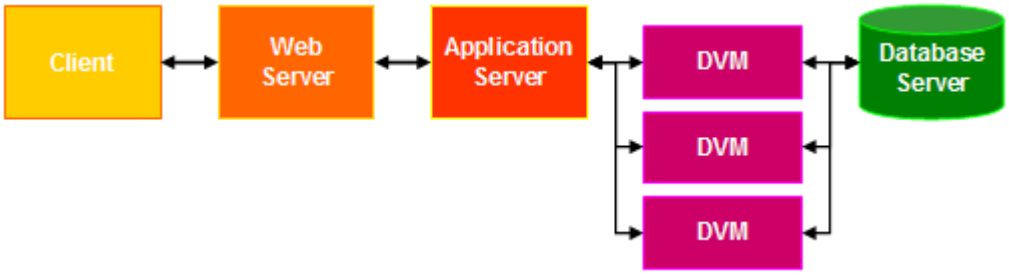
Within the EXECUTION element, the POOL element sets the limitations regarding the number of Virtual Machines that are attached to a Web Service. You specify three values within a POOL element:

- 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.

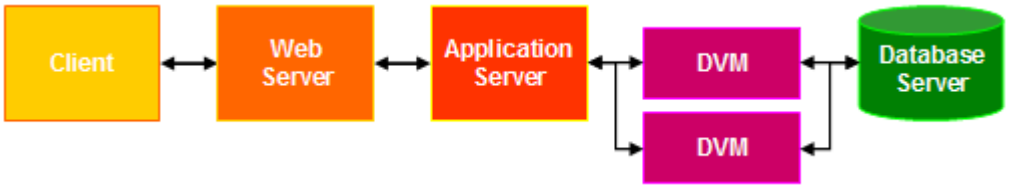
For the discussion that follows, assume the following values have been specified for the three pool elements for a Web Service applicaiton:

```
START=3
MIN_AVAILABLE=2
MAX_AVAILABLE=5
```

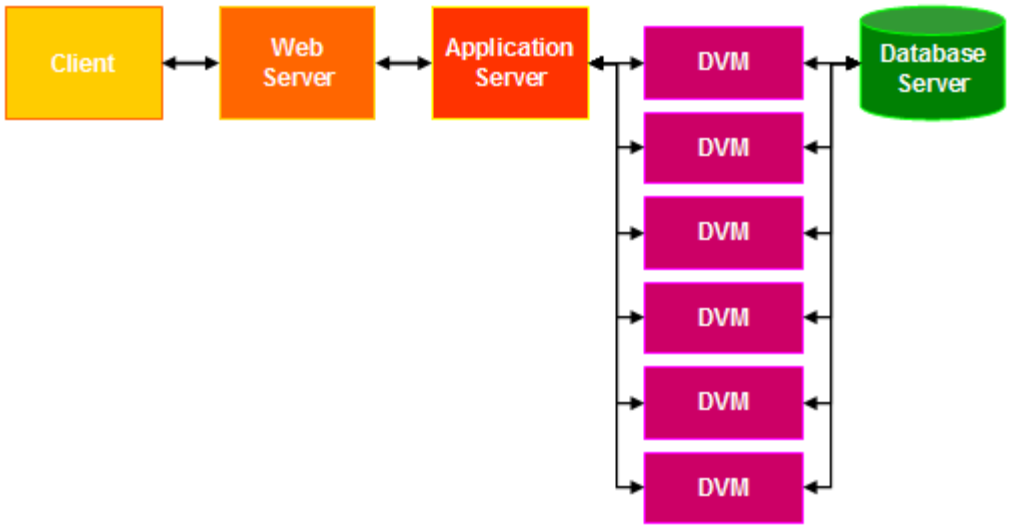
When the Genero Application Server first starts, the START element defines how many DVMs to start for a particular Web service. For our example, this means that 3 DVMs are started.



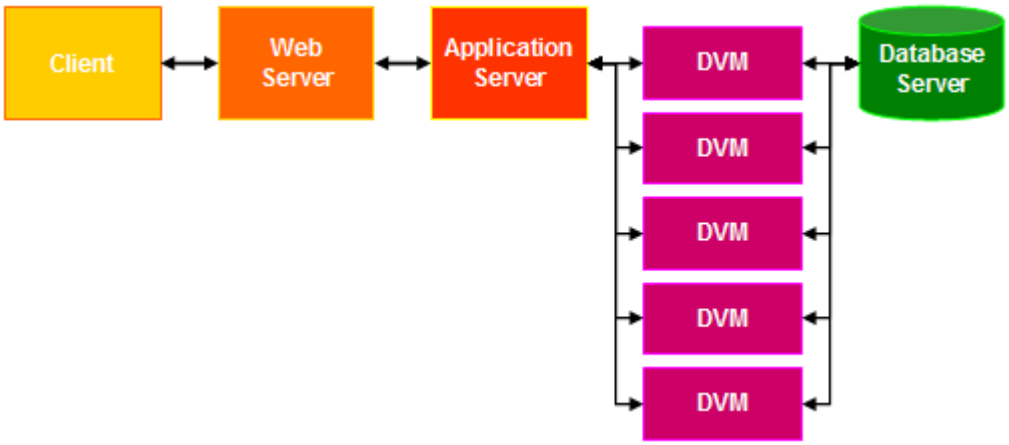
In addition to the POOL elements mentioned above, a Web Service application definition also includes a DVM_FREE timeout. The DVM_FREE timeout, specified in seconds, is used to shut down a DVM that has no request to process. After DVM_FREE seconds pass, if there are no requests to process, the DVMs are released to reach MIN_AVAILABLE. For this example, one DVM is released.



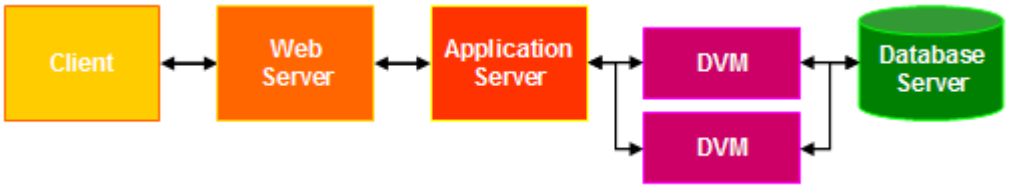
As a request come in, if there are no available DVMs free to process the request, then a DVM is launched to process the request. For example, if six requests come in, then six DVMs are started to process those six requests.



Once a request is completed, the DVM is immediately released to reach the MAX_AVAILABLE number. It does NOT wait for DVM_FREE to release the DVM. In our example, this reduces the total number of DVMs to five.



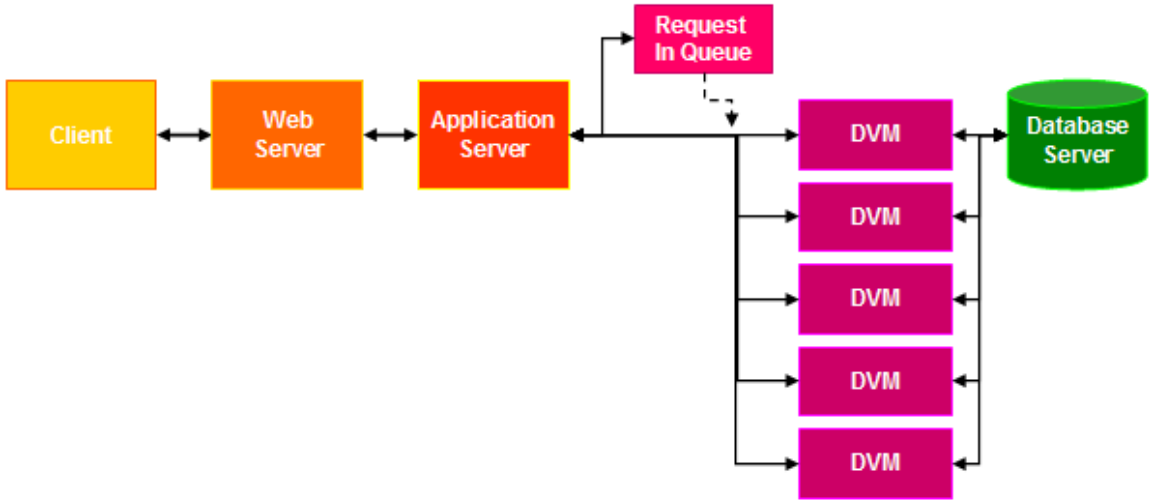
After DVM_FREE time, if there are no more requests to process, then the DVMs are released to reach MIN_AVAILABLE number.



The only way to limit the number of DVMs is to use the MaxLicenseConsumption attribute within the SERVICE_LIST element. This attribute limits the amount of DVMs that can be concurrently running across all applications referenced by the SERVICE_LIST element.

When this attribute is used, a DVM can be placed in the queue waiting for an available DVM. Once in the queue, request waits in the request queue until a DVM becomes available and is able to process that request OR the REQUEST_QUEUE timeout is reached. If the REQUEST_QUEUE timeout is reached, the Genero Application Server returns an error message and logs "REQUEST_QUEUE timeout expired".

For example, if MaxLicenseConsumption=5 and a new request comes along, then that request waits in the request queue until a DVM becomes available and is able to process that request OR the REQUEST_QUEUE timeout is reached.



GAS Startup and Command Options

You configure the GAS through a 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 about setting configuration parameters in the configuration file, refer to *GAS Configuration File Overview*.

Starting the GAS involves running the gasd command. This starts the gasd - the Genero Application Server daemon. When starting the GAS, you can specify options to provide additional information on how the application server is started, in addition to the settings in the configuration file, as well as providing information about the application server.

Summary of gasd command:

This tool (gasd) manages the Genero Application Server daemon and performs some configuration setting and checking.

Syntax:

```
gasd [options]
```

Options:

Option	Description
`-h`	This option displays help information.

`-p directory`
 	
`--as-directory directory`
	This option allows you to specify the Genero Application Server directory.

`--configuration-check`	This option validates the GAS configuration file and exits. Errors are displayed to error output.
`--configuration-explode`	This option explodes the GAS configuration file into separate files, one for each application, which are then stored in $FGLASDIR/tmp. Each file lists the entire configuration for an application, expanding the inherited components.

`-f configuration_file`
 	
`--configuration-file configuration_file`
	This option allows you to specify which configuration file to use when starting the Genero Application Server daemon (gasd). If not specified, the default configuration file ($FGLASDIR/etc/as.xcf) is used.

`--configuration-no-validation`	This option skips the XSD validation phase for the configuration file. The $FGLASDIR/etc/**cfas.xsd** file is the XML Schema Description file that documents the rules and constraints of the as.xcf syntax. You can use this file with an XML tool to

	validate the as.xcf file.
`--product-information`	This option displays product information.
`--development`	By default, template and snippet files are cached by the GAS daemon. Setting this option causes the GAS daemon to reload template and snippet files each time a new page is created. By setting this option, developers can see changes made to template and snippet files without having to restart the GAS daemon.
`-d` **Unix Only !** `--no-daemon` **Unix Only !**	This option starts the daemon as a foreground process. When this option is omitted, the daemon is started as a background process.
`-E` *name=value* `--resource-overwrite` *name=value*	This option overwrites the resource defined in the configuration file or creates a new one. Example: `gasd -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**.
`--resource-define` *name=value*	This option declares a resource. If the declared resource exists in the configuration file, the **final value** of the resource is the one set in the configuration file. The resources can be set to any resource defined in the configuration files or the ones defined by the -E or --resource-overwrite options.
`--service-install` **Windows Only !**	This option installs Genero Application Server as a service and exits.
`--service-uninstall` **Windows Only !**	This option uninstalls the Genero Application Server as a service and exits.

`-V`
`--version`
`--Version`
	This option displays version information.

Note:

- As of version 2.0, the "-l" option for controlling Genero Application Server logging has been removed. Logging options are now controlled by specifying LOG elements in the Genero Application Server configuration file. For more details, see *Logging - Configuration Reference*.

What does "address already in use" mean ?

The message "address already in use" means that an appliction server (gasd) has already been started on the same port. Check in the AS configuration file (default **$FGLASDIR/as.xcf**) to identify the port where the application server (gasd) 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.

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 either www.w3c.org or www.w3schools.com.

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.

DVM The Dynamic Virtual Machine or Runtime System that is installed on the Application Server and executes the application program.

GAS Genero Application Server. Defined by the computer system that houses the Dynamic Virtual Machine (DVM).

GDCAX Genero Desktop Client / Active X.

GJC Genero Java Client. A client technology that renders applications in a Java Graphical User Interface.

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.
User Agent	A User Agent is a client agent. It can be a browser, the Genero Desktop Client or the Genero Java 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.4js.com in your browser, this sends a request to the server whose domain name is 4js.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.

Installation

Topics

- System requirements
- Installation
- Directory and files created or touched
- Starting the GAS and Validating the Installation with the GDC or GDC/AX
- Starting the GAS and Validating the Installation with the GWC
- The 404/400 error (when using fglccgi.exe or fglcisapi.dll)

System Requirements

- Operating systems
- Virtual machine
- Web Server
- User Agent
- Database

Operating System

Genero Application Server is supported on a large brand of operating systems, such as Linux, IBM AIX, HP-UX, SUN Solaris and Microsoft Windows.

Each Genero Application Server package is identified with an operating system code (hpx1100, w32vc71). You must install the Genero Application server package corresponding to the operating system that you use.

For the detailed list of supported operating systems, please refer to the Four J's support web site.

Virtual Machine

DVM for Genero 2.00+ is required.

Web Server

Any web server compliant with CGI (Common Gateway Interface) version 1.1 is supported. For development platforms, we recommend Apache httpd. For more information, refer to http://httpd.apache.org.

User Agent

Any Genero Front End or a Web Services client.

For the Genero Web Client, the supported user agents include:

User Agent Provider	Version(s) Supported
Microsoft	Internet Explorer (IE) 6.x Internet Explorer (IE) 7.x
Mozilla	FireFox 2.x
Opera Software	Opera 9.x
Apple, Inc.	Safari 3.x

Database

Any database accessible from the DVM or from the ODI. Refer to the Genero Business Development Language Manual for details.

Installing the Genero Application Server

The software is provided as an auto-extractible installation program (i.e. product files and installation program are provided in the same file). The name of the package includes the operating system type and version. Ensure the package name corresponds to your operating system before starting the installation program.

You should also know what type of installation to choose when prompted.

Do I need to be superuser to install Genero Application Server ?

You do not need to be superuser to install Genero Application Server. However, some parts of the Genero Application Server need to be installed with special rights. For example, installation of the connector assumes that you have the rights to install the product in the web server directories.

Select the Installation Type

The installation type you select depends on whether you are installing the software on the application server host, the web server host, or a host that will contain both the application server and web server.

- **Type 1**: Install the Application Server - choose for the application server host.
- **Type 2**: Install the Connector - choose for the web server host.
- **Type 3**: Install both the Application Server and Connector.

The installation procedure differs between UNIX and Windows platforms:

- Installing on UNIX
- Installing on Windows

Installing on UNIX platforms

The installation program provides options that allow you to specify configuration options from the command line . You can display the installation program options using the **-h** option:

```
$ /bin/sh fjs-gas-version-hpx1100.sh -h
```

The installation program identifies the operating system and checks that all the system requirements are met before starting to copy the product files to your disk.

To perform the installation, run the auto-extractible shell script with the -i option:

```
$ /bin/sh fjs-gas-version-hpx1100.sh -i
```

Your application server and web server may reside on separate machines. As such, you are presented with three installation choices:

```
1 --- Application Server (Application server - gasd)
2 --- Web Server         (CGI Connector)
3 --- Full installation  (Application server and CGI Connector)
```

Installation **type 1** installs the GAS engine on your application server. Only this part is needed for development purpose.

Installation **type 2** installs the GAS Connector on your web server.

Installation **type 3** assumes that your application server and web server are the same machine.

After you select an installation type, the installation program copies the product files to the relevant directories on disk.

Once the files are copied to disk, follow the instructions displayed.

Note: You can install the package as root using -r or --root option.

Installing on Microsoft Windows platforms

On Microsoft Windows, GAS is provided as a standard Windows setup program. Distribution files and installation program are provided in the same file.

```
fjs-gas-version-windows.exe
```

When you execute the setup program, a wizard guides you through the installation process. At one point, you will be asked to select the type of installation: Application Server, Web Server, Full Installation, or Custom.

- If your Application Server and Web Server sit on separate hosts, you would select Application Server or Web Server according to the host machine's role. For development, you may bypass the Web Server and connect directly to the Application Server for development and testing; in this scenario, select Application Server.
- If your Application Server and Web Server reside on the same host, select Full Installation.

On Microsoft Windows, GAS can be installed as a Windows service. If installed as a Windows service, the GAS daemon can be started automatically at the server startup.

With Microsoft Internet Information Services, the installed files may not have the right permissions. You will need to update these files permissions to fit IIS permissions.

Directory and files created or touched

The following table lists those directories and files created by or touched during the installation process.

Directory	File	Description
<*webserver*>		Web Server installation directory.
<*webserver*>/<*script*>/	fglccgi	GAS connector.
	fglcgienv	Tool to check Web Server environment.
	connector.xcf	Connector configuration file.
$FGLASDIR		GAS installation directory.
	envas	Script for setting environment variables.
$FGLASDIR/app		GWC applications external configuration files.
$FGLASDIR/bin	gasd	GAS daemon.
$FGLASDIR/etc	as.xcf	GAS configuration file.
$FGLASDIR/log		By default, GAS log files are written to this directory.

$FGLASDIR/tmp		Default file transfer directory.
$FGLASDIR/tpl/	generodefault.html	Template directory containing the default template file for the built-in rendering engine. **Note:** The generodefault.html template is not used for the snippet-based rendering engine.
$FGLASDIR/tpl/set1 $FGLASDIR/tpl/set2 $FGLASDIR/tpl/set3	main.xhtml	Snippet-based rendering engine template subdirectories. Each sub-directory contains a template file (main.xhtml) and a set of .xhtml snippets that define how the snippet-based rendering engine displays the objects in the UI.
$FGLASDIR/web	demos.html	Root directory for direct communication to the application server. Demonstrations listing.
$FGLASDIR/web/fjs	gwccore.js gwccomponents.js gwccomponents.css	JavaScript handling the application behavior. Default cascading style sheet for Set1.
$FGLASDIR/web/fjs/asapi	application.js	JavaScript handling communication with Genero Web Client.
	wrappers.js	JavaScript handling widgets behavior.
$FGLASDIR/web/fjs/uaapi	webBrowser.js	JavaScript handling user agents specifics.
$FGLASDIR/web/fjs/defaultTheme	genero.css	Default cascading style sheet for legacy engine.
	genero.js	JavaScript handling the application design.

<*script*> is the script directory of your web server (example "cgi-bin" for Apache and "scripts" for Internet Information Services).

See also Genero Web Client Application Directory Structure.

Starting the GAS and Validating the Installation with the GDC/AX

Application Server

Note: If you installed the GAS daemon as a service on Windows and have started the service, skip to step 3.

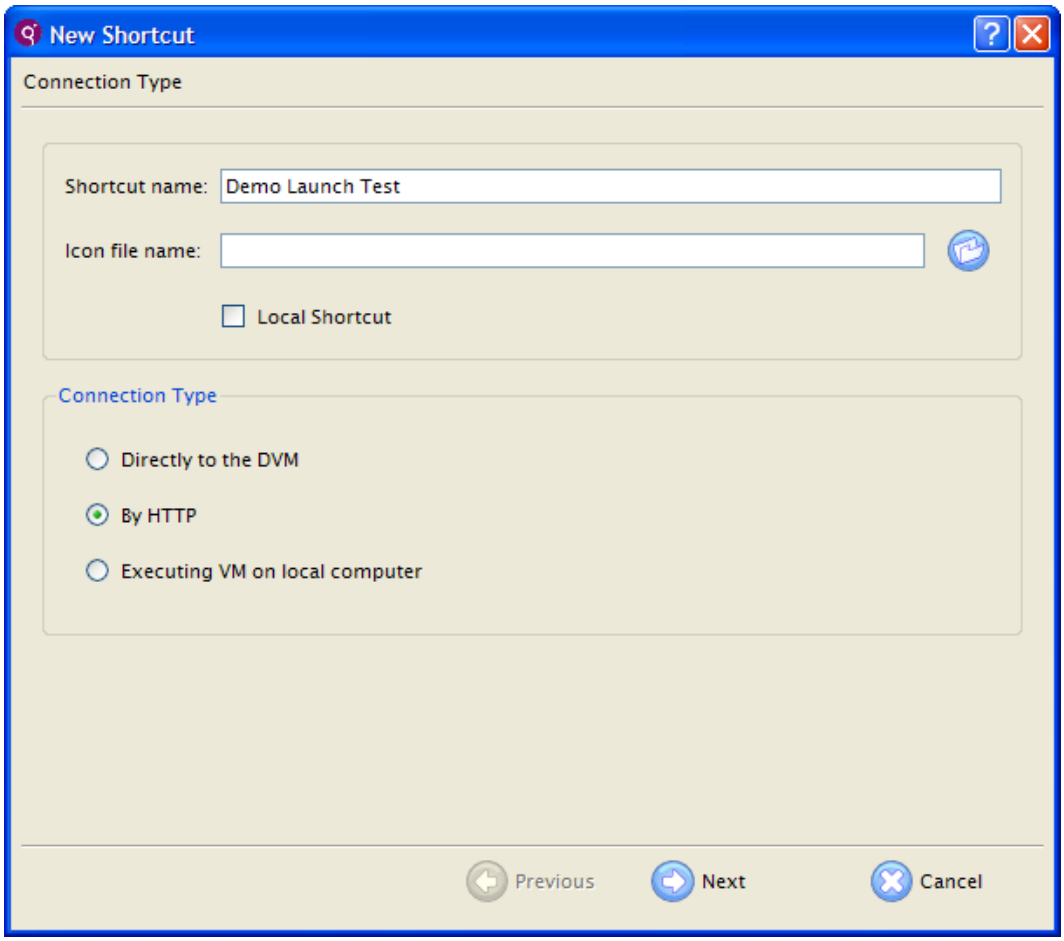
1. Set the GAS environment using the script $FGLASDIR/envas
2. Launch the GAS daemon **gasd**
3. Check the connection to the GAS with a direct connection by launching a demo application.

   For the GDCAX, use Internet Explorer to access to the demo application (`http://myApplicationServer:6394/wa/r/gdc-demo`), replacing `myApplicationServer` with the name of the server hosting the GAS. The first time you access to the demo application, the ActiveX will install by itself. For more information, refer to the *Genero Desktop Client Manual*.

   For the GDC, you need to create a shortcut pointing to the demo application (`http://myApplicationServer:6394/wa/r/gdc-demo`)

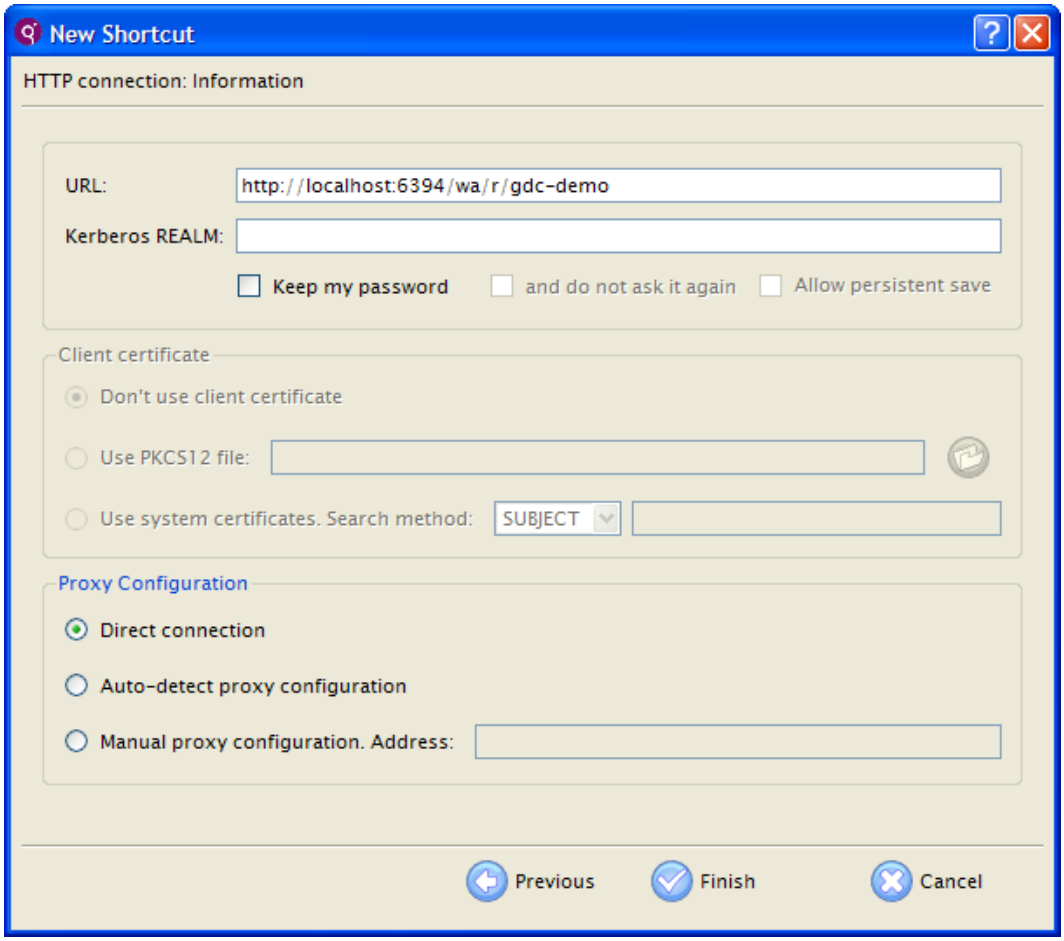
   To create the shortcut, you must start the GDC in administrative mode using the `--admin` or `-a` option. Refer to the *Genero Desktop Client Manual* for more information on creating shortcuts.

   To create the shortcut required for this test, once you have started the New Shortcut wizard, select the **By HTTP** option:



In the HTTP connection information page, you provide the application URL. On most systems, you can replace the "myApplicationServer" with "localhost" for this test.

```
http://localhost:6394/wa/r/gdc-demo
```



Tips:

On Windows platforms, if an application does not start you can debug the problem by manually launching the program. For example, use the command: gasd -E res.dvm.wa="cmd /K start cmd". The GAS opens a DOS command window when it accesses the application. In the DOS window, the environment for the application is set; you can now manually run the program and check step-by-step what went wrong.

Web Server

1. Perform the Application Server checkup first
2. Ensure that your webserver is correctly configured by accessing a static page like index.html
3. Check your connector.xcf file, figure out if you access the right GAS
4. Launch a demonstration program through the web server

```
http://myWebServer/cgi-bin/fglccgi/wa/r/myApp
```

Starting the GAS and Validating the Installation with the GWC

Important! After you upgrade your GAS, you must refresh the css and js downloaded in the browser cache by clearing the browser cache. For many browsers, you can accomplish this by pressing CTRL + F5.

Application Server

Note: If you installed the GAS daemon as a service on Windows and have started the service, skip to step 3.

1. Set the Genero Web Client environment using the script `$FGLASDIR/envas`.
2. Launch the GAS daemon **gasd** with the `gasd` command.
   To have the gasd reload template and snippet files each time a new page is created, start the GAS daemon with the `--development` flag. This is useful if you are making changes to either the template or snippet files, and wish to see the results without having to restart the GAS daemon. For more information on starting the Genero Application Server daemon and the various command options, please refer to the chapter Startup and Command Options .
3. Check the connection to the application server using a URI providing a direct connection. A variety of demonstration applications are provided with the installation of Genero Web Client:

   ```
   http://<myApplicationServer>:6394/wa/r/Edit
   http://<myApplicationServer>:6394/demos.html
   ```

   The latter URI displays a list of the available demonstration programs.The Demos application is provided with the installation files, and is pre-configured and ready to run.

Web Server

1. Check the installation of your application server (as stated in the previous paragraph).
2. Ensure that your web server is correctly configured by accessing a static page (such as index.html)
3. Launch a demonstration program using a URI inclusive of the Web server connector.

   ```
   http://<myWebServer>/cgi-bin/fglccgi/wa/r/myApp
   ```

4. **Note:** On Windows platforms, when connecting via a Web server, you must include the extension when calling fglccgi.exe, as shown in the following URL:

   ```
   http://<web_server>/cgi-bin/fglccgi.exe/demos.html
   ```

The 404/400 error (when I use fglccgi.exe or fglcisapi.dll)

With IIS 6.x, running cgi or isapi is disabled by default. To use fglccgi.exe or fglcgisapi, you need to enable their execution.

1. In the IIS manager console, go to the "Web Service Extension".
2. Select "CGI Extensions".
3. Click on "Allow".
4. Repeat this process with "ISAPI Extensions".

Quick Start - Adding New Applications

In order to have the GAS deliver an application, you need to provide the application's configuration details to the GAS.

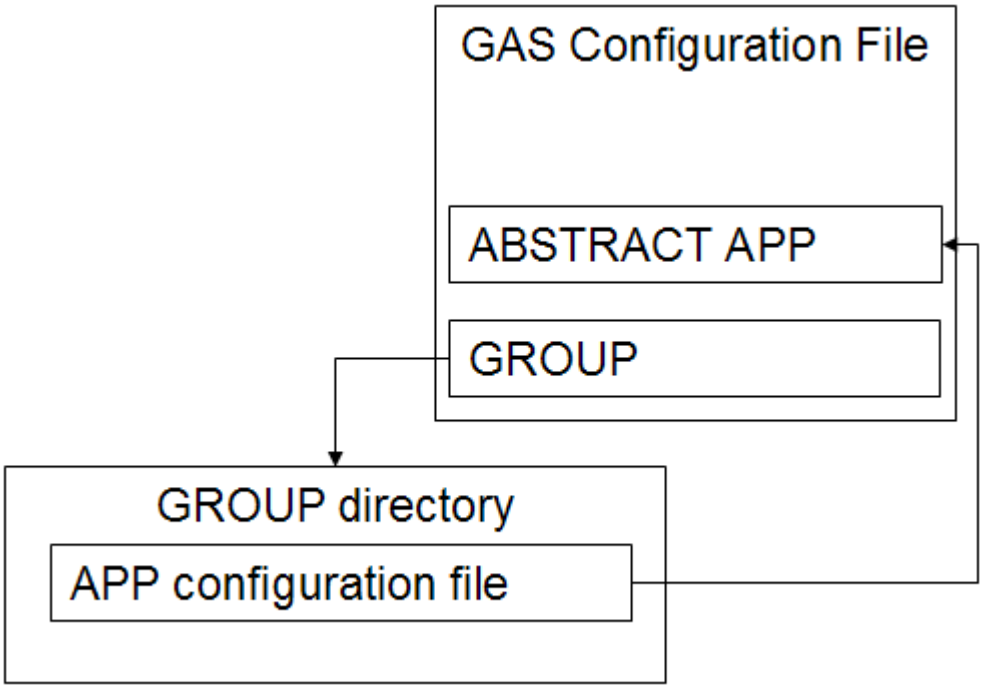
Topics

- Understanding application configuration
- Creating an application group
- Creating an application configuration file
- Adding an application directly to the GAS configuration file
- Configuring database environment variables for an application
- Using a script to set the application’s environment
- Specifying an images directory for an application

Understanding Application Configuration

Before you start, you should have successfully installed the Genero Application Server and validated your installation. See Installation for more details.

Once you've verified that you can display the demo application with the desired front-end client, you then add your own custom applications to the configuration. While there are several ways to specify application configuration, this topic will outline the best practices for adding your application configuration details to the GAS configuration.



To add applications:

Step 1: Identify (or create) an abstract application

An abstract application is an application definition that defines the various configuration settings (known as components) that the Genero Application Server needs to run an application. In other words, the purpose of an abstract application is to provide default components / configuration details that can be inherited by other applications. These components can include defining resources, environment variables, timeout settings, themes, and more. An abstract application cannot be directly executed.

By default, abstract applications have been defined for you in the GAS configuration file (default as.xcf) for each of the various front-end clients. For example, an abstract application named "defaultgwc" has been defined for you for use with the Genero Web Client. You can use this default abstract application, or you can create a new abstract application.

Abstract applications must be defined in the GAS configuration file. After updating the GAS configuration file, the GAS must be restarted.

An abstract application is used to define a configuration set shared by multiple applications. Internal and external applications will inherit an abstract application's configuration via the `Parent` attribute of their `APPLICATION` tag.

An abstract application is defined in the same way an internal or external application is defined, except it has an extra attribute set in its `APPLICATION` tag, where `Abstract="TRUE"`.

When you define the configuration for a specific application, you can override the settings you inherit from the abstract application.

Step 2: Create an application group

A GROUP defines a directory that contains application-specific configuration files. Groups must be defined in the GAS configuration file.

By creating a group, you can add new application configuration files in the group directory, and the applications are instantly available to the GAS without having to restart the application server.

Instructions for creating an application group are provided below.

Step 3: Create the external application configuration file

To add a configuration file for an application, you create a new file within the group directory. The name of the file should match the name of the application, and have an XCF suffix. For example, if the application name was "app1", then you would create a

configuration file named "app1.xcf". In this file, you specify a parent application - the abstract application from which this application will inherit its default settings.

These files are known as external application configuration files. You could also add the same information to the GAS configuration file; however you have to restart the GAS whenever you modified the GAS configuration file. By using groups and external application configuration files, you've provided flexibility in growing the number of applications delivered by the GAS without having to worry about the impact of restarting the GAS.

NOTE: When an application is defined within the GAS configuration file, it is known as an *internal application.* When an application is defined in a separate application configuration file, it is referred to as an *external application*, and its configuration file is known as an *external application configuration file*.

Instructions on creating an external application configuration file are provided below.

Creating an Application Group

The GROUP element defines an alias for a directory containing one or more external application configuration files. The alias is then used in the application URL, letting the GAS know in which directory to find the external application configuration file.

You can use application groups to organize your applications into logical groups or a hierarchy.

Consider the following application URL:

```
http://<server>/cgi-bin/fglccgi/wa/r/accounting/app1
```

In this URL, both a group ("accounting") and an application name ("app1") are specified. The Genero Application Server, 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>
```

Within this directory, the Genero Application Server would expect to file a file whose name matches the name of the application with an .xcf suffix. In this example, the Genero Application Server would be looking for a file named "app1.xcf".

Syntax

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

Notes

1. *groupId* is the alias.

2. *path* is the physical path to the directory.

Examples

```
<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 a RESOURCE **$(res.path.demo.app)**. By wisely using RESOURCE elements, you can set your configuration where a change to the directory structure only requires a change to a single RESOURCE element in the configuration file.

To access an application defined externally and contained within this group, you would enter an application URL that includes the alias in its path:

```
http://server/cgi-bin/fglccgi/wa/r/demo/CardStep1
```

Based on this URL, the Genero Application Server expects to find the configuration file **CardStep1.xcf** within the directory specified for the **demo** group.

The default group

The Genero Application Server configuration file provides a default group, defined using the name **_default**. When an external application configuration file is added to this group, the application URL can omit using a group name and simply reference the application.

For example, the GAS installs with demo applications for use with the GWC, to include an application that demonstrates Edit fields. You access the application by entering:

```
http://server/cgi-bin/fglccgi/wa/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 **$(res.path.app)** resolves to **$FGLASDIR/app**, where you find a file named **Edit.xcf**. This is the Edit application's external application configuration file.

Creating an Application Configuration File

The `APPLICATION` element defines an application environment. Within this element, you can define local resources, change the execution environment, the timeout settings and

the picture and output settings. You can refer to previously defined components by using the tag attribute `Using`.

Syntax

```
<APPLICATION Id="appId" [ Abstract="{ TRUE | FALSE }" ] [
Parent="pAppId" ] >
   [ resource ] [...]
   [ <EXECUTION [ Using=" exCompId " ] > execution </EXECUTION> ]
   [ <TIMEOUT [ Using=" timeCompId " ] > timeout </TIMEOUT> ]
   [ <PICTURE [ Using=" picCompId " ] > picture </PICTURE> ]
   [ <OUTPUT Rule="UseGWC">
       <MAP Id="DUA_GWC" Allowed=" { TRUE | FALSE } " >
            [ <THEME [ Using=" themeCompId " ] > theme </THEME> ]
        </MAP>
     </OUTPUT> ]
</APPLICATION>
```

Notes

1. *appId* is the application identifier
2. *pAppId* is the parent application identifier
3. An abstract application is used to share common configuration between multiple child applications. An abstract application can't be instantiated.
4. *resource* is a local `RESOURCE` definition
5. *exCompId*, *timeCompId*, *picCompId* and *themeCompId* are components identifiers
6. the content of *execution*, *timeout*, *picture* and *theme* is the same as the content of their respective components

Example 1

The simplest external application configuration file only needs to specify a parent application and the path to the compiled application files. In this example, the application inherits the configuration settings of the parent application. This XML would be saved in a file named *appname*.xcf, where *appname* is the name of the application.

The following XML defines the Edit application in an external application configuration file *Edit.xcf*.

```
01 <APPLICATION Parent="defaultgwc">
02   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.x
sd">
04  <EXECUTION>
05    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
06  </EXECUTION>
07 </APPLICATION>
```

Notes

1. The name of the application is the name of the .xcf file. The `Id` attribute of `<APPLICATION>` tag is omitted for external applications; even if included, its value is not read. Instead, the GAS uses the name of the configuration file to match to the value of the `Id` attribute.
   In the example above, the `Id` of the application is `Edit`.
2. The external application configuration file is re-read at each application launch. There is no need to restart GAS after modifying an external configuration file.
3. The directory where the GAS searches for the external application configuration file is defined in **as.xcf** by the tag `<GROUP Id="_default">`*directory*`</GROUP>`. The default after installation is `$FGLASDIR/app`.
4. The Parent application is defined as "defaultgwc". This means that the application will inherit the configuration settings defined for the default GWC (defaultgwc) application, which is an abstract application defined in the GAS configuration file.
5. The path to the application executables is defined by the PATH component.
6. No MODULE element is needed when the external configuration file shares the same name as the application. When there is no defining `<MODULE>` tag in the application configuration, the module taken by default is the name of the application.

Limitations

1. An external application cannot be an `Abstract` application.
2. An external application can only inherit from an internal application.

Example 2

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 of a hypothetical external application configuration file, tutorialStep1.xcf, which would be found in the demo directory.

```
01 <APPLICATION Parent="demo-tut-abstract">
02   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.x
sd">
04   <!-- Define a resource to the template HTML file -->
05   <RESOURCE Id="res.template.tutorial"
Source="INTERNAL">$(res.path.demo.dem-
tut)/web/tutorial/tutorialStep1.html</RESOURCE>
06   <EXECUTION>
07     <PATH>$(res.path.demo.dem-tut)/src</PATH>
08     <MODULE>tutStep1.42r</MODULE>
09   </EXECUTION>
10   <!-- Override default rendering template -->
11   <OUTPUT>
12     <MAP Id="DUA_AJAX">
13       <THEME>
14         <TEMPLATE Id="_default">$(res.template.tutorial)</TEMPLATE>
15       </THEME>
16     </MAP>
```

```
17   </OUTPUT>
18 </APPLICATION>
```

Notes

1. Line 01 specifies the parent application as "demo-tut-abstract". Unless a value is overwritten locally in the application configuration, the child application inherits the configuration elements defined by the parent application.
2. Line 05 defines a local RESOURCE. This resource maps to a template file.
3. Lines 07-08 provide the path and file name of the program executable. It is common to exclude the MODULE element when the executable name matches the application name as provided in the URL. In this example, if the external application configuration file had been named "tutStep1.xcf", then the MODULE element specifying the program executable as "tutStep1.42r" could have been excluded.
4. Line 12 defines the OutputMap as "DUA_GWC".
5. Line 14 overrides the _default template with the template defined by the $(res.template.tutorial) resource in Line 04.

For additional information on the APPLICATION element and its child elements, see Application List Reference.

To summarize, applications are typically defined using external application configuration files, which are recognized as soon as they are added to a GROUP directory.

Adding Applications to the GAS Configuration File

While it is recommended that you add new applications using application groups and external application configuration files, you have the option of adding your application's configuration details directly in the GAS configuration file. Modifications to this file are recognized only after the application server is restarted.

Both GROUP and APPLICATION configuration takes place within the APPLICATION_LIST element in the Genero Application Server configuration file.

```
<APPLICATION_LIST>
  [ group | application ] [...]
</APPLICATION_LIST>
```

To provide the application configuration within the Genero Application Server configuration file, you add an APPLICATION element and use the Id attribute to specify the application name. It is this application name that is then used within the application URL.

Consider the following URL:

```
http://<server>/cgi-bin/fglccgi/wa/r/app1
```

If the application is defined internally, you would expect to find an APPLICATION element with an Id that matches the application name provided in the URL:

```
<APPLICATION Id="app1" Abstract="FALSE" Parent="defaultgwc">
  ...
</APPLICATION>
```

The elements you can define between the APPLICATION tags is the same as for those applications defined externally. For details, see Creating an Application Configuration File.

Example in as.xcf defining 'gwc-demo' application

```
01 <?xml version="1.0" encoding="ISO-8859-6"?>
02 <?fjsApplicationServerConfiguration Version="1.30"?>
03 <CONFIGURATION>
...
181 <APPLICATION_LIST>
...
222   <!--Sample application for GWC-->
223   <APPLICATION Id="gwc-demo" Parent="defaultgwc">
224     <EXECUTION>
225       <PATH>$(res.path.fgldir.demo)</PATH>
226       <MODULE>demo.42r</MODULE>
227     </EXECUTION>
228   </APPLICATION>
...
234 <APPLICATION_LIST>
...
235 <CONFIGURATION>
```

Notes

The above example shows the minimum information required to define an application in the application server configuration file.

1. The application is defined within the `APPLICATION` tags. The attributes shown in the example are only a few of the attributes allowed within the `APPLICATION` tags. For a more complete list of application tags, see the Application List Reference.
2. The `Id` tag specifies the name of the application. It is this name that is referenced in the URI.
3. The `Parent` tag identifies the parent application. This may be an executable or abstract application. This application inherits the attribute values set for the parent application. For those attributes that are assigned a value both in the parent application definition and within this application's definition, the value set for the application overrides the value set for the parent.
4. The `EXECUTION` section contains additional tags providing information needed to execute the correct application.

5. The `PATH` attribute defines the directory containing the module to be executed. It is typical to list a resource that maps to the directory than the actual directory.
6. The `MODULE` attribute identifies the module to execute. Note that the extension is used.

Warning: After making changes to the internal application configuration file, the application server (gasd) must be restarted for the changes to take effect.

Configuring the Database Environment

You may need to set database-related environment variables for your application to work correctly. Environment variables are set within the EXECUTION element of an application's configuration.

Syntax

```
<ENVIRONMENT_VARIABLE Id="env_var">env_value</ENVIRONMENT_VARIABLE>
```

Notes

1. *env_var* is the environment variable name.
2. *env_value* is the value used to set the variable name.

Example (using Informix)

```
<APPLICATION Id="myapp" Parent="defaultgwc">
  <EXECUTION>
    <ENVIRONMENT_VARIABLE
Id="INFORMIXDIR">ifx_path</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE
Id="INFORMIXSERVER">ifx_server</ENVIRONMENT_VARIABLE>
    <ENVIRONMENT_VARIABLE
Id="LD_LIBRARY_PATH">library_path</ENVIRONMENT_VARIABLE>
    <PATH>/home/myapp/bin</PATH>
    <MODULE>app.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

Notes

1. Replace *ifx_path* with the value of the INFORMIXDIR environment variable.
2. Replace *ifx_server* with the value of the INFORMIXSERVER environment variable.
3. Replace *library_path* with the value of the LD_LIBRARY_PATH environment variable.

Using a Script to set the Environment

Rather than specifying the environment variables with ENVIRONMENT_VARIABLE elements, you can provide an application configuration that calls a script, where the script sets the execution environment.

Example

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.x
sd">
  <EXECUTION>
    <PATH>/home/f4gl/gep/configfiles/officestoredemo</PATH>
    <DVM>/bin/sh</DVM>
    <MODULE>gdc-kiosk.sh</MODULE>
  </EXECUTION>
</APPLICATION>
```

Notes:

1. PATH is where the script is stored.
2. DVM is the command to execute the script.
3. MODULE is the script file.

Specifying an Application's Images Directory

By default, the GAS looks for images in $FGLASDIR/pic. You can add your images to this directory or you can specify your own image directory. To define your own directory of images, you first define an alias for the directory in the Genero Application Server configuration file, and then reference the alias in the application configuration.

Defining an Alias

An alias is like any Web Server alias. This maps a URL path to the server directory. For example:

```
<INTERFACE_TO_CONNECTOR>
  ...
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  <ALIAS Id="/images">/home/app/images</ALIAS>
  ...
<INTERFACE_TO_CONNECTOR>
```

In this example, the alias /images is mapped to the directory /home/app/images.

After you have added the alias to the GAS configuration file, you must restart the GAS daemon. Remember, you must restart the Genero Application Server daemon (gasd) whenever you make any changes to the GAS configuration file, as this file is read at start-up.

Once the GAS restarts, you can access an image with the URL:

```
http://<app_server>:6394/images/img.png
```

(This example assumes the img.png file is in directory /home/app/images.)

Referencing the Alias in the application configuration

Having defined the alias, you reference the alias within the application configuration.

Syntax

```
<PICTURE>
  <PATH>$(connector.uri)alias</PATH>
</PICTURE>
```

Notes

1. *alias* is the alias previously defined in the INTERFACE_TO_CONNECTOR element.
2. $(connector.uri) allows for pictures stored on the application server to be available to the Web server.

Example

```
<APPLICATION Id="myapp" Parent="defaultgwc">
  <EXECUTION>
    <PATH>/home/myapp/bin</PATH>
    <MODULE>app.42r</MODULE>
  </EXECUTION>
  <PICTURE>
    <PATH>$(connector.uri)/images</PATH>
  </PICTURE>
</APPLICATION>
```

For more details on application configuration, see also:

- Application List Reference
- Adding a GDCAx / GJC application
- Adding a GWC Application
- Adding a GWS application

Configuration of the Genero Application Server

Configuration involves two configuration files: the Genero Application Server configuration file (`as.xcf`) and the GAS Connector configuration file (`connector.xcf`).

Topics

- Configuring the GAS
- Configuring the GAS Connector
- Deploying applications
- Creating an application deployment strategy

Configuring the GAS

To request an application, you can access the Genero Application Server directly or you can access the server via a Web server. See Connection Types in the Architecture section for more information.

To achieve a desired level of performance, it is possible to host multiple application servers and multiple Web servers.

For each GAS added to the solution, an administrator must create an application server configuration file specifically to support that application server. An application server configuration file specifies the resources (variables), timeout parameters, environment variables, port settings, and application-specific details for an application server. For a full explanation of the application server configuration file, you can start with the GAS Configuration File Overview.

TheGAS installs with a default configuration file, **as.xcf**. To start an application server using this default configuration file, run:

- **gasd** (to start as a daemon in the background)
- **gasd -d** (to start the process in the foreground)

To specify a different application server configuration file, use the "`-f`" option to specify the file by name:

- **gasd -f *custom_as.xcf* -d**

where *custom_as.xcf* is the application configuration file.

To create an custom application server configuration file, create a copy of the default application server configuration file **as.xcf** , rename the file, and modify the file as needed.

Configuring multiple application servers

When configuring multiple application servers on the same host, take care to assign mutually exclusive ports between the application servers. In the application server configuration file, you specify two types of port settings: `INTERFACE_TO_CONNECTOR` and `INTERFACE_TO_DVM`.

The `INTERFACE_TO_CONNECTOR` section specifies the port number where the application server listens for requests. If you plan to have multiple application servers (gasd) on the same host, ensure the application servers (gasd) daemons are listening on different ports. To accomplish this, change the port offset for each application server you plan to run. For example, one daemon can be configured to listen on port 6394 (base port of 6300 + port offset of 94), while another can be configured to start on port 6395 (base port of 6300 + port offset of 95).

If you do not specify unique ports for each application server, you will receive an error when starting the second or subsequent application server, stating that the application server could not start or that the specified port is already in use.

Warning!: Any change in the port set in the INTERFACE_TO_CONNECTOR section of the application server configuration file requires a similar change in the Connector configuration file.

The `INTERFACE_TO_DVM` section specifies the range of port numbers on which the application server can start a DVM to service an application request. When setting the range, you specify three things:

- The DVM base port
- The range interval
- The list of excluded ports

The combination of these settings determine the range of port values available for the application server to start DVMs to service requests for applications. For example, if you set the DVM base port as 6420, the port range interval to 10000, and list 10 excluded ports, the range becomes 6420 through (6420 + 10000 + 10), or 6420 through 16430.

When several application servers run on the same host, each application server should specify a mutually exclusive range of ports. As an administrator, ensure that there is no overlapping of ports in the ranges specified for the various application servers. Continuing with our previous example, when adding a second application server, the DVM base port would be set to 16431.

If the ranges do overlap, the application servers continue to function, looking for the next available DVM within its port range to service new requests. Failure to prevent overlapping port ranges simply result in an application server being able to only run a subset of the expected number of applications, as DVMs will not be able to start once all ports within the specified range are in use.

Configuring the GAS Connector (for a Web server)

For each Web server you introduce into your solution, you must install and configure the GAS Connector. For information on configuring the GAS Connector, see Configuring the GAS Connector.

When configuring a Connector, you should ensure that each server reference reaches the correct application server. In other words, verify that each base port and port offset set in the GAS Connector configuration file (`connector.xcf`) file match a base port and port offset set in an application server configuration file (`as.xcf`).

Deploying Applications

To deploy an application, it must be defined for the Genero Application Server.

For information about adding applications to your Genero Application Server configuration, see:

- Quick Start - Adding New Applications
- Adding GDCAX / GJC Applications
- Adding GWC Applications
- Adding GWS Applications

Creating an application deployment strategy

When an application is requested, the application server starts a DVM to handle the request. Having all applications served by a single application server may not perform as desired. To provide scalability, the GAS Connector can direct specific applications to specific application servers and/or spread the requests for one application across several application servers.

When a user enters the URL for an application that goes through a Web server, the Connector references its configuration file in order to identify the application server to receive the request. For information on modifying the Connector configuration file, see Configuring the GAS Connector.

Once the application server has been identified, the request is passed to the application server. The application server identifies which application to display by matching the application asked for in the URI against the `Id` listed in either the GAS configuration file

(`as.xcf`) or, if not present, by matching the application name against the file names used for external application configuration files.

Applications defined as an external application have the benefits of enabling organization by groups (allowing for a taxonomy of applications to be constructed), for adding/removing applications without having to restart the application server, and reducing risk of overwriting application configuration settings during upgrades of the GAS.

Automatic Discovery of User Agent (adua.xrd)

adua.xrd is the configuration file used by the Application Server to determine which user agent is used. This file is stored in $FGLASDIR/as/etc.

Tip: Under most circumstances, modification of this file is not necessary.

Topics

Output Maps Overview

This section will document Maps > Resources > Templates and template snippets.

How an Output Map is chosen

The Output Map used by an application is defined by the `auda.xrd` file, located in the `$FGLASDIR/etc` directory.

The Genero Application Server first identifies the value of the RULE element for the application. The RULE element is defined in the application configuration defined for and interpreted by the Genero Application Server.

For example, if the value of the RULE element is UseGWC, then drop into that element. Once inside that element, the type of device that is being used to display the application in the browser determines which Output Map is used to render the application.

```
<?xml version="1.0" encoding="UTF-8"?>
<?fjsRuleConfiguration Version="2.11"?>
<RULE_LIST
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/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="UseGJC">
  <TABLE Id="1" Key="User-Agent">
   <ROW>
     <ACTION Type="RESULT">DUA_GJC</ACTION>
   </ROW>
  </TABLE>
</RULE>
<RULE Id="UseGWC">
  <TABLE Id="1" Key="User-Agent">
    <ROW>
      <IN>Symbol-WC</IN>
      <ACTION Type="RESULT">DUA_Symbol-WC</ACTION>
    </ROW>
    <ROW>
      <IN>Symbian OS</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>EPOC</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>iPhone</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>Windows CE</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>PalmOS</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>PalmSource</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>Opera Mini</IN>
      <ACTION Type="RESULT">DUA_PDA</ACTION>
    </ROW>
    <ROW>
      <IN>MSIE</IN>
      <ACTION Type="RESULT">DUA_AJAX_HTML</ACTION>
    </ROW>
```

```
      <ROW>
        <ACTION Type="RESULT">DUA_AJAX</ACTION>
      </ROW>
    </TABLE>
  </RULE>
</RULE_LIST>
```

As shown in the sample `auda.xrd` file listed above, for handheld devices such as EPOC, iPhone, Windows CE, and PalmOS, the DUA_PDA Output Map is chosen. This Output Map consists of snippets that are designed to deliver an application on a handheld device. For other devices (such as desktop computers and standard laptops), the DUA_AJAX Output Map is selected.

Note: The built-in rendering engine remains for backwards compatibility and legacy implementations, however to use the built-in rendering engine you must either modify the auda.xrd file to specify "DUA_GWC" as the Output Map to use or explicitly reference the DUA_GWC Output Map in the application's URL. Both topics are covered below.

Modifying the auda.xrd file to specify custom Output Maps

If you have created a custom Output Map, you simply modify the Output Maps specified within the auda.xrd file to reference your new Output Map.

For example, imagine you have created an Output Map named DUA_AJAX1.

In the auda.xrd file installed by default, you would identify those situations where that application should use your custom Output Map. The decision is still based on the device on which the browser will display the application.

```
...
<ROW>
  <ACTION Type="RESULT">DUA_AJAX1</ACTION>
</ROW>
...
```

Specifying a specific Output Map in the application URI

You can specify an exact Output Map to use by including the Output Map declaration within the URL of the application.

For example, to specifically use the DUA_PDA Output Map when calling the Edit demo application, you would use the following URI:

```
http://localhost:6394/wa/r/gwc-demo?OutputMap=DUA_PDA
```

You can force an application to use a specific Output Map by providing the Output Map as an argument in the application URL.

Example 1

```
http://localhost:6394/wa/r/gwc-demo?OutputMap=DUA_GWC
```

In this example, the demo application is rendered using the DUA_GWC Output Map, which renders the application using the built-in rendering engine and its default theme.

Example 2

```
http://localhost:6394/wa/r/gwc-demo?OutputMap=DUA_PDA
```

In this example, the demo application is rendered using the DUA_PDA Output Map, which renders the application using the snippet-based rendering engine and the PDA theme.

AUDA Syntax Diagrams

- 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:

```
01 <RULE_LIST>
02   <RULE Id="UseGDC">
```

```
03      <TABLE Id="1" Key="User-Agent">
04        <ROW>
05          <IN>MSIE</IN>
06          <ACTION Type="RESULT">DUA_GDC</ACTION>
07        </ROW>
08        <ROW>
09          <IN>GDC</IN>
10          <ACTION Type="RESULT">DUA_GDC</ACTION>
11        </ROW>
12      </TABLE>
13    </RULE>
14    <RULE Id="UseGJC">
15      <TABLE Id="1" Key="User-Agent">
16        <ROW>
17          <ACTION Type="RESULT">DUA_GJC</ACTION>
18        </ROW>
19      </TABLE>
20    </RULE>
21 </RULE_LIST>
```

Note: A usage example is provided at the bottom of this section.

RULE_LIST

The RULE_LIST element is the main element of an XRD (XML Rule Definition) used by the Genero Application Server and 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 Genero Application Server configuration file. Valid values for the Id attribute include:

- UseGDC
- UseGWC
- UseGJC

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 which what is gioing to be analyzed. Currently, only two values are supported: HTTP_ACCEPT and HTTP_USER_AGENT. Those values are transmitted by the connector to the Genero Application Server.

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 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 sub-string that must be in the search string.
2. Zero or more OUT elements (optional). The OUT element takes a string value, and specifies a string or sub-string that must not be in the search string (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.

Usage Example

For this example, suppose you want to use GDCAX for Internet Explorer browsers and GJC for all other browser types. To achieve this, you have configure your application to support both Output Maps.

In your Genero Application Server configuration file (`as.xcf`), you would add a rule, such as `UseAllOutputDriver` shown below:

```
<APPLICATION Id="test" Parent="defaultwa">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>demo.42r</MODULE>
  </EXECUTION>
  <OUTPUT Rule="UseAllOutputDriver">
    <MAP Id="DUA_GJC" Allowed="TRUE">
      <RENDERING Using="cpn.rendering.wa"/>
      <THEME Using="cpn.theme.default.gjc"/>
    </MAP>
    <MAP Id="DUA_GDC" Allowed="TRUE">
      <RENDERING Using="cpn.rendering.wa"/>
      <THEME Using="cpn.theme.default.gdc"/>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

In `adua.xrd`, you would have the following:

```
<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_GJC</ACTION>
    </ROW>
  </TABLE>
</RULE>
```

With this example, if the User-Agent value contains `MSIE`, GDCAX will be used; otherwise the GJC (java applet) will be used.

Using the Debugger

This section provides instructions for using the debugger for the GAS.

Topics

- Using the debugger on Windows
- Using the debugger on Unix

Using the Debugger for the GAS on the Windows platform:

To run the FGL debugger, you have to tell gasd not to run "fglrun" directly; instead, gasd must open a DOS command or a xterm window and run "fglrun -d".

1. In %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">cmd /K start cmd</RESOURCE>` (Windows)
   `<RESOURCE Id="res.dvm.wa" Source="INTERNAL">/home/test/xterm.sh</RESOURCE>` (Unix)
2. In the application configuration file (default **as.xcf**), 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 gasd. (The gasd must be restarted whenever you modify the application server configuration file (default **as.xcf**) 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 <<< Sets the debugger on program test.42r.
   b test:20          <<< Sets a break point.
   run                <<< Runs the application.
   ```
   This refreshes the browser like FGL debugger does with GDC.

Tip: You can also run gasd from the command line and override some the settings for `res.dvm.wa`:

- `gasd -E res.dvm.wa="cmd /K start cmd"` (Windows)
- `gasd -E res.dvm.wa="/home/test/xterm.sh"` (Unix)

Warning! Using gasd as a service

If you are using gasd as a service, you need to allow the service to interact with the desktop.

- Select the service.
- Open the properties
- In the "Log On" folder tab, check "Allow service to interact with desktop".
- Apply the change.
- Restart the service.

Using the Debugger for the GAS on Unix

The following 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 gasd, enter the following:

```
gasd -E res.dvm.wa="/home/test/xterm.sh"
```

In the **xterm.sh** shell, you have: `/usr/X11R6/bin/xterm` (the complete path to xterm).

This removes all of the options given by gasd 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.

Validating Configuration (XCF) Files

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

Topics

What is an XML Schema Definition file?

An XML Schema Definition (XSD) file provides the syntax and defines a way in which elements and attributes can be represented in a XML document. It also advocates that the given XML document should be of a specific format and specific data type. 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.

Specifying the XSD file

The XSD file to use for validation is explicitly defined within the XML file. For example:

- In the GAS configuration file (as.xcf), the following entry exists:

```
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfas.xsd"
```

- In external configuration files, add the following entry if not present:

```
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.xsd"
```

- In the Connector configuration file (connector.xcf), the following entry exists:

```
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfconnector.xsd"
```

Validating with the gasd tool

There are two options that can prove useful in validating the GAS configuration file with the gasd tool:

--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 $FGLASDIR/tmp. Each file lists the entire configuration for an application, expanding the inherited components.

For more information on the options for the gasd command, see GAS Startup and Command Options.

Enhanced XML Editors

Any search for "XML Editors" across the various search engines return a long list of XML Editors that use the XML Schema Definition file to validate the XML within the various configuration (XCF) files. 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

Validation Steps

With a good XML editor, you can validate your Genero configuration (XCF) files. You can:

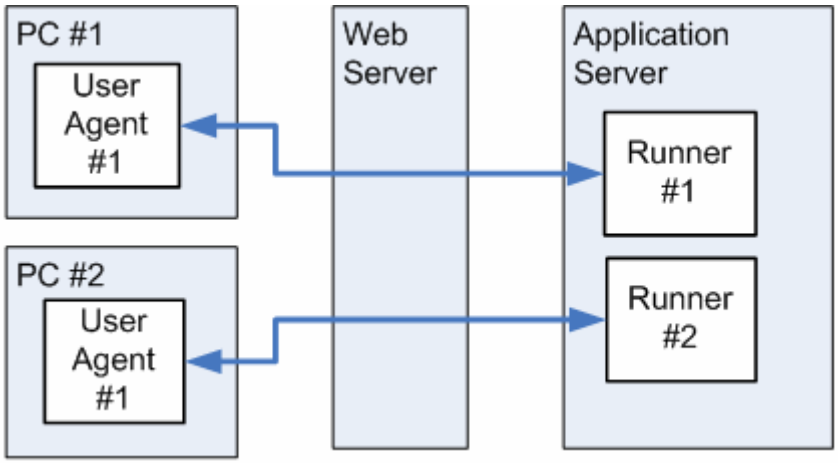
1. Check that your XML file is well-formatted.
2. Validate your XML file against the referenced XML Schema Definition (XSD) file.
3. Add new elements with completion assistance.

Licensing

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

- Licensing - Base Example
- Licensing - Using the RUN command
- Licensing - Multiple User Agents
- Licensing - Summary Case

Licensing - Base Example

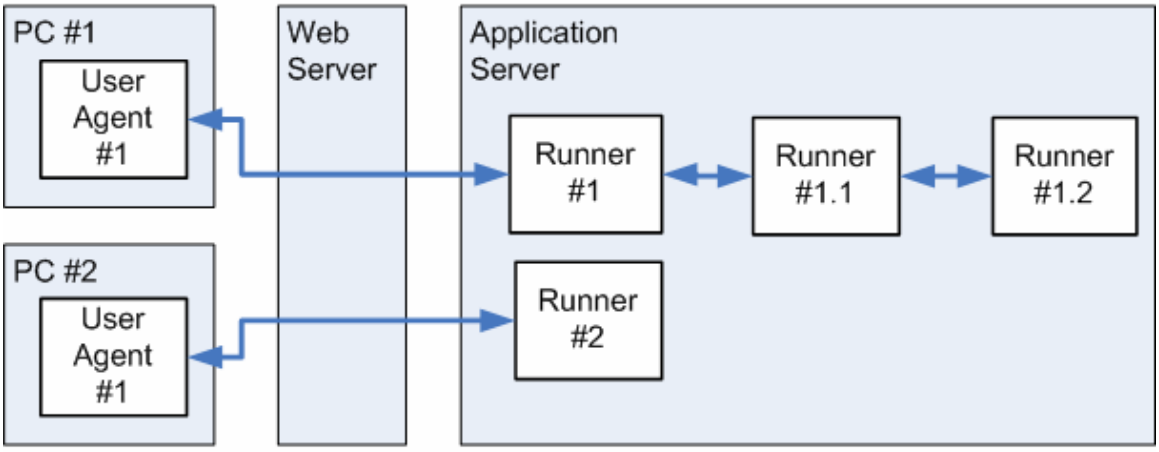


This diagram shows two (2) User Agents connected to two (2) DVMs. (The connection is made via the Web server, fglccgi / fglcisapi, and gasd.)

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.

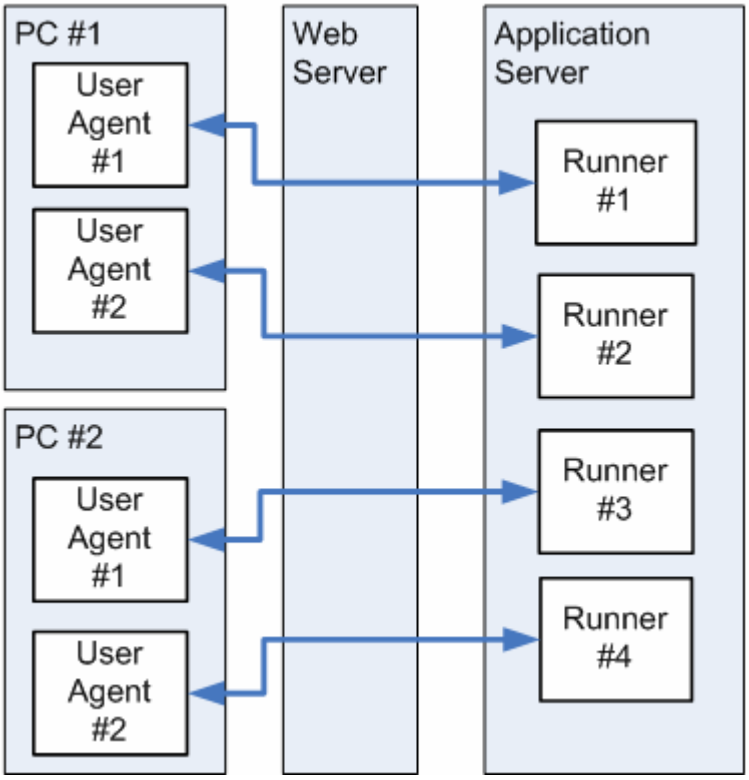
Licensing - Using the RUN command



This diagram shows two (2) 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.

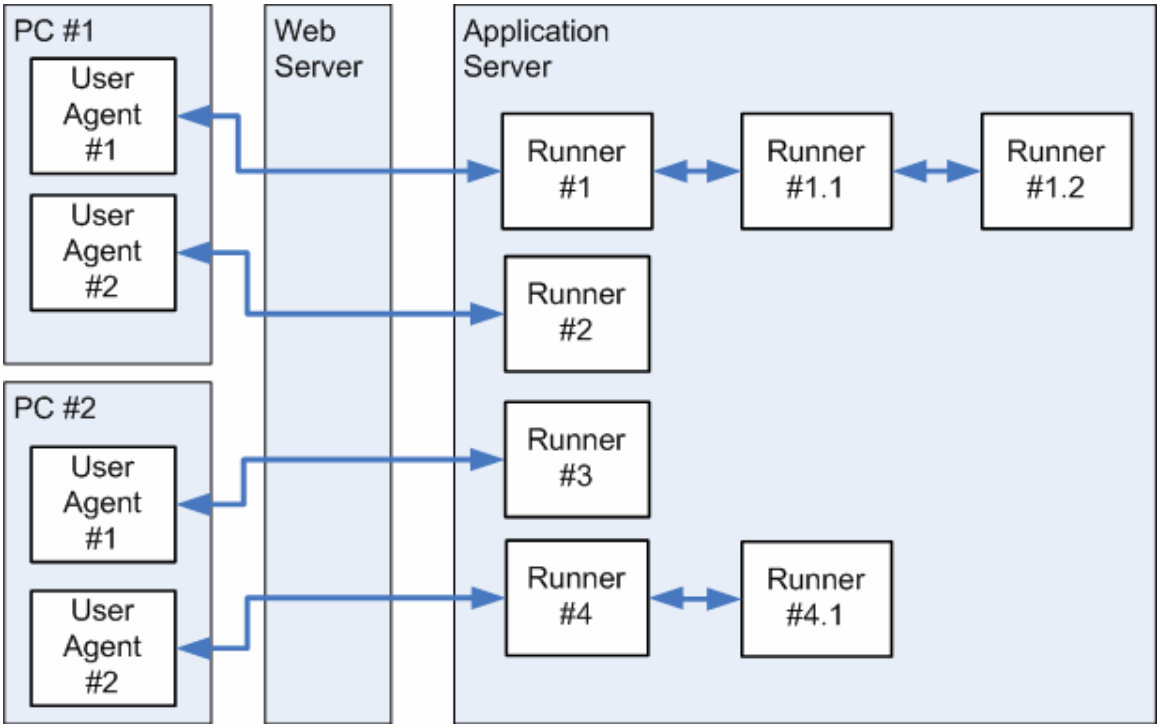
Licensing - Multiple User Agents



This diagram shows four (4) user agents running on two (2) different PCs and connected to four (4) DVMs.

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

Licensing - Summary Case



This diagram shows four (4) user agents running on two (2) different PCs and connected to four (4) 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

The Genero Front End -- whether GWC, GDC-HTTP, GDCAX, or GJC -- does not require any additional license information.

When a user requests an application, the gasd 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.

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 gasd 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`".

Warning

GDC-HTTP and GDCAX do not have the same license counting rules as the GDC has in other connection modes (rlogin, telnet or ssh). For the latter, one license is consumed per GDC monitor, no matter the number of applications launched. This is also true for GJC-HTTP and GJC Applet versus GJC in other connection modes.

The Application URI

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

Topics

- URI Syntax
- Examples

URI Syntax

The syntax of a URI follows the standards described in the RFC 2616. A list of example URIs is provided below.

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

Note: https is slower than http due to encryption.

Explanation of syntax options

Option	Data Type	Explanation	Valid Values
web-server	STRING	Name or IP address of the Web Server.	
web-server-port	INTEGER	Port on which the Web Server listens.	
directory	STRING	Any directory or virtual directory on the Web Server.	
script-directory	STRING	The script directory.	
connector	STRING	The name of the connector.	fglccgi, fglccgi.exe, fglcisapi.dll
app-server	STRING	Name or IP address of the Application Server.	
app-server-port	INTEGER	Port on which the Application Server listens.	
scope	STRING	Scope we are working on.	wa, ja, ws
action	STRING	Action requested of the Application Server.	r
group	STRING	Application Group defined in as.xcf	
web-application-id	STRING	Web Application identifier.	
parameter	STRING	Parameter to communicate to the Application Server.	Arg, UserAgent
parameter-value	STRING	Parameter value.	

URI Examples

Example 1 - Connection through CGI connector

Calls the "myApp" application through the "myWebServer" Web Server, using the CGI connector:

```
http://myWebServer/cgi-bin/fglccgi/wa/r/myApp
```

Note: On Windows platforms, when connecting via a Web server, you must include the extension when calling fglccgi.exe, as shown in the following URL:

```
http://myWebServer/cgi-bin/fglccgi.exe/wa/r/myApp
```

Example 2 - Connection through isapi connector

Calls the "myApp" application through the "myWindowsWebServer" Web Server, running IIS, using the ISAPI connector:

```
http://myWindowsWebServer/scripts/fglcisapi.dll/wa/r/myApp
```

Example 3 - Direct connection to gasd

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

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

Example 4 - Connection using a group

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

```
http://myWebServer/cgi-bin/fglccgi/wa/r/demo/myApp
```

Example 5 - Starting applications with arguments

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

```
http://myWebServer/cgi-bin/fglccgi/wa/r/myApp?Arg=Val1&Arg=Val2
```

Notes:

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 (&).

For more details on arguments configuration see the PARAMETERS section.

Example 6 - Calling Desktop application

Calls the "appid" application from the Genero Desktop Client monitor using a http connection.

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

A call to the same application using the GDCAX use the URL with "wa" not "ja".

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

Example 7 - 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 webserver (apache for example):

```
http://webserver/cgi-bin/fglccgi/ws/r/appid/service
```

Aliases

Creating an ALIAS element for the Genero Application Server is similar to creating an alias on a Web server. When you create an alias, you are creating a virtual directory. Any document you wish to publish should be in a directory that can be published; the alias is a mechanism for publishing a directory. Just as a Web browser uses an alias to publish content, the Genero Front End uses the alias to publish content required by the Front End application (such as images, cascading style sheets, html, and JavaScript).

Why specify an alias?

- Most aliases are shorter than the actual path name of the directory.
- An alias is more secure; users do not know where your files are physically located and cannot use that information to modify your files.
- Aliases make it easier for you to relocate directories for your site. Rather than of changing the path for the directory within your configuration files and templates, you simply change the mapping between the alias and the physical location of the directory.

Syntax:

```
<ALIAS Id="AliasPath"> physicalPath </ALIAS>
```

Notes:

When creating an ALIAS element, you specify the alias path with the Id attribute and you provide the physical path as the element's value.

1. *aliasPath* is the name of the alias.
2. *physicalPath* is the physical directory path.

Example:

```
<INTERFACE_TO_CONNECTOR>
 <TCP_BASE_PORT>6300</TCP_BASE_PORT>
 <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
 <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
 <ALIAS Id="/fjs/pics">/fjs/gas/web/fjs/pics</ALIAS>
</INTERFACE_TO_CONNECTOR>
```

In the example shown above, if the client requests the file /fjs/pics/myimage.png, the Genero Application Server returns myimage.png, found in the server directory /fjs/gas/web/fjs/pics.

For more information on working with images, see *Picture Component - Configuration Reference*.

Note: Some aliases are not taken into account if they are not defined in the right order. If alias X is a substring of alias Y, the aliases must be defined in order by the length of the alias path, from longest to shortest. For example:

- X is the alias /myapp for directory /xxx/test.
- Y is the alias /myapp/images for directory /yyy/images.

The definition of alias Y (/myapp/images) must appear before the definition of alias X (/myapp) to allow you to reach files in the physical path /yyy/images.

Authentication and the Genero Application Server

Authentication in the Genero Application Server provides the ability to:

- Start an application with authenticated user rights using a user's environment, profiles, access privileges, and so on.
- Provide a Single Sign-On (SSO) solution.

Topics

Using Kerberos

Authentication for applications served by the Genero Application Server utilizes the Kerberos protocol. Kerberos is a secure method for authenticating a request for a service in a computer network.

A free implementation of this protocol is available from the Massachusetts Institute of Technology (http://web.mit.edu/Kerberos). Kerberos is available in many commercial products as well.

Active Directory provides central authentication and authorization services for Windows-based computers. Kerberos is used by default for authentication in Active Directory.

System Requirements

Implementation constraints are related to the Kerberos architecture and implementation.

- A well-configured Active Directory / Kerberos server is required.
- Clients (to include the User Agent and Web Server Connector) must have network access to the Kerberos server.

Windows Only:

- The Web Server (Apache or IIS) must be launched as either the host account or a domain user account. It must not be launched as a local user account.

- The Genero Application Server must be launched as either as the Local System Account (the account, by default, when installing the Genero Application Server) or as Domain Administrator (or a member of the Domain Administrators group).

Configuring Authentication

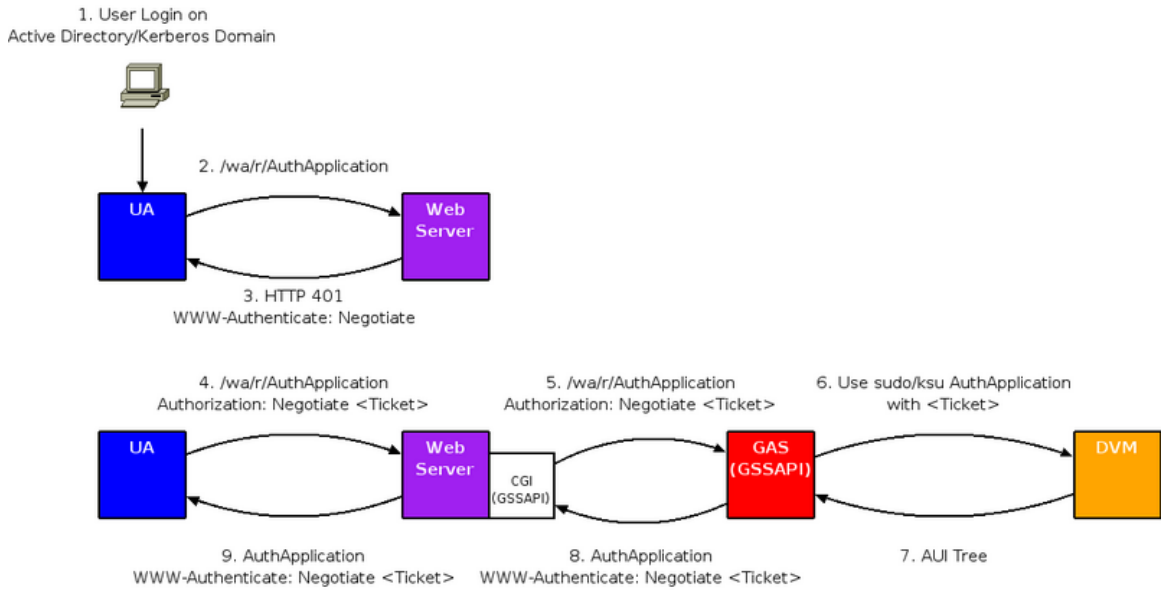
Authentication relies on Kerberos and it is assumed that the system administrator has set up the participating servers to support Kerberos. The list below summarizes the steps one must take to enable Kerberos authentication for a Genero application:

1. Set authentication configuration in the Genero Application Server configuration file (default as.xcf). See Application Server Configuration Reference: Authentication for more information.
2. Configure the application to use authentication by adding an AUTHENTICATION element to the application configuration. See Defining Applications , Defining Web Services, and Setting the Execution Environment for more information.
3. Update the Connector configuration file to use fully-qualified names for the Genero Application Server hosts. See Connector Configuration Reference for more information.

For a summarized walkthrough of implementing authentication for a GWC application, see "Authentication for GWC Applications" in the *Genero Web Client Manual*.

Authentication Process for Applications delivered by the GWC

This section outlines the authentication process for an application requiring authentication that is delivered via the Genero Web Client.



Authentication Process for a GWC Application

1. User logs on to the domain (such as Active Directory).
2. User attempts to access an application from a User Agent (browser), where access to the application is restricted and therefore requires authentication. For example, the user enters the URL for the application, such as http://server.fully.qualified.domain.name:6394/wa/r/AuthApplication.
3. User Agent receives an HTTP 401 response from the Web Server asking for authentication credentials. The response header includes: "WWW_Authenticate: Negotiate". An HTTP 401 response code is used when access to a resource is protected and the client did not provide valid authentication credentials.
4. The User Agent sends its granted ticket to the Web Server. The response header includes: "Authorization: Negotiate <Ticket>". At this point, the user is authenticated on the Web Server.
   The Web Server can now relay the request for the application through the Connector to the Genero Application Server.
5. The Web Server Connector sends the application request to the Genero Application Server along with another ticket that authenticates the user to the server. The ticket grants the access to the Genero Application Server; no additional login or password information is required.
6. The Genero Application Server starts the requested application by launching a Dynamic Virtual Machine (DVM) as the authenticated user.
   Note: When not using authentication, the DVM is started as the user who started the Genero Application Server.
7. The DVM sends the AUI tree to the Genero Application Server.
8. The Genero Application Server processes the AUI tree using the Genero Web Client and sends the resulting html page to the Web Server Connector.
9. The Connector forwards the page to the User Agent.

Enabling Kerberos authentication for a GWC application

Assuming your network has been configured to support Kerberos authentication, this section outlines the steps you must take to enable Kerberos authentication for your application. Note that this section is not intended to provide you with all possible configurations for Kerberos within the Genero Application Server, but instead highlights configuration changes necessary to implement Kerberos authentication in order for a Web application to be delivered by the Genero Web Client.

Step One: Configure the Genero Application Server

Configure the Genero Application Server to handle authentication using Kerberos by configuring the `<AUTHENTICATION>` element in the Genero Application Server configuration file (`as.xcf`):

```
<AUTHENTICATION Type="KERBEROS">
  <REALM></REALM>
  <SERVICE_NAME>gassvc</SERVICE_NAME>
  <KEYTAB>$(res.path.as)/etc/gwc.keytab</KEYTAB>
</AUTHENTICATION>
```

See also Authentication Configuration Reference

Step Two: Configure the application

Modify the application configuration to include an `<AUTHENTICATION>` element. As only Kerberos authentication is supported at this time, specify `KERBEROS` as the authentication type:

```
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
    <AUTHENTICATION>KERBEROS</AUTHENTICATION>
  </EXECUTION>
</APPLICATION>
```

See also Application List Reference

Step Three: Configure the Connector

Update the GAS Connector configuration file (`connector.xcf`) and replace all server IP addresses with fully-qualified domain names:

```
<REQUEST_LIST>
  <DEFAULT>
    <SERVER>myserver.mydomain.com:port</SERVER>
  </DEFAULT>
<REQUEST_LIST>
```

See also Connector Configuration Reference

Step Four (for Windows systems only): Configure Kerberos Service Principal Names

Two Kerberos Service Principal Names are used:

- An HTTP service name is used by the User Agent for access to the Web Server.
- gassvc service name is used by the Connector to ask for access to the Genero Application Server.

Internationalization and GAS

This section explains how the Genero Application Server handles international applications.

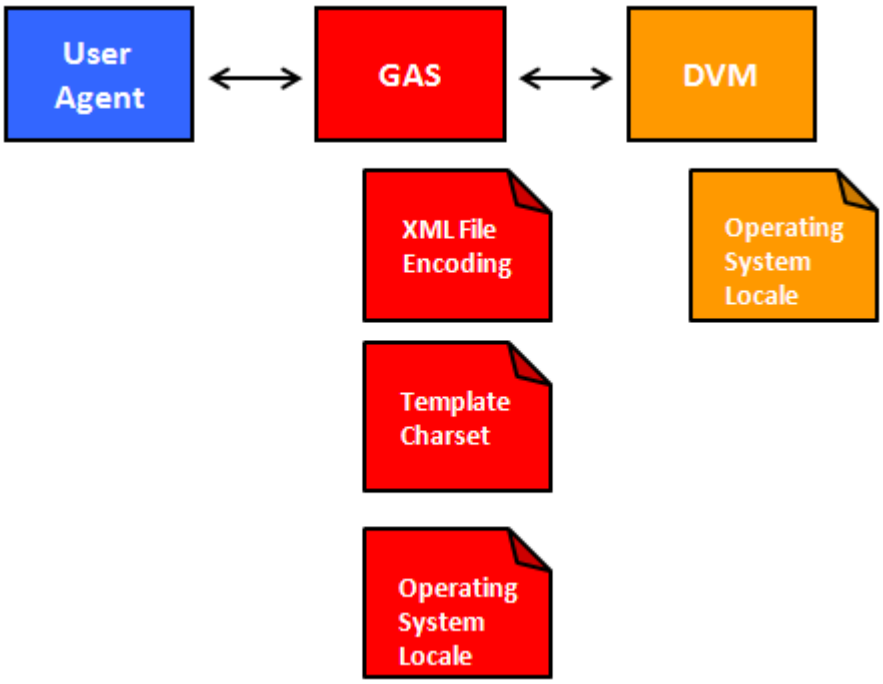
Topics

- Encoding Architecture
- Charsets Configuration
- Supported Charsets

NOTE: Encoding rules have been enhanced for the snippet-based rendering engine, used by the GAS for the GWC. 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 are applications using one or more non-ASCII character sets to support one or more languages. The diagram below summarizes the GAS encoding architecture:



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

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 in a specific encoding is described in the *Genero BDL Reference Manual*, section "Programming Applications", chapter "Localization". Locales can be set in the GAS executing environment, or with the `<ENVIRONMENT_VARIABLE>` tag inside the **as.xcf** file.

Example in as.xcf with KOI8-R (Russian) charset:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <?fjsApplicationServerConfiguration Version="1.30"?>
...
130 <COMPONENT_LIST>
131   <EXECUTION_COMPONENT Id="cpn.wa.execution.local">
132     <ENVIRONMENT_VARIABLE
Id="FGLDIR">$(res.fgldir)</ENVIRONMENT_VARIABLE>
133     <ENVIRONMENT_VARIABLE
Id="FGLGUI">$(res.fglgui)</ENVIRONMENT_VARIABLE>
134     <ENVIRONMENT_VARIABLE
Id="PATH">$(res.path)</ENVIRONMENT_VARIABLE>
...
139     <ENVIRONMENT_VARIABLE Id="LC_ALL">ru_RU.KIO8-
R</ENVIRONMENT_VARIABLE>
140     <DVM>$(res.dvm.wa)</DVM>
141   </EXECUTION_COMPONENT>
...
158 </COMPONENT_LIST>
```

HTML charset

In order to correctly handle application data in the User Agent, the HTML page charset needs to be set. Because GAS generates HTML pages from templates, charset needs to be defined in templates. Information about setting a charset in an HTML page can be found in HTML Specification - The Document Character Set.

Example in generodefault.html with BIG5 (Chinese) charset:

```
01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
```

```
03   <head>
04   $(res.meta-tags)
05   <meta http-equiv="Content-Type" content="text/html; charset=BIG5">
06   <title>Title of the page</title>
07
08   <script language=javascript
src="$(connector.uri)/fjs/uaapi/webBrowser.js"></script>
09   <script language=javascript
src="$(connector.uri)/fjs/asapi/application.js"></script>
...
19   </head>
...
```

XML Encoding

GAS uses XML files like `as.xcf` or external application configuration files, and these files may include international characters. How to define an encoding in an XML file is described in Extensible Markup Language - Character Encoding.

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

```
01 <?xml version="1.0" encoding="ISO-8859-6"?>
02 <?fjsApplicationServerConfiguration Version="1.30"?>
03 <CONFIGURATION>
...
```

GAS System Encoding

GAS interacts with Operating Systems in many ways:

- Writes log files
- Opens files defined in `as.xcf`
- Reads arguments on command line
- *and more*

In these cases, GAS needs to know which encoding is used by the Operating System. The Operating System encoding is defined via environment variables as described in The Single Unix - Specification Version 2 - Locale.

Example in command line with th_TH.tis620 (Thai) locale:

```
01 LC_ALL=th_TH.tis620 gasd -d
Then gasd starts with 'TIS-620' system encoding
```

Locales supported by an Operating System can be displayed with command `locale -a`. If the Operating System doesn't support the desired encoding, or if a specific encoding is needed, the system encoding can be defined with the `FGLAS_SYSENCODING` environment variable which overrides system locales.

Example in command line with UTF-8 :

```
01 LC_ALL=th_TH.tis620 FGLAS_SYSENCODING=UTF-8 gasd -d
```

```
Then gasd starts with 'UTF-8' system encoding
```

Note: Encodings have different names across Operating Systems. To unify them, GAS manages an encoding name conversion. For each UNIX platform, a `charset.alias` file is provided for mapping the Operating System encoding name to a canonical encoding name.

Default Encoding

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

Supported Charsets

The following list contains all character sets known by the GAS. One coded character set can be listed with several different names. Depending on your Operating System, DVM may support these character sets. Refer to the *Genero BDL Reference Manual*, section "Programming Applications", chapter "Localization" for more information.

```
ANSI_X3.4-1968 ANSI_X3.4-1986 ASCII CP367 IBM367 ISO-IR-6 ISO646-US
ISO_646.IRV:1991 US US-ASCII CSASCII
UTF-8
ISO-10646-UCS-2 UCS-2 CSUNICODE
UCS-2BE UNICODE-1-1 UNICODEBIG CSUNICODE11
UCS-2LE UNICODELITTLE
ISO-10646-UCS-4 UCS-4 CSUCS4
UCS-4BE
UCS-4LE
UTF-16
UTF-16BE
UTF-16LE
UTF-32
UTF-32BE
UTF-32LE
UNICODE-1-1-UTF-7 UTF-7 CSUNICODE11UTF7
UCS-2-INTERNAL
UCS-2-SWAPPED
UCS-4-INTERNAL
UCS-4-SWAPPED
C99
JAVA
CP819 IBM819 ISO-8859-1 ISO-IR-100 ISO8859-1 ISO_8859-1 ISO_8859-1:1987
L1 LATIN1 CSISOLATIN1
ISO-8859-2 ISO-IR-101 ISO8859-2 ISO_8859-2 ISO_8859-2:1987 L2 LATIN2
CSISOLATIN2
ISO-8859-3 ISO-IR-109 ISO8859-3 ISO_8859-3 ISO_8859-3:1988 L3 LATIN3
CSISOLATIN3
ISO-8859-4 ISO-IR-110 ISO8859-4 ISO_8859-4 ISO_8859-4:1988 L4 LATIN4
CSISOLATIN4
CYRILLIC ISO-8859-5 ISO-IR-144 ISO8859-5 ISO_8859-5 ISO_8859-5:1988
CSISOLATINCYRILLIC
```

```
ARABIC ASMO-708 ECMA-114 ISO-8859-6 ISO-IR-127 ISO8859-6 ISO_8859-6
ISO_8859-6:1987 CSISOLATINARABIC
ECMA-118 ELOT_928 GREEK GREEK8 ISO-8859-7 ISO-IR-126 ISO8859-7
ISO_8859-7 ISO_8859-7:1987 CSISOLATINGREEK
HEBREW ISO-8859-8 ISO-IR-138 ISO8859-8 ISO_8859-8 ISO_8859-8:1988
CSISOLATINHEBREW
ISO-8859-9 ISO-IR-148 ISO8859-9 ISO_8859-9 ISO_8859-9:1989 L5 LATIN5
CSISOLATIN5
ISO-8859-10 ISO-IR-157 ISO8859-10 ISO_8859-10 ISO_8859-10:1992 L6
LATIN6 CSISOLATIN6
ISO-8859-13 ISO-IR-179 ISO8859-13 ISO_8859-13 L7 LATIN7
ISO-8859-14 ISO-CELTIC ISO-IR-199 ISO8859-14 ISO_8859-14 ISO_8859-
14:1998 L8 LATIN8
ISO-8859-15 ISO-IR-203 ISO8859-15 ISO_8859-15 ISO_8859-15:1998 LATIN-9
ISO-8859-16 ISO-IR-226 ISO8859-16 ISO_8859-16 ISO_8859-16:2001 L10
LATIN10
KOI8-R CSKOI8R
KOI8-U
KOI8-RU
CP1250 MS-EE WINDOWS-1250
CP1251 MS-CYRL WINDOWS-1251
CP1252 MS-ANSI WINDOWS-1252
CP1253 MS-GREEK WINDOWS-1253
CP1254 MS-TURK WINDOWS-1254
CP1255 MS-HEBR WINDOWS-1255
CP1256 MS-ARAB WINDOWS-1256
CP1257 WINBALTRIM WINDOWS-1257
CP1258 WINDOWS-1258
850 CP850 IBM850 CSPC850MULTILINGUAL
862 CP862 IBM862 CSPC862LATINHEBREW
866 CP866 IBM866 CSIBM866
MAC MACINTOSH MACROMAN CSMACINTOSH
MACCENTRALEUROPE
MACICELAND
MACCROATIAN
MACROMANIA
MACCYRILLIC
MACUKRAINE
MACGREEK
MACTURKISH
MACHEBREW
MACARABIC
MACTHAI
HP-ROMAN8 R8 ROMAN8 CSHPROMAN8
NEXTSTEP
ARMSCII-8
GEORGIAN-ACADEMY
GEORGIAN-PS
KOI8-T
MULELAO-1
CP1133 IBM-CP1133
ISO-IR-166 TIS-620 TIS620 TIS620-0 TIS620.2529-1 TIS620.2533-0
TIS620.2533-1
CP874 WINDOWS-874
VISCII VISCII1.1-1 CSVISCII
TCVN TCVN-5712 TCVN5712-1 TCVN5712-1:1993
ISO-IR-14 ISO646-JP JIS_C6220-1969-RO JP CSISO14JISC6220RO
```

```
JISX0201-1976 JIS_X0201 X0201 CSHALFWIDTHKATAKANA
ISO-IR-87 JIS0208 JIS_C6226-1983 JIS_X0208 JIS_X0208-1983 JIS_X0208-
1990 X0208 CSISO87JISX0208
ISO-IR-159 JIS_X0212 JIS_X0212-1990 JIS_X0212.1990-0 X0212
CSISO159JISX02121990
CN GB_1988-80 ISO-IR-57 ISO646-CN CSISO57GB1988
CHINESE GB_2312-80 ISO-IR-58 CSISO58GB231280
CN-GB-ISOIR165 ISO-IR-165
ISO-IR-149 KOREAN KSC_5601 KS_C_5601-1987 KS_C_5601-1989 CSKSC56011987
EUC-JP EUCJP EXTENDED_UNIX_CODE_PACKED_FORMAT_FOR_JAPANESE
CSEUCPKDFMTJAPANESE
MS_KANJI SHIFT-JIS SHIFT_JIS SJIS CSSHIFTJIS
CP932
ISO-2022-JP CSISO2022JP
ISO-2022-JP-1
ISO-2022-JP-2 CSISO2022JP2
CN-GB EUC-CN EUCCN GB2312 CSGB2312
CP936 GBK MS936 WINDOWS-936
GB18030
ISO-2022-CN CSISO2022CN
ISO-2022-CN-EXT
HZ HZ-GB-2312
EUC-TW EUCTW CSEUCTW
BIG-5 BIG-FIVE BIG5 BIGFIVE CN-BIG5 CSBIG5
CP950
BIG5-HKSCS BIG5HKSCS
EUC-KR EUCKR CSEUCKR
CP949 UHC
CP1361 JOHAB
ISO-2022-KR CSISO2022KR
```

Configuring the GAS Connector

To include a Web server in your solution, you must install the GAS Connector on the host of the Web server. Once the GAS Connector is installed, you modify the configuration file **connector.xcf** and define how the Web server routes its application requests, provide load-balancing of applications across application servers, and what error messages display.

Topics

- What is a GAS Connector?
- Routing an application request
- Load balancing application requests
- Setting error messages

Important: This help topic provides examples of common configuration needs. For a complete listing of all configuration options, see the *Connector Configuration Reference*.

What is a GAS Connector?

A GAS Connector allows a Web server to handle requests from the user agent to the Genero Application Server daemon (gasd). The GAS Connector can dispatch requests to different Application Server daemons based on the requested application, resulting in load balancing of applications. The configuration of the GAS Connector is completed in the **connector.xcf** file.

The Connector configuration file is installed on the webserver in the directory <*webserver*>/<*script*>, where *webserver* is the document root directory and *script* the executables directory. See *Installation* for more information on installing and locating the Connector configuration file,

---

Routing an application request

When the GAS Connector receives an application request, it needs to know two things: the application server host and the port number where the application server is listening for new requests. This server and port number information must be provided in the configuration file.

Within the REQUEST_LIST element, you define REQUEST elements that specify the server and port offset information for an application request for a named application (specified by the Id tag). Create a REQUEST element for each application you wish to specifically route. Using a REQUEST element routes an application to one or more

application servers based on its application Id. This allows you to effectively run an application on a dedicated server, if that is a need. You also have a DEFAULT element, specifying the server and port offset for application requests not explicitly named in any REQUEST element. If you have no application servers defined in the DEFAULT element, then only applications whose Id matches a REQUEST element will be run.

Elsewhere, within the INTERFACE_TO_APPLICATION_SERVER element, you define a single **base port** value in the TCP_BASE_PORT element. This value is added to the port offset to identify the port number where the application server is listening for application requests.

Example:

```
01 <CONFIGURATION>
02  <CONNECTOR>
03    <INTERFACE_TO_APPLICATION_SERVER>
04      <TCP_BASE_PORT>6300</TCP_BASE_PORT>
05    </INTERFACE_TO_APPLICATION_SERVER>
06    <REQUEST_LIST>
07      <DEFAULT>
08        <SERVER>server_1:94</SERVER>
09      </DEFAULT>
10      <REQUEST Id="Edit">
11        <SERVER>server_3:95</SERVER>
12     </REQUEST>
13   </REQUEST_LIST>
14 ...
15 </CONFIGURATION>
```

In this example:

- Lines 03-05 define the base port as 6300.
- Lines 07-09 define the default server, which will be forwarded all application requests not handled by an REQUEST element.
- Line 08 specifies the server (by name in this example), and gives a port offset of 94. Add this to the base port, and the GAS Connector will send the application requests for this server to port 6394.
- Lines 10-12 define the server to receive all requests for the application named "Edit".
- Line 11 specifies the server (again by name), and gives a port offset of 95. Add this to the base port, and the GAS Connector will send the application requests for this server to port 6395.

For detailed configuration guidance, refer to the *Connector Configuration Reference*.

Load balancing application requests

As discussed in the previous section, the REQUEST and DEFAULT elements of the GAS Connector configuration file specifies to which server an application request is routed.

Balancing requests by application name

One method of load balancing is to have specific application servers dedicated to handling specific application requests. In this setup, you simply add the requisite REQUEST elements to the configuration file.

```
01 <CONFIGURATION>
02  <CONNECTOR>
03  ...
04    <REQUEST_LIST>
05      <DEFAULT>
06        <SERVER>server_1:94</SERVER>
07      </DEFAULT>
08      <REQUEST Id="Edit">
09        <SERVER>server_3:94</SERVER>
10     </REQUEST>
11     <REQUEST Id="Button">
12       <SERVER>192.168.0.10:94</SERVER>
13     </REQUEST>
14   </REQUEST_LIST>
15 ...
16 </CONFIGURATION>
```

In this example, application requests are balanced across three servers:

- Lines 05-07 define the server to receive all application requests that are NOT for the "Edit" or "Button" application.
- Lines 08-10 define the server to receive all requests for the "Edit" application.
- Linex 11-13 define the server to receive all requests for the "Button" application.

Balancing across application servers

A second method of load balancing is to have multiple application servers handle requests for one application. This method is useful when a single application server cannot keep up with the number of requests being sent to it. To distribute application requests across multiple application servers, define multiple SERVER elements within the REQUEST or DEFAULT element. Balancing the requests across the listed servers is managed by the GAS Connector. The requests are evenly spread across the servers listed.

```
01 <CONFIGURATION>
02  <CONNECTOR>
03  ...
04    <REQUEST_LIST>
05      <DEFAULT>
```

```
06        <SERVER>server_1:94</SERVER>
07        <SERVER>server_2:94</SERVER>
08      </DEFAULT>
09      <REQUEST Id="Edit">
10        <SERVER>server_3:94</SERVER>
11        <SERVER>server_4:94</SERVER>
12     </REQUEST>
13     <REQUEST Id="Button">
14       <SERVER>192.168.0.10:94</SERVER>
15     </REQUEST>
16   </REQUEST_LIST>
17 ...
18 </CONFIGURATION>
```

In this example, requests are routed across six application servers:

- Lines 05 - 08 specify the set of application servers that will serve all applications not specified by name in a REQUEST element. Requests are evenly distributed across the two servers specified.
- Lines 09 - 12 specify the two application servers that will serve all requests for the "Edit" application. Requests are evenly distributed across the two servers specified.
- Lines 13 - 15 specify the single application server that will server all requests for the "Button" application.

For detailed configuration guidance, refer to the *Connector Configuration Reference*.

Setting the error messages

The administrator specifies the errors that display when the application cannot be served. The default installation provides html files that display appropriate error pages for six common error types. The administrator can customize these html files or change other error-related settings.

Syntax:

```
<ERROR_LIST>
  <ERROR Id="num">
    <HTTP_STATUS>status</HTTP_STATUS>
    <HTTP_HEADER Id="hname">hvalue</HTTP_HEADER> [...]
    <BODY_FILE>errfile</BODY_FILE>
  </ERROR>
<ERROR_LIST>
```

Notes:

1. *num* is the error identifier
2. *status* is the http status

3. *hname* is the http header name, for example Pragma
4. *hvalue* is the *hname* value, for example no-cache
5. *errfile* is the file to display when the error occurs

Genero Web Services Notes:

1. For SOAP errors, *status* is ignored and set to "500 Internal Server Error" as specified in Simple Object Access Protocol (SOAP) 1.1 - 6.2 SOAP HTTP Response.

Example:

```
01 <ERROR_LIST>
02   <ERROR Id="1">
03     <HTTP_STATUS>400 Bad Request</HTTP_STATUS>
04     <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
05     <BODY_FILE>connector-error-1</BODY_FILE>
06   </ERROR>
07  ...
08   <ERROR Id="3">
09     <HTTP_STATUS>404 Not Found</HTTP_STATUS>
10     <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
11     <BODY_FILE>connector-error-3</BODY_FILE>
12   </ERROR>
13   ...
14 </ERROR_LIST>
```

For detailed configuration guidance, refer to the *Connector Configuration Reference.*

Connector Configuration Reference

The file **connector.xcf** is the configuration file for the Genero Application Server (GAS) Connector. The file is stored in the Web Server's CGI binaries directory.

Syntax:

```
<CONFIGURATION>
  <CONNECTOR>
    <INTERFACE_TO_APPLICATION_SERVER>
       ...
    </INTERFACE_TO_APPLICATION_SERVER>
    <REQUEST_LIST>
      ...
    </REQUEST_LIST>
    <ERROR_LIST>
      ...
    </ERROR_LIST>
  </CONNECTOR>
</CONFIGURATION>
```

Notes:

1. One `CONFIGURATION` element. The main element of all configuration files. There are no attributes for the `CONFIGURATION` element.
2. One `CONNECTOR` element. A connector is uniquely defined in the configuration file. There are no attributes for the `CONNECTOR` element.
3. Zero or one `INTERFACE_TO_APPLICATION_SERVER` element. This section defines the parameters necessary to access the Application Server by providing the application port definition section.
4. One `REQUEST_LIST` element. This section lists all the requests to be forwarded to the Application Server. A request can be made for a Web Service or a Web Application. The request forwarding section.
5. Zero or one `ERROR_LIST` element. This section contains all errors that the connector can produce, and specifies which files are displayed when an error occurs.

INTERFACE_TO_APPLICATION_SERVER

Within the `INTERFACE_TO_APPLICATION SERVER` element, you specify the base port for the Genero Application Server Connector. The base port specifies which port will be used in conjunction of the offset specified in each server.

Syntax:

```
<CONFIGURATION>
  <CONNECTOR>
    <INTERFACE_TO_APPLICATION_SERVER>
      <TCP_BASE_PORT> port </TCP_BASE_PORT>
    </INTERFACE_TO_APPLICATION_SERVER>
    request_list section
    error_list section
  </CONNECTOR>
</CONFIGURATION>
```

Notes:

1. *port* is the base port number. This number is added to the port offset specified in a `SERVER` element within the `REQUEST_LIST` section. Usually, this base port corresponds to the `TCP_BASE_PORT` for the Genero Application Server, defined in the `INTERFACE_TO_CONNECTOR` section of the Genero Application Server configuration file.
2. The default value for the `TCP_BASE_PORT` element is 6300.
3. There are no attributes available for the `TCP_BASE_PORT` element.

Example:

```
01 <INTERFACE_TO_APPLICATION_SERVER>
02   <TCP_BASE_PORT>6300</TCP_BASE_PORT>
03 </INTERFACE_TO_APPLICATION_SERVER>
```

REQUEST_LIST

The `REQUEST_LIST` element lists all the requests to be forwarded to the Application Server. A request can be made for a Web Service or a Web Application. This section defines the ports where application servers are listening for requests. At least one default server must be specified, although multiple default servers can be listed to provide load balancing for performance. Application-specific application servers can also be specified.

Syntax:

```
<CONFIGURATION>
  <CONNECTOR>
    interface_to_application_server section
    <REQUEST_LIST>
      <DEFAULT>
        <SERVER> defserver:offset </SERVER> [ ... ]
      </DEFAULT>
      rqServer [...]
```

```
    </REQUEST_LIST>
    error_list section
  </CONNECTOR>
</CONFIGURATION>
```

where *rqServer* is:

```
<REQUEST Id="appName">
  <SERVER>appServer:offset</SERVER> [...]
</REQUEST>
```

Notes:

1. *defserver* is the default application server name or IP address.
   NOTE: When configuring the connector for participation in a solution that includes Kerberos authentication, you must specify the fully-qualified name of the server (as shown in Example 2 below).
2. *offset* is the offset port number for an application server. This value, when added to the application server base port (specified in the INTERFACE_TO_APPLICATION_SERVER section), identifies the port where the application server is listening for requests. For example, the default port for GAS is 6394 (the base port of 6300 + the offset of 94).
3. You can only specify one `DEFAULT` element. Within the `DEFAULT` element, you can specify multiple `SERVER` elements.
4. *appName* is the unique identifier (Id) of either the Web Application or the Web Service, as defined in the Genero Application Server configuration file for the specified *appserver* or in an external application configuration file accessible by the Genero Application Server specified by *appserver*.
5. *appServer* is a Genero Application Server server name or IP address.
6. You can specify multiple `REQUEST` elements. A request is defined by its required Id attribute. The Id attribute must either match an application Id specified for an `APPLICATION` element in the Genero Application Server configuration file, a Web Service Id specified in the Genero Application Server configuration file, or the filename of an external application configuration file accessible by the Genero Application Server. If the Id does not match either of these, the GAS Connector will answer requests for this Id with a "404 object not found" error.
7. There are no available attributes for the `REQUEST_LIST`, `DEFAULT`, and `SERVER` elements.

Example 1:

```
01 <REQUEST_LIST>
02   <DEFAULT>
03     <SERVER>localhost:94</SERVER>
04     <SERVER>localhost:95</SERVER>
05   </DEFAULT>
06   <REQUEST Id="Edit">
07     <SERVER>localhost:96</SERVER>
08   </REQUEST>
09 </REQUEST_LIST>
```

In this example, if the called application is "Edit", the request is redirected to the application server launched on port 6396 (base port + offset), otherwise the request is sent to one of the default servers listening on ports 6394 and 6395.

Example 2:

```
01 <REQUEST_LIST>
02   <DEFAULT>
03     <SERVER>myserver.mydomain.com:94</SERVER>
04   </DEFAULT>
05 </REQUEST_LIST>
```

In this example, the fully-qualified domain name is used to define the server. You must use a fully-qualified domain name when using Kerberos authentication.

ERROR_LIST

The ERROR_LIST element specifies what information is displayed when specific errors occur.

Syntax:

```
<CONFIGURATION>
  <CONNECTOR>
    interface_to_application_server section
    request_list section
    <ERROR_LIST>
      <ERROR Id="errID">
        <HTTP_STATUS> status </HTTP_STATUS>
        <HTTP_HEADER Id=" header"> headerValue </HTTP_HEADER>
        <BODY_FILE> filename </BODY_FILE>
      </ERROR>
      [ ... ]
    </ERROR_LIST>
  </CONNECTOR>
</CONFIGURATION>
```

Syntax:

```
<ERROR_LIST>
   error [...]
</ERROR_LIST>
```

where *error* is:

```
<ERROR Id="code">
```

```
  <HTTP_STATUS> status </HTTP_STATUS>
  <HTTP_HEADER Id="header"> value </HTTP_HEADER>
  <BODY_FILE> filename </BODY_FILE>
</ERROR>
```

Notes:

1. *code* is the error code; a string value specifying the connector error identifier. There are six different error codes:

Error Code	Description
1	Application ID not specified.
2	Unable to find the configuration.
3	Application not found.
4	Cannot connect to the Application Server.
5	Cannot connect to the Application Server any longer.
6	Connection lost between the Connector and the Application Server.

2. *status* is the http status associated to this error, such as:

   ```
   400 Bad Request
   ```

3. HTTP status to return when the error occurs.
4. *header* is the header variable name, such as:

   ```
   Pragma
   ```

5. *value* is the header variable value, such as:

   ```
   no-cache
   ```

6. When set to `no cache`, it directs the browser to not keep the file in the cache. The headers are put before the error message, like cache control, redirection, and so on.

7. *filename* is the name of the file associated to this error. The file contains the html to display when the error is encountered, and is located in your Web Server script directory (the same directory that stores the **connector.xcf** file).

Genero Web Services Notes:

1. For SOAP errors, *status* is ignored and set to "500 Internal Server Error" as specified in Simple Object Access Protocol (SOAP) 1.1 - 6.2 SOAP HTTP Response.

Example:

```
<ERROR_LIST>
  <ERROR Id="1">
    <HTTP_STATUS>400 Bad Request</HTTP_STATUS>
    <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
    <BODY_FILE>connector-error-1</BODY_FILE>
  </ERROR>
  <ERROR Id="2">
    <HTTP_STATUS>500 Internal Server Error</HTTP_STATUS>
    <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
    <BODY_FILE>connector-error-2</BODY_FILE>
  </ERROR>
  <ERROR Id="3">
    <HTTP_STATUS>404 Not Found</HTTP_STATUS>
    <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
    <BODY_FILE>connector-error-3</BODY_FILE>
  </ERROR>
  <ERROR Id="4">
    <HTTP_STATUS>503 Service Unavailable</HTTP_STATUS>
    <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
    <BODY_FILE>connector-error-4</BODY_FILE>
  </ERROR>
  <ERROR Id="5">
    <HTTP_STATUS>503 Service Unavailable</HTTP_STATUS>
    <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
    <BODY_FILE>connector-error-5</BODY_FILE>
  </ERROR>
  <ERROR Id="6">
    <HTTP_STATUS>502 Bad Gateway</HTTP_STATUS>
    <HTTP_HEADER Id="Pragma">no-cache</HTTP_HEADER>
    <BODY_FILE>connector-error-6</BODY_FILE>
  </ERROR>
</ERROR_LIST>
```

This example is taken from the default settings in the GAS Connector configuration file.

Adding a GDCAX or GJC Application

This section will help you to set up an application rapidly. For complete details on possible configuration parameters and settings, see *Application List Reference*.

Topics

- An overview of application configuration
- Creating an abstract application
- Configuring an application for the Genero Desktop Client ActiveX
- Configuring an application for the Genero Java Client
- An overview of external application configuration
- Creating an external application group
- Configuring an external application

Application Configuration Overview

When you configure an application, there are many pieces of information that must be provided to the Genero Application Server. However, much of this information is common across a set of applications. Therefore, 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 default configuration from another application, you specify it as the parent of the application.

In general, an abstract application is defined first. This abstract application is not executable, but is intended to provide the baseline default configuration for other applications to inherit. You can create as many abstract applications as you require, and abstract applications can inherit a default configuration from another abstract application.

When configuring an application that is to be an executable, you can provide the configuration details in either the Genero Application Server configuration file, or you can create a separate application-specific configuration file, known as an ***external application configuration file***. If you add the application to the Genero Application Server configuration file, you must stop and restart the Genero Application Server for the application to be recognized. If you create an external application configuration file, you can add the file into a defined GROUP directory and the application is immediately available without having to do a GAS restart.

See also:

- Quick Start - Adding New Applications
- Configuring the Genero Application Server
- Application List Reference

Creating Abstract Applications

To simplify application configuration, an application can specify a parent application to provides default configuration settings needed to run the application. An abstract application is an application that is created not as an executable, but to be a parent 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 a configuration file.

To specify an abstract application, set the **Abstract** attribute to **TRUE**.

Warning: Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.

Example for web applications:

```
<APPLICATION Id="defaultwa" Abstract="TRUE">
  <EXECUTION Using="cpn.wa.execution.local"/>
  <OUTPUT>
    <MAP Id="DUA_GWC" Allowed="FALSE"/>
    <MAP Id="DUA_GJC" Allowed="FALSE"/>
    <MAP Id="DUA_GDC" Allowed="FALSE"/>
  </OUTPUT>
</APPLICATION>
```

In this example, `DUA_GWC`, `DUA_GJC` and `DUA_GDC` are OutputMap, which indicates the Front End used to display the application. Note that no OutputMap is enabled as the attribute `Allowed` is set to `FALSE`.

Example for GDCAX applications:

```
<!--This is the default application for GDC-->
<APPLICATION Id="defaultgdc" Parent="defaultwa" Abstract="TRUE">
  <OUTPUT Rule="UseGDC">
    <MAP Id="DUA_GDC" Allowed="TRUE"/>
      <RENDERING Using="cpn.rendering.wa"/>
      <THEME Using="cpn.theme.default.gdc"/>
    </MAP>
  </OUTPUT>
</APPLICATION>
```

Configuring applications for Genero Desktop Client ActiveX (GDC/AX)

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

- your application Id
- the parent application where the main configuration is set (in this example, **defaultgdc**)
- the path to your compiled files
- the main module to launch

In the following example the path is a resource; this can also be an absolute path to your application files.

Example:

```
01 <APPLICATION Id="gdc-demo" Parent="defaultgdc">
02   <EXECUTION>
03     <PATH>$(res.path.fgldir.demo)</PATH>
04     <MODULE>demo.42r</MODULE>
05   </EXECUTION>
06 </APPLICATION>
```

defaultgdc is the parent of any web application for GDCAX.

Example:

```
01 <APPLICATION Id="defaultgdc" Parent="defaultwa" Abstract="TRUE">
02   <TIMEOUT Using="cpn.gdc-gjc.timeout.set1"/>
03   <OUTPUT Rule="UseGDC">
04     <MAP Id="DUA_GDC" Allowed="TRUE">
05       <RENDERING Using="cpn.rendering.gdc-gjc"/>
06       <THEME Using="cpn.theme.default.gdc"/>
07     </MAP>
08   </OUTPUT>
09 </APPLICATION>
```

The **defaultgdc** application inherits from the **defaultwa** application. In the Abstract application section, **defaultwa** is the parent for any web application and no OutputMap is enabled; **defaultgdc** enables `DUA_GDC` OutputMap and specifies the environment for this OutputMap.

Configuring applications for Genero Java Client (GJC)

To add an application for GJC, you only need to specify:

- your application Id
- the parent application where the main configuration is set (in this example, **defaultgjc**)
- the path to your compiled files
- the main module to lunch

In the following example the path is a resource; this can also be an absolute path to your application files.

Example:

```
01 <APPLICATION Id="gjc-demo" Parent="defaultgjc">
02  <EXECUTION>
03    <PATH>$(res.path.fgldir.demo)</PATH>
04    <MODULE>demo.42r</MODULE>
05  </EXECUTION>
06 </APPLICATION>
```

defaultgjc is the parent of any web application for GJC.

Example:

```
01 <APPLICATION Id="defaultgjc" Parent="defaultwa" Abstract="TRUE">
02  <TIMEOUT Using="cpn.gdc-gjc.timeout.set1"/>
03  <OUTPUT Rule="UseGJC">
04   <MAP Id="DUA_GJC" Allowed="TRUE">
05    <RENDERING Using="cpn.rendering.gdc-gjc"/>
06    <THEME Using="cpn.theme.default.gjc"/>
07   </MAP>
08  </OUTPUT>
09 </APPLICATION>
```

The **defaultgjc** application inherits from the **defaultwa** application. In the Abstract application section, **defaultwa** is the parent for any web application and no OutputMap is enabled; **defaultgjc** just enables `DUA_GJC` OutputMap and specifies the environment for this OutputMap.

Using External Application Configuration Files

To configure an application using an external application configuration file, you provide the same code that you would for adding an application directly to the Genero

Application Server configuration file, except that you store this information in a file whose name matches that of the application. For example, to create a file for a program named **Edit**, you would create an external application configuration file with the name **Edit.xcf**. You must use the **.xcf** suffix. You then place this file in a group directory as configured in the Genero Application Server configuration file.

Creating a Group

A ***group*** consists of a group name and a directory in which external application configuration files can be placed. When a front-end starts an application whose configuration information is in an external application configuration file, it must provide the group name to direct the Genero Application Server to the directory where the file resides, and the application name to identify which file to read. As with applications, a group is specified in the APPLICATION_LIST component of the Genero Application Server configuration file.

Syntax:

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

Notes:

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

Example:

```
01 <GROUP Id="_default">$(res.path.app)</GROUP>
02 <GROUP Id="demo-gdc">$(res.path.app)/tutorial/appNotes</GROUP>
```

For information on how Genero Front-Ends may use GROUP entries, refer to the documentation for the specific front-end.

Configuring an External Application

Create a separate .xcf file for each application. Because the application and configuration file share the same name, there is no need to specify the **Id** attribute. In the following example, if the file was named "gdc-demo.xcf, then this configuration file would accomplish the same task as when included in the Genero Application Server configuration file; the only difference between this example and the example shown above for Genero Desktop Client is the lack of the **Id** attribute.

Example:

```
01 <APPLICATION Parent="defaultgdc">
02   <EXECUTION>
03     <PATH>$(res.path.fgldir.demo)</PATH>
04   </EXECUTION>
05 </APPLICATION>
```

How Templates Work for the GDCAX or GJC

A template is an html file that displays your application through a browser, using a Genero Front End. A template can have predefined variables or tags. Templates are stored in $FGLASDIR/tpl directory. Genero Web Services Extension does not need a template as it is not a Front End but a server that waits for requests.

You can define your own templates and use them in your applications. See the GAS Configuration Reference section on how to set up a template.

Topics

- GAS tags
- GDCAX Template
- GJC Template

GAS tags

GAS tags are predefined variables you can use in a GDCAX / GJC template.

Syntax:

```
$(resource-name)
|
<tag gwc:tpl-attribute="tpl-value" [...]>...</tag>
```

Notes:

1. *resource-name* is the name of a resource defined in the Genero Application Server configuration file.
2. *tag* is an html tag.
3. *tpl-attribute* is a gwc template attribute (see the GWC manual for more information).
4. *tpl-value* is the value of the template attribute.

While most resources are defined in the Genero Application Server configuration file, pre-defined resources are provided that, while not explicitly defined in the configuration file, are available for your use. These resources include:

Pre-Defined Resource	Description
application.id	Application identifier in as.xcf
constant.meta-tags	Will be replaced by **GAS meta tags**. Only used for GWC.

server.version	Will be replaced by **GAS version**.
application.start.uri	Will be replaced by **GAS URI**. It is used in the restart template page. For example: URL is http://localhost**/cgi-bin/fglccgi/wa/r/demo?Arg=val1&Arg=val2** GAS URI will be /cgi-bin/fglccgi/wa/r/demo?Arg=val1&Arg=val2
application.querystring	Will be replaced by the **URL substring after the question mark**. For example: URL is http://localhost/cgi-bin/fglccgi/wa/r/demo?**Arg=val1&Arg=val2** The querystring will be Arg=val1&Arg=val2
connector.uri	Will be replaced by the **URL path to the connector**. Available for any Front End (GWC, GDC, or GJC). For example: URL is http://localhost**/cgi-bin/fglccgi**/wa/r/demo?Arg=val1&Arg=val2 The querystring will be /cgi-bin/fglccgi

---

GDCAX Template

The template provided by the GAS package only indicates that the GDCAX package needs to be installed. After GDCAX installation, this template is replaced by the GDCAX default template.

Example fglgdcdefault.html:

```
01 <HTML>
02   <HEAD>
03     <TITLE>
04       $(application.id) - Four J's Genero Desktop Client - Active X
05     </TITLE>
06     <META http-equiv="expires" content="1">
07     <META http-equiv="pragma" content="no-cache">
08   </HEAD>
09
10   <BODY BGCOLOR="#FFFFFF" onload="startIt();"
onbeforeunload="preventClose();">
11     <H2>
12       Application: $(application.id)
```

```
13     </H2>
14     <CENTER>
15       <OBJECT NAME="gdc"
16         Id="DesktopClient"
17         CLASSID="clsid:2311DF65-9D1A-4dda-94AA-90568D989633"
18         CODEBASE="/fjs/activex/gdc.cab#version=1,32,1,5"
19         height=440
20         width=395>
21         [Object not available! Did you forget to build and register
the server?]
22       </OBJECT>
23     </CENTER>
24   </BODY>
25
26   <SCRIPT language="javascript">
27     function startIt()
28     {
29     // the serverUrl must be set BEFORE starting the application
30     if ("$(connector.uri)" != ""){
31     gdc.setSrvUrl(location.protocol + "//" + location.host + "1
$(connector.uri)" + "/wa/r/" + "$(application.id)" + "?" +
"$(application.querystring)");
32     } else {
33     gdc.setSrvUrl(location.href);
34     }
35     gdc.setPictureUrl("$(pictures.uri)");
36     gdc.setAppName("$(application.id)");
37     return false;
38     }
39     function preventClose()
40     {
41     event.returnValue = "Genero Desktop Client";
42     }
43
44   </SCRIPT>
45
46 </HTML>
```

Notes:

1. `version` indicates the GDC ActiveX version. If this version is greater than the GDC ActiveX installed on the client computer, the client installation will be updated. In this example, `1,21,1,3` corresponds to GDCAX version 1.21.1c.
2. `startIt()` javascript function has been added to launch the application specified in `gdc.setAppName` function.
3. `preventClose()` javascript function asks the user if he really wants to leave the html page, which will close all the GDCAX applications.

Tip: To hide the GDCAX monitor, use a style.

Example:

```
01 <STYLE type="text/css">
```

```
02       .hidden { display: none; }
03 </STYLE>
04 ...



05 <OBJECT NAME="gdc" class="hidden" ...>
06 ...
07 </OBJECT>
```

In this example, the GDCAX monitor is not displayed thanks to the `hidden` style.

GJC Template

The template provided with the GAS package, only indicates that you need to install the GJC package.

Example fglgjcdefault.html:

```
01 <HTML>
02   <HEAD>
03     <TITLE>
04       $(application.id) - Four J's Genero Java Client
05     </TITLE>
06     <META http-equiv="expires" content="1">
07     <META http-equiv="pragma" content="no-cache">
08   </HEAD>
09   <BODY BGCOLOR="#FFFFFF">
10     <H2>
11       You must install the Four J's Genero Java Client extension
before you can use it with the Application Server
12     </H2>
13   </BODY>
14 </HTML>
```

After GJC installation, this template file is replaced by the GJC default template.

GJC default template:

```
01 <HTML>
02   <HEAD>
03     <TITLE>
04                   $(application.id) - Four J's Genero Java Client
05     </TITLE>
06     <META http-equiv="expires" content="1">
07     <META http-equiv="pragma" content="no-cache">
08   </HEAD>
09   <BODY BGCOLOR="#FFFFFF">
10     <H1>
```

```
11              Application: $(application.id)
12      </H1>
13      <CENTER>
14      <APPLET NAME="gjc" CODE="com.fourjs.monitor.Monitor"
ARCHIVE="gjc.jar" codebase="/fjs/applet" WIDTH=300 HEIGHT=200
MAYSCRIPT>
15        <PARAM NAME="resourcesPath" VALUE="/fjs/applet">
16        <PARAM NAME="applicationId" VALUE="$(application.id)">
17        <PARAM NAME="applicationQuerySring"
VALUE="$(application.querystring)">
18        <PARAM NAME="connectorURI" VALUE="$(connector.uri)">
19      </APPLET>
20      </CENTER>
21    </BODY>
22 </SCRIPT>
23 </HTML>
```

Adding a Web Service Application

This section will help you to set up an application rapidly. For complete details on possible configuration parameters and settings, see *Service List Reference*.

Topics

- An overview of application configuration
- Creating an abstract application
- Configuring an application for the Genero Web Services Extension
- An overview of external application configuration
- Creating an external application group
- Configuring an external application

Application Configuration Overview

When you configure an application, there are many pieces of information that must be provided to the Genero Application Server. However, much of this information is common across a set of applications. Therefore, 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 default configuration from another application, you specify it as the parent of the application.

In general, an abstract application is defined first. This abstract application is not executable, but is intended to provide the baseline default configuration for other applications to inherit. You can create as many abstract applications as you require, and abstract applications can inherit a default configuration from another abstract application.

When configuring an application that is to be an executable, you can provide the configuration details in either the Genero Application Server configuration file, or you can create a separate application-specific configuration file, known as an ***external application configuration file***. If you add the application to the Genero Application Server configuration file, you must stop and restart the Genero Application Server for the application to be recognized. If you create an external application configuration file, you can add the file into a defined GROUP directory and the application is immediately available without having to do a GAS restart.

See also:

- Quick Start - Adding New Applications
- Configuring the Genero Application Server
- Service List Reference

Creating Abstract Applications

To simplify application configuration, an application can specify a parent application to provides default configuration settings needed to run the application. An abstract application is an application that is created not as an executable, but to be a parent 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 a configuration file.

To specify an abstract application, set the **Abstract** attribute to **TRUE**.

Warning: Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.

Example for web applications:

```
<APPLICATION Id="defaultwa" Abstract="TRUE">
  <EXECUTION Using="cpn.wa.execution.local"/>
  <OUTPUT>
    <MAP Id="DUA_GWC" Allowed="FALSE"/>
    <MAP Id="DUA_GJC" Allowed="FALSE"/>
    <MAP Id="DUA_GDC" Allowed="FALSE"/>
  </OUTPUT>
</APPLICATION>
```

In this example, `DUA_GWC`, `DUA_GJC` and `DUA_GDC` are OutputMap, which indicates the Front End used to display the application. Note that no OutputMap is enabled as the attribute `Allowed` is set to `FALSE`.

Example for web services applications:

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

Configuring applications for Genero Web Services Extension

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

- your application Id
- 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

In the following example, 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.

Example:

```
01 <APPLICATION Id="calculator" Parent="ws.default">
02   <EXECUTION>
03     <PATH>$(res.path.calculator)/server</PATH>
04     <MODULE>calculatorServer.42r</MODULE>
05   </EXECUTION>
06 </APPLICATION>
```

ws.default is the parent of any web services application.

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.

Example 2:

```
01 <APPLICATION Id="echo" Parent="ws.default">
02   <EXECUTION>
03     <PATH>$(res.path.fgldir.demo)/WebServices/echo/server</PATH>
04     <MODULE>echoServer</MODULE>
05   </EXECUTION>
06 </APPLICATION>
```

Accessing the Web Service (Web Services URI information)

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 webserver (apache for example):

```
http://webserver/cgi-bin/fglccgi/ws/r/appid/service
```

Using External Application Configuration Files

To configure a Web Service application using an external application configuration file, you provide the same code that you would for adding an application directly to the GAS configuration file, except that you store this information in a file whose name matches that of the application. For example, to create a file for a Web Service named **echoServer**, you would create an external application configuration file with the name **echoServer.xcf**. You must use the **.xcf** suffix. You then place this file in a group directory as configured in the GAS configuration file.

Creating a Group

A ***group*** consists of a group name and a directory in which external application configuration files can be placed. When a front-end starts an application whose configuration information is in an external application configuration file, it must provide the group name to direct the Genero Application Server to the directory where the file resides, and the application name to identify which file to read. A group is specified in the SERVICE_LIST component of the GAS configuration file.

Syntax:

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

Notes:

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

Example:

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

Configuring an External Application

Create a separate .xcf file for each application. Because the application and configuration file share the same name, there is no need to specify the **Id** attribute. In the following example, if the file was named "echo.xcf, then this configuration file would accomplish the same task as when included in the Genero Application Server configuration file; the only difference between this example and the Example 2 shown above is the lack of the **Id** attribute.

Example:

```
01 <APPLICATION Parent="ws.default">
02   <EXECUTION>
03     <PATH>$(res.path.fgldir.demo)/WebServices/echo/server</PATH>
04     <MODULE>echoServer</MODULE>
05   </EXECUTION>
06 </APPLICATION>
```

Hot Restart of Genero Web Services

Topics

- Overview
- Principles
- Using Hot Restart of Web Services

Overview

The GWS hot restart in the Genero Application Server provides the ability to:

- Change Web Services programs without restarting Genero Application Server.
- Enhance High Availability of Genero Web Services.

Principles

The hot restart is only available for Web Services defined in external configuration files. It is not available for services defined in the GAS configuration file.

When a hot restart is issued, all current requests are finished properly, after which all new requests are serviced by the new Web service.

Using Hot Restart of Web Services

Step 1: To avoid the overriding of the previous version of the program that still serve the current requests, put the new service in a different directory

Step 2: Change the external configuration file. For example:

Original Configuration Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- echo demo service application -->
<APPLICATION Parent="ws.default">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/WebServices/echo/echoServer</PATH>
```

```
    <MODULE>echoServer</MODULE>
  </EXECUTION>
</APPLICATION>
```

New Configuration Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- echo demo service application -->
<APPLICATION Parent="ws.default">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/WebServices/newecho/echoServer</PATH>
    <MODULE>echoServer</MODULE>
  </EXECUTION>
  <TIMEOUT />
</APPLICATION>
```

In this example, the old service is in **echo** directory and the new service in **newecho** directory.

Step 3: Tell the Application Server to reload the modified configuration files:

For Linux, issue the command: **kill -1 gasdPID** or **killall -1 gasd**

Instead of number -1, you can use the **-SIGHUP** keyword.

For Windows, if the Application Server runs in a DOS console, use: **Ctrl+Break**

To use the **Ctrl+Break** on windows, in the fglprofile set the entry **fglrun.ignoreDebuggerEvent** to true, otherwise the Ctrl+Break will be sent to the DVM.

For Windows, if the Application Server runs as a service, use: **sc control *name_of_gas_service* paramchange**

For example:

```
SERVICE_NAME: fglas_2.10.01_210907122547
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING

(STOPPABLE,NOT_PAUSABLE,IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

In this example, the service name is fglas_2.10.01_210907122547.

Note:

- ***name_of_gas_service*:** You can find this name on the properties of the service or do a query with on a DOS console: **sc query | more**

What is the Genero Web Client?

This section introduces you to the Genero Web Client (GWC). The GWC is included as part of the GAS, starting with GAS 2.11.

Topics

Overview

The Genero Web Client (GWC), you can deliver true Web applications from those 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 a simple Web application to a corporate Web application, with only a few limitations. It brings to BDL applications the Internet world and the ability to be integrated in a Web site. It renders the application using technologies understood by a browser using well-known Web technologies like xHTML, XSLT, CSS and JavaScript. As a result, it can deliver the application to any device equipped with a Web browser.

Why deliver an application as a Web application?

- Web application deployment is easier and cheaper than desktop application deployment.
- The client requires a browser; no software needs to be installed on the client.

The GWC uses templates and snippet sets, written using a Genero "template language" in addition to other Web-based languages (xHTML, XSLT, JavaScript) to create dynamic web pages. Different snippet sets are provided to allow the dynamic web pages to be created using the language appropriate for the client browser. The AJAX theme snippet set is aimed at desktop browsers, while the PDA theme snippet set is aimed at handheld devices.

Rendering Engine Change with GWC 2.10

Starting with GWC 2.10, a snippet-based rendering engine replaces the legacy built-in rendering engine. This new rendering engine provides you with complete control of the customization of the user interface components, and removes many of the limitations imposed by the legacy rendering engine.

Four J's will continue to provide the legacy built-in rendering engine for those GWC deployments that are already in place, however it will no longer be enhanced. If you do not wish to migrate your application to use the snippet-based rendering engine, you can configure your applications to run in the legacy mode. The documentation in this guide is primarily for those applications that use the snippet-based rendering engine. For documentation covering the legacy built-in rendering engine, refer to the GWC 2.00 manual available on the Four J's Web site (http://www.4js.com).

Key Players

When working with applications deployed in a Web environment, you will likely 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:

Area	Player Responsibility
Application Design	Responsible for the rendering aspects of the application within GWC by adding and modifying templates and CSS to influence the look-and-feel of the application. Members of this team should be proficient with HTML and CSS.
Application Development	Responsible for the development of the Genero application, concentrating on the business logic. Members of this team should be proficient with Genero Business Development Language.
Advanced Production	Responsible for the additional functionality and navigation added to an application through the use of the template language to link BDL form objects and JavaScript to define the behavior. Members of this team should be proficient with HTML and JavaScript.
Deployment and Infrastructure	Responsible for the complete GWC 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.

Having some knowledge of Web technologies like HTML, XML, style sheets and JavaScript ease GWC understanding. You can find Web standards at http://www.w3.org. Take a look at http://www.w3schools.com tutorials to get a quick start.

Limitations

For the most part, a Genero application reacts the same across the different Front End clients (GWC, GDC, and so on). There are, however, limitations for applications being rendered by the GWC 2.10 snippet-based rendering engine:

- Local actions (such as Copy, Paste, and Cut) are not supported.
- MDI (Multi Document Interface) is not supported. For more information on MDI Windows, refer to the *Genero Business Development Language Manual*.
- With multiple windows, in PAGE MODE (Set 2) and PDA MODE (Set 3), only the active window is displayed.
- With multiple windows, in AJAX MODE (Set1), only the one modal dialog (STYLE='dialog') is displayed as a pop-up. The first normal window displays as the page background.
- Old widgets are not supported
- DialogTouched is not supported
- FrontCalls and Front End Functions are not fully supported by the GWC. Often this is due to permissions; the browser does not have the permissions necessary to execute the front end function. For more information on Front End Functions, refer to the *Genero Business Development Language Manual*.

Accelerators

The GWC will recognize the first accelerator defined for an action (acceleratorName). It will not recognize those accelerators defined by acceleratorName2, acceleratorName3, or acceleratorName4.

You can set the accelarator key combination to any combination you wish. There are, however, a set of keys that you should avoid using, as there may be unintended side effects. This set includes:

- Tab
- Shift-Tab
- Down
- Up
- Next
- Prior
- Home
- End

Adding Applications

This section will help you to set up an application rapidly. For complete details on possible configuration parameters and settings, see *Application List Reference*.

Topics

- An overview of application configuration
- Creating an abstract application
- Configuring an application for the Genero Web Client
- An overview of external application configuration
- Creating an external application group
- Configuring an external application
- What if the applicaiton doesn't start?

Application Configuration Overview

When you configure an application, there are many pieces of information that must be provided to the Genero Application Server. However, much of this information is common across a set of applications. Therefore, 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 default configuration from another application, you specify it as the parent of the application.

In general, an abstract application is defined first. This abstract application is not executable, but is intended to provide the baseline default configuration for other applications to inherit. You can create as many abstract applications as you require, and abstract applications can inherit a default configuration from another abstract application.

When configuring an application that is to be an executable, you can provide the configuration details in either the Genero Application Server configuration file, or you can create a separate application-specific configuration file, known as an ***external application configuration file***. If you add the application to the Genero Application Server configuration file, you must stop and restart the Genero Application Server for the application to be recognized. If you create an external application configuration file, you can add the file into a defined GROUP directory and the application is immediately available without having to do a GAS restart.

See also:

- Quick Start - Adding New Applications
- Configuring the Genero Application Server
- Application List Reference

Creating Abstract Applications

To simplify application configuration, an application can specify a parent application to provides default configuration settings needed to run the application. An abstract application is an application that is created not as an executable, but to be a parent 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 a configuration file.

To specify an abstract application, set the **Abstract** attribute to **TRUE**.

Warning: Abstract applications can only be defined in the application server configuration file, they cannot be defined in an external application configuration file.

Example for web applications:

```
<APPLICATION Id="defaultwa" Abstract="TRUE">
  <EXECUTION Using="cpn.wa.execution.local"/>
  <OUTPUT>
    <MAP Id="DUA_GWC" Allowed="FALSE"/>
    <MAP Id="DUA_GJC" Allowed="FALSE"/>
    <MAP Id="DUA_GDC" Allowed="FALSE"/>
  </OUTPUT>
</APPLICATION>
```

In this example, `DUA_GWC`, `DUA_GJC` and `DUA_GDC` are OutputMap, which indicates the Front End used to display the application. Note that no OutputMap is enabled as the attribute `Allowed` is set to `FALSE`.

Example for a GWC application:

```
<!--This is the default application for GWC-->
<APPLICATION Id="defaultgwc" Parent="defaultwa" Abstract="TRUE">
 <TIMEOUT Using="cpn.gwc.timeout.set1"/>
 <PICTURE Using="cpn.gwc.picture"/>
 <OUTPUT Rule="UseGWC">
  <MAP Id="DUA_Symbol-WC" Allowed="TRUE">
   <RENDERING Using="cpn.rendering.xslt"/>
   <THEME Using="cpn.theme.default.gwc">
    <TEMPLATE Id="_default">$(res.theme.symbol-
wc.stylesheet)</TEMPLATE>
   </THEME>
  </MAP>
  <MAP Id="DUA_GWC" Allowed="TRUE">
   <RENDERING Using="cpn.rendering.gwc"/>
   <THEME Using="cpn.theme.default.gwc"/>
  </MAP>
```

```
16   <MAP Id="DUA_AJAX" Allowed="TRUE">
17    <RENDERING Using="cpn.rendering.gwc2" />
18    <THEME Using="cpn.theme.ajax.gwc" />
19   </MAP>
20   <MAP Id="DUA_PAGE" Allowed="TRUE">
21    <RENDERING Using="cpn.rendering.gwc2" />
22    <THEME Using="cpn.theme.page.gwc" />
23   </MAP>
24   <MAP Id="DUA_PDA" Allowed="TRUE">
25    <RENDERING Using="cpn.rendering.gwc2" />
26    <THEME Using="cpn.theme.pda.gwc" />
27   </MAP>
28  </OUTPUT>
29 </APPLICATION>
```

Configuring applications for Genero Web Client (GWC)

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

- your application Id
- the parent application where the main configuration is set (in this example, defaultgwc)
- the path to your compiled files
- the main module to launch

In the following example the path is a resource; this can also be an absolute path to your application files.

Example:

```
01 <APPLICATION Id="demo" Parent="defaultgwc">
02   <EXECUTION>
03     <PATH>$(res.path.fgldir.demo)</PATH>
04    <MODULE>demo.42r</MODULE>
05   </EXECUTION>
06 </APPLICATION>
```

defaultgwc is the parent of any web application for GWC.

Example:

```
01 <APPLICATION Id="defaultgwc" Parent="defaultwa" Abstract="TRUE">
02   <TIMEOUT Using="cpn.gwc.timeout.set1"/>
03   <PICTURE Using="cpn.picture.webserver"/>
04   <OUTPUT Rule="UseGWC">
05     <MAP Id="DUA_GWC" Allowed="TRUE">
06       <THEME Using="cpn.theme.default.gwc"/>
```

```
07     </MAP>
08   </OUTPUT>
09 </APPLICATION>
```

The **defaultgwc** application inherits from the **defaultwa** application. In the Abstract application section, **defaultwa** is the parent for any web application and no OutputMap is enabled; **defaultgwc** just enables `DUA_GWC` OutputMap and specifies the environment for this OutputMap.

Using External Application Configuration Files

To configure an application using an external application configuration file, you provide the same code that you would for adding an application directly to the Genero Application Server configuration file, except that you store this information in a file whose name matches that of the application. For example, to create a file for a program named **demo**, you would create an external application configuration file with the name **demo.xcf**. You must use the **.xcf** suffix. You then place this file in a group directory as configured in the Genero Application Server configuration file.

Creating a Group

A ***group*** consists of a group name and a directory in which external application configuration files can be placed. When a front-end starts an application whose configuration information is in an external application configuration file, it must provide the group name to direct the Genero Application Server to the directory where the file resides, and the application name to identify which file to read. As with applications, a group is specified in the APPLICATION_LIST component of the Genero Application Server configuration file.

Syntax:

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

Notes:

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

Example:

```
01 <GROUP Id="_default">$(res.path.app)</GROUP>
02 <GROUP Id="gwc-demo">$(res.path.app)/tutorial/gwc-demo</GROUP>
```

Configuring an External Application

Create a separate .xcf file for each application. Because the application and configuration file share the same name, there is no need to specify the **Id** attribute. In the following example, if the file was named "demo.xcf", then this configuration file would accomplish the same task as when included in the Genero Application Server configuration file; the only difference between this example and the example shown above is the lack of the **Id** attribute.

Example:

```
01 <APPLICATION Parent="defaultgwc">
02   <EXECUTION>
03     <PATH>$(res.path.fgldir.demo)</PATH>
04    <MODULE>demo.42r</MODULE>
05   </EXECUTION>
06 </APPLICATION>
```

What if the application doesn't start?

When you request an application, if it does not start, then chances are there is something wrong in the configuration. Some suggestions:

- Check your environment variables in $FGLASDIR/etc/as.xcf.
- The log files in $FGLASDIR/log may have some details about the error messages.
- If you access the application through a web server, ensure that your connector.xcf is correctly configured.

When you receive the "Error: Runtime error. Try again ..." page

Your application cannot start. Check your application configuration.

Usually, the path to your program is not the correct one.

How Browser-Based Themes, Templates, and Snippet Sets work for the GWC

When the GAS displays an application with the GWC, it detects the type of browser being used to display the application and applies the appropriate theme to render the application correctly for that browser type. This section discusses the use of themes, maps, templates, and snippet sets and the browser types they support.

Topics

Themes, Templates, and Snippet Sets

"Snippet sets are adapted to the different classes of Web browsers."

When the GWC displays an application in a browser, it identifies the browser type or class and, based on that browser type, uses a theme (a specific set of template and snippet files) to render the application. The Genero Web Client installs with five pre-defined snippet sets (described below); you can customize those snippet files or you can create your own theme and snippet set. After determining the browser type, the GWC uses the information provided by the auda.xrd file to select the theme to use.

Within the GAS configuration file, the association is made between the THEME, a MAP, and the TEMPLATE and SNIPPET files.

See also:

- How the GWC uses web technologies to deliver an application
- Customizing the User Interface with templates and snippets

Default Themes (Snippet Sets)

The GAS installs with default snippet sets.

AJAX (and AJAX_HTML) Theme

The AJAX theme is based on a JavaScript framework. It provides your applications with a Genero experience that closely resembles the Genero Desktop Client experience, but provides it in a Web 2.0 environment. The AJAX theme should work with all desktop

browsers. Applications displayed in a FireFox browser should display as expected. Applications displayed in Internet Explorer (IE6 and IE7) are largely functional, however be aware that some layout problems exist. Safari and Opera support is currently at an early stage.

Note: Starting with 2.11, there is a new output map DUA_AJAX_HTML specifically made for IE special features like Canvas. This theme has the same functionality as the DUA_AJAX theme has for the other browsers.

PDA Theme

The PDA theme does not require JavaScript. This theme should be used for PDA and SmartPhone browsing. It is tested on a Windows Mobile 5 PDA.

The PDA theme is based on pages exchanged with the Web server. As a result, some Genero features cannot be rendered as they are on a Desktop client (autonext, picture, triggers execution, focus-based actions, and so on). In some cases, your applications might behave differently than it would if delivered by a Desktop client. We have set up a server side algorithm to support legacy applications as close as possible to the Desktop clients and we will continue to improve this support.

The PDA theme also modifies the layout of your application to fit on a PDA screen. For example, tables and HBoxes are "verticalized". A table does not display as a traditional table; the PDA theme displays each column on a separate line. This is a designed behavior, created because most PDA devices do not have screens wide enough to handle a traditional table layout.

PAGE (and PAGE_HTML) Themes

The PAGE theme does not require JavaScript and offers a Web 1.0 non-intrusive (no CSS) rendering for your application. This theme is based on pages exchanged with the Web server (like the PDA theme). Examining the PAGE theme and its snippet set is the best way to understand our new rendering engine principles. Feel free to play with it and modify it.

Note: Starting with 2.11, there is a new output map DUA_PAGE_HTML specifically made for IE special features like Canvas. This theme has the same functionality as the DUA_PAGE theme has for the other browsers.

To see an application rendered by each of these themes, launch the Demos application.

GWC Template

GWC template uses the GWC instructions. The instructions are tag attributes prefixed by the namespace **gwc**.

Example:

```
01 <?xml version="1.0"?>
02 <html xmlns:gwc="http://www.4js.com/GWC"
xmlns="http://www.w3.org/1999/xhtml"
xmlns:svg="http://www.w3.org/2000/svg">
03 <head>
04   <title gwc:condition="application/ui"
05          gwc:marks="title [application/CID, (application/ui ?
application/ui/text : '')
06          + (application/ui/window ? ' - ' +
application/ui/window/name : '')]">Genero Web Client</title>
07 ...
08 </head>
09
10 <body ... >
11   ...
12   <!-- Application ending -->
13   <div gwc:condition="application/state/ended" class="gEnding"
gwc:content="includeSnippet('EndingPage')" />
14   ...
15   <!-- first the topmost normal window -->
16   <div gwc:condition="application/ui/topmostNormalWindow"
gwc:content="application/ui/topmostNormalWindow" />
17   ...
18 </body>
19 ...
20 </html>
```

Notes:

1. On lines 04-06, the page title will be replaced by the application title.
2. On line 16, the `<div ... gwc:content="application/ui/topmostNormalWindow" />` section will add the application window.

How the GWC uses Web Technologies (to deliver an application)

The Genero Web Client allows developers to create applications using the Genero Business Development Language (Genero BDL) and deliver the applications as web applications. To deliver a Genero application as a web application, the Genero Web Client must render the application as an xHTML-based application.

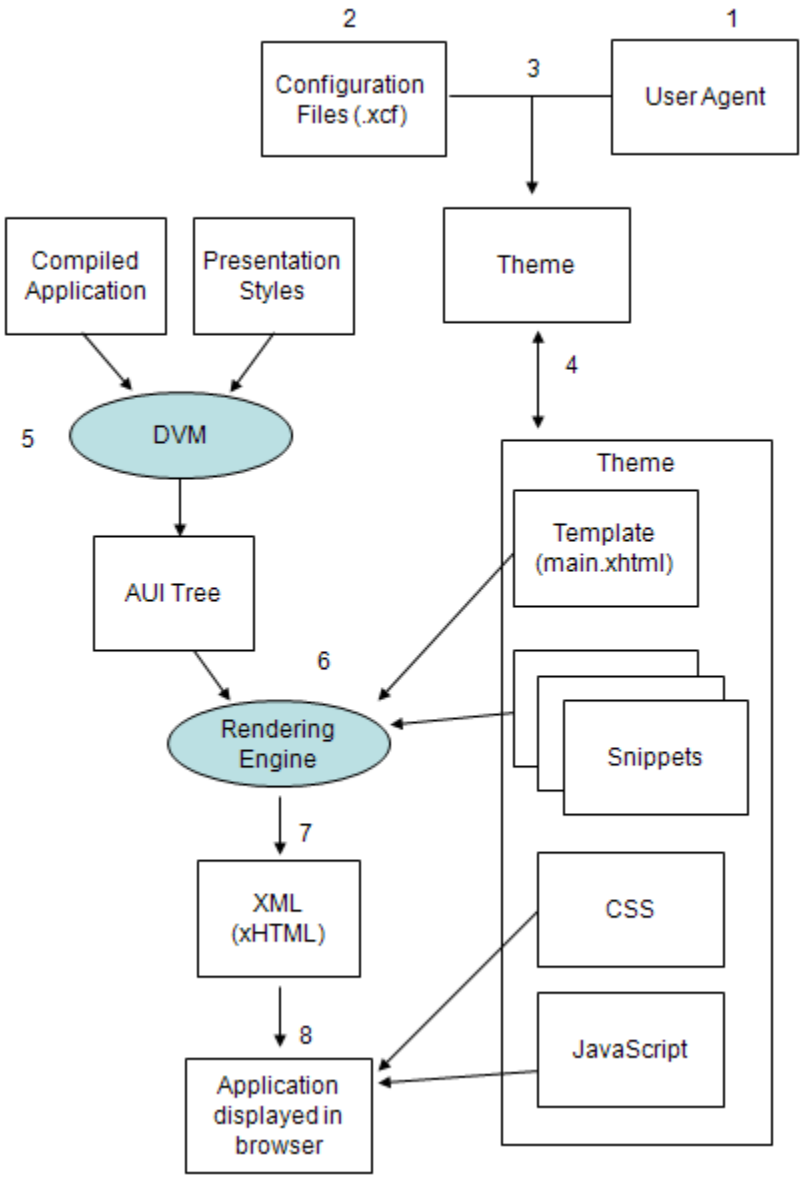
This section presents the main concepts that drive a Genero Web Client project. Genero Web Client uses a template for rendering. There are four main parts to Genero Web Client rendering: generated HTML (core), CSS (look), JavaScript (widgets shaping and behavior), and template language.

Topics

- How an application is rendered by the GWC
- Templates
- Cascading Style Sheets
- JavaScript
- Template Language

How an Application is Rendered by the GWC

The following provides a general explanation of the steps taken when the Genero Web Client delivers an application, starting with the user entering in the URL and ending with the application displayed in the user agent (browser).



The end user requests the application by entering the appropriate URI in the user agent (1).

The request for the application is routed to a Genero Application Server. If a Web server is involved, the Connector configuration file routes the application to the correct application server. The Genero Application Server's configuration files contain the information about which themes are to be used for the requested application. (2)

Unless explicitly specified by the URI or the application configuration, the appropriate theme is selected based on the type of user agent (desktop browser, PDA, and so on). The auda.xrd file is responsible for selecting the appropriate theme for a specific type of user agent. (3)

In the GAS configuration file, a theme is defined by a TEMPLATE and a series of SNIPPETS. The template file in turn provides the references for any JavaScript and CSS files. (4)

The GAS, meanwhile, has started a DVM to serve the application. The DVM creates the AUI tree (Abstract User Interface describing components and behaviours), which is sent to the Genero Application Server. (5)

The Genero Web Client's snippet-based rendering engine uses the AUI tree (provided by the DVM) and the template and snippet files (specified by the application's configuration in the GAS configuraiton files) to create an XML / xHTML document that is passed to the user agent. (6, 7)

Any JavaScript and CSS is applied to the xHMTL file by the user agent before displaying the application to the user. (8)

Template

A template is an xHTML file that displays your application through a browser, using a Genero front end. A template defines how and where your application is displayed inside a HTML page. Genero Web Client has a default template showing the current application window.

With the old rendering engine, we focused on layout change based on container selections. We need to ensure to change that to concentrate on speaking of snippets based rendering offering complete flexibility around HTML generation.

The following template example is an excerpt of the default template file provided at installation for the DUA_AJAX output map, $FGLASDIR/tpl/set1/main.xhtml.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Copyright 2006-2008 Four J's Development Tools. All Rights
Reserved. -->
<html xmlns:gwc="http://www.4js.com/GWC"
xmlns="http://www.w3.org/1999/xhtml"
xmlns:svg="http://www.w3.org/2000/svg">
<gwc:dummy
  gwc:omit-tag="true"
  gwc:define="
  tplImages application/connectorURI+'/fjs/set1/img/';
  ...
"
>
<head>
  <title gwc:condition="application/ui" gwc:marks="title
[application/CID, (application/ui ? application/ui/text : '')
        + (application/ui/window ? ' - '
        + application/ui/window/name : '')]">Genero Web Client</title>
```

```
  <meta http-equiv="Content-Type" gwc:attributes="content
XPathConfig('/APPLICATION/OUTPUT/RENDERING/MIME_TYPE/text()')+';
charset='+document/encoding" />
  <script type="text/javascript"> var gwc = { initTime:new
Date().getTime(), cfg:{} } </script>
  <script type="text/javascript">
    gwc.cfg.development = <span gwc:replace="server/development"/>;
    ...
  </script>
  <script type="text/javascript" gwc:attributes="src
application/connectorURI+'/fjs/gwccomponents.js'" defer="defer">
</script>
  <script type="text/javascript" gwc:attributes="src
application/connectorURI+'/fjs/gwccore.js'" defer="defer"> </script>
  <link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/gwccomponents.css'" type="text/css"
title="Default Theme"/>
  <link rel="shortcut icon" gwc:attributes="href
application/connectorURI+'/favicon.ico'" type="image/x-icon" />
  ...
  <style type="text/css" id="gStyleList"/>
</head>

<body
  gwc:marks="
    _href document/URL;
    launch
application/meta/launch?[application/CID,application/meta/launch/url+'?
t=']:null;
    processing application/state/processing &amp;&amp;
noop(document/URL)?[application/CID]:null;
    interactive application/state/interactive &amp;&amp;
noop(document/URL)?[application/CID]:null;
    end application/state/ended/normal?[application/CID]:null;
    style application/ui ? [application/CID, application/ui/StyleList]
: null;
    "
    gwc:attributes="
    class (!isIe6 ? 'gHasContentBoxModel' : '') + (isIe6 ?
'gHasBorderBoxModel' : '');
    "
>
  <!--noscript><br/><br/> Your browser must support
javascript.</noscript-->
  <input type="hidden" id="gSuaURL" gwc:attributes="value
document/URL"/>
  ...
  <!-- Application ending -->
  <div gwc:condition="application/state/ended" class="gEnding"
gwc:content="includeSnippet('EndingPage')" />

  <gwc:dummy gwc:omit-tag="true" gwc:condition="application/ui">
  <!-- Application user interface -->

  <!-- first the topmost normal window -->
  <div gwc:condition="application/ui/topmostNormalWindow"
gwc:content="application/ui/topmostNormalWindow" />
```

```
  <!-- second, each modal window -->
  <div
    gwc:condition="application/ui/window &amp;&amp; (
!application/ui/topmostNormalWindow ||
application/ui/topmostNormalWindow &amp;&amp; application/ui/window
&amp;&amp; application/ui/topmostNormalWindow/name !=
application/ui/window/name )"
    gwc:content="application/ui/window"
  />

  <div gwc:condition="application/ui/StartMenu"
gwc:replace="application/ui/StartMenu" />
  </gwc:dummy>

  <!-- File Transfer -->
  <div class="gFiles"
gwc:condition="application/meta/fileTransfer/currentFiles/length"
gwc:content="includeSnippet('FileTransfer')"/>

  <!-- Errors -->
  <fieldset style="float:left; clear:both"
gwc:condition="document/errors">
    <legend>Errors</legend>
    <pre gwc:content="document/errors" />
  </fieldset>
</body>
</gwc:dummy>

</html>
```

Cascading Style Sheet

Cascading style sheets (CSS) are used to format xHTML pages. If you are not familiar with this technology, please refer to the W3C web site at www.w3.org.

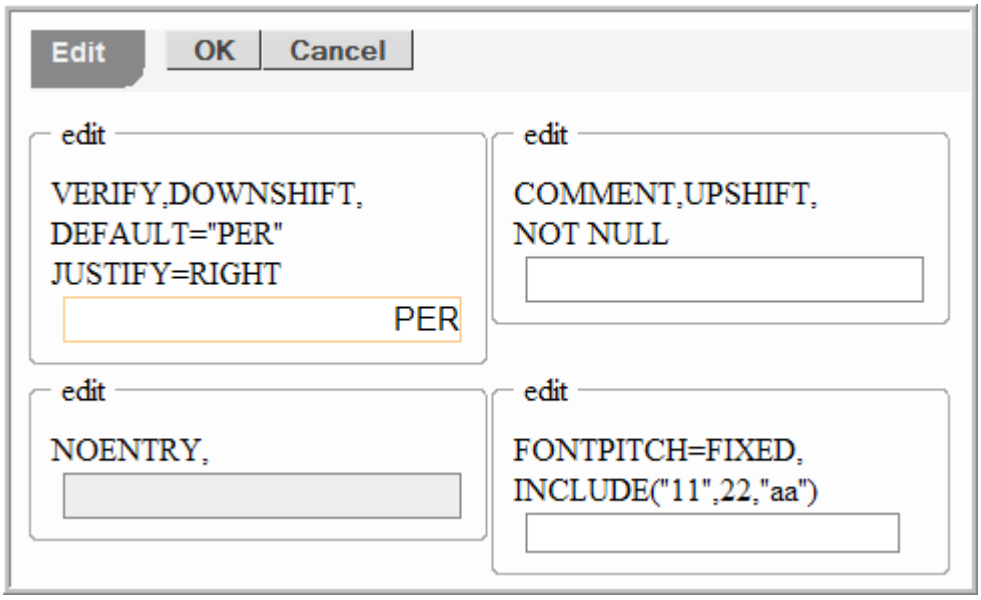
Genero Web Client uses CSS to place and shape widgets. The menu buttons are displayed flat, thanks to a CSS style:

Excerpt from gwccomponents.css:

```
01 .gDialog,
02 .gMenu {
03   float: left;
04   clear: both;
05   width: 95%;
06   padding: 0px 16px 2px 0px;
07   margin: 0px 2px 10px 2px;
08   -moz-border-radius-bottomright: 16px;
09   -moz-border-radius-topright: 16px;
10   background-color: #F4F4F4;
11   }
```

Any INPUT inside an element of the class menu has no border.

The widget shaping includes widget states. The following screenshot illustrates noEntry and other states for input fields:



Thanks to the snippet concept, developers can add any kind of CSS class to their generated HTML code - you are not limited to the CSS provided by default.

JavaScript

GWC is able to interact with the browser using the browser framework (HTTP, HTML Forms, and so on) named Page Mode. GWC will provide the browser a complete xHTML page describing the current 4GL application state. JavaScript could be used, in this case, on the browser side for any purpose such as a calendar pop-up or input control as picture, format, and so on.

GWC is also able to interact with the browser using a JavaScript framework. In this case, the GWC provides to this JavaScript framework incremental HTML modifications (according to the template and template snippets) and some dialog structures directly issued from the DVM. The JavaScript framework can then have a more 4GL-like behavior.

We provide a template/snippets customization set with a complete JavaScript FrameWork. You will be able to customize widgets dynamic behaviour by modifying JavaScript hooks, but in reality main modifications can (and should) be done mostly on snippets and thru CSS.

Template Language

The template language has been introduced to integrate web-designed pages and extend generated HTML capabilities. The selection criteria are:

- web design tools friendly (does not disturb a web page layout)
- powerful enough

This language is used in templates and snippets. It is interpreted by the Genero Web Client engine which generates new HTML code. You can perform instructions ranging from simple condition tests to loop on table lines.

There are three types of items: template instructions, template expressions, and template paths.

Template instruction

A template instruction is a predefined attribute added to a HTML tag. It defines how the HTML tag is interpreted.

Syntax

```
<tag gwc:instruction="expression" ... > ... </tag>
```

Example

```
01 <div gwc:replace="window"></div>
```

The template instruction is `gwc:replace`. This is an instruction to replace the `<div>` tag with the HTML code for the current window.

Template expression

A **template expression** is the template instruction value. It can be a string, an operation, or a conversion function.

Example

```
01 orientation=='horizontal'?'':' gRadioGroupVertical'
```

This conditional expression returns an empty string if the value of orientation path is horizontal otherwise it returns gRadioGroupVertical.

Template path

The **template path** is used to access an element of the current application. The element can range from the entire application window to a field value in a table.

Example

```
01 <title gwc:content="application/ui/window/text">Title of the
page</title>
```

The template path is `application/ui/window/text`, returning the title of the window.

For more information

- For more information on the template language, refer to the *Template Language Reference* section.

Genero Web Client Application Directory Structure

When developing and testing applications with the Genero Web Client, you typically conduct all your development using the Genero Application Server and not the Web server. For production, when including the Web server, certain files need to be relocated to the Web server host.

- Development application directory structure (application server only)
- Production application directory structure

For a list of the files and directories created during installation, see the Installation - Directory and Files help topic.

Development Application Directory Structure

For any Genero application, you have to manage multiple files: source code files, their compiled versions, and other files associated with the DVM (.4ad, .4st, and so on). When you display a Genero application using the Genero Web Client, additional files must be managed: templates, snippets, CSS, and JavaScript files.

Four J's recommends that you organize the directory to represent a small local web site, as shown in the example below.

Example

- myApp: root directory of your application
  - src: 4GL programs, per files, and so on
  - web: root directory of the local application web site
    - img: images required by the application
    - inc: JavaScript and CSS files
    - html: application-specific template and snippet files

This example provides a simple directory structure for organizing files for a single application. The directory structure and organization can get more complex when you are managing several modules or applications and/or a larger web site, especially when you re-use files across applications.

You should then define an ALIAS that points to the root directory of the local application Web site, as shown in the example below.

Example

```
<INTERFACE_TO_CONNECTOR>
```

```
   ...
   <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
   <ALIAS Id="/mysite">/myApp/web</ALIAS>
   ...
</INTERFACE_TO_CONNECTOR>
```

In this example, the alias "/mysite" enables access to the files stored in the directory "/myApp/web/". Using the alias, you can connect to the local web site using the URI `http://<app_server>:6394/mysite/page.html`, where page.html is in the myApp/web directory.

For deployment, you can copy the entire web site to your Web server, or set a special directory to gather the modified templates. This makes the template configuration in the as.xcf easier.

To avoid breaking links, build the web site with absolute paths. For example, specify "/mysite/img/pic.png" not "../img/pic.png".

Where to Place Files for Production

When it is time to put your application in production, you will introduce the Web server to your solution and may want to move some of the files to the Web server. It is not necessary to move the files to the Web server, however. In fact, Four J's recommends you keep all files on the application server and move them to the Web server only if performance issues arise.

Topics

- Files that can exist on either the application server or the Web server
- Keeping files on the application server
- Moving files to the Web server
- Maintaining files on both the application server and Web server.

Files that can exist on either the application server or the Web server

The following files can exist on either the application server or the Web server:

- CSS
- JavaScript
- Images
- Documents (html, MS Word, PDF, and so on)

Note: Template and snippet files must be located on the application server, as they are used to create the XML (xHTML) file that is passed to the browser.

Keeping files on the application server

If you plan to leave the files on the application server only, you should not have to do any alterations to the existing files with the possible exception of the template files. In order for a Web server to access files stored on the application server, `$(connector.uri)` must be used in front of the directory path for the files listed above. At runtime, `$(connector.uri)` is replaced by:

- Nothing when you access the application by connecting directly to the application server.
- `/cgi-bin/fglccgi/<session>` when you access via a Web server.

To summarize, `$(connector.uri)` is required to retrieve documents sitting on the application server when going through a Web server.

Moving files to the Web server

When moving files from the application server to the Web server, you must:

- Ensure you have defined the same aliases for the Web server as you have declared in the application server configuration file. Refer to the `<ALIAS>` section in your application configuration file.
- Move the CSS, JavaScript, and documents to the correct place on the Web server (in the directories specified by the aliases) .
- Move template images. Template images are those used by the template, not by your application (background images, logos, and so on).
- Whether you move application images depends on whether the image extension is referenced in your application.
    - If you haven't specified the image extension in your application, the Genero Application Server will automatically make the extension resolution for you (by searching for .png, .gif, and so on). Such images need to remain on the application server. Otherwise, the html page will search for an image without extension on your Web server and won't find it.
    - If you have specified the extension in your program, you can move the image to the Web server.
    - Remove `$(connector.uri)` from your `<PICTURE>` path in **as.xcf**.
    - In the template file, remove reference to `$(connector.uri)` where the path references files now stored on the Web server.

Locating Files on both the Application Server and the Web Server

If you wish to allow access from both the Web server or the application server simultaneously, you should:

- Put the files on both the application server and the Web server.
- Remove references to `$(connector.uri)`.

By removing the `$(connector.uri)`, as long as the files exist on both the application server and the Web server, you can access the files regardless of the connection type used.

Tips:

- If you do not have the necessary files on the Web server and you remove `$(connector.uri)`, the Web server will not be able to access the files on the application server files.
- You may decide to leave a subset of the files on the application server and not place them on the Web server. For example, you may decide to leave **gwccomponents.css** on the application server, and only copy your custom CSS files to the Web server. In this situation, you would leave `$(connector.uri)` in the path to **gwccomponents.css** in your template.

Session Variables and Cookies

Topics

- What are Session Variables and Cookies
- Why Session Variables? Use Cases
- Working with Session Variables and Cookies
- Building HTTP Cookies in the configuration file
- Setting session variables with front-end functions
- Setting session variables with a snippet-based rendering engine function
- Setting session variables with a Client Side Framework function
- Session variables template path

Session Variables and Cookies

A **session variable** is a name-value pair maintained for the duration of the session by the GAS. A session variable is accessible from the user agent, the application, and the GWC snippet-based rendering engine.

In order to make a session variable persistent between two runs of an application, you must store the session variable in an HTTP cookie. The **cookie** is a text-only string containing the name-value pair and is saved into the memory of your browser. The cookie is then sent back to the GAS on all future requests for the application.

Why Session Variables? Use Cases

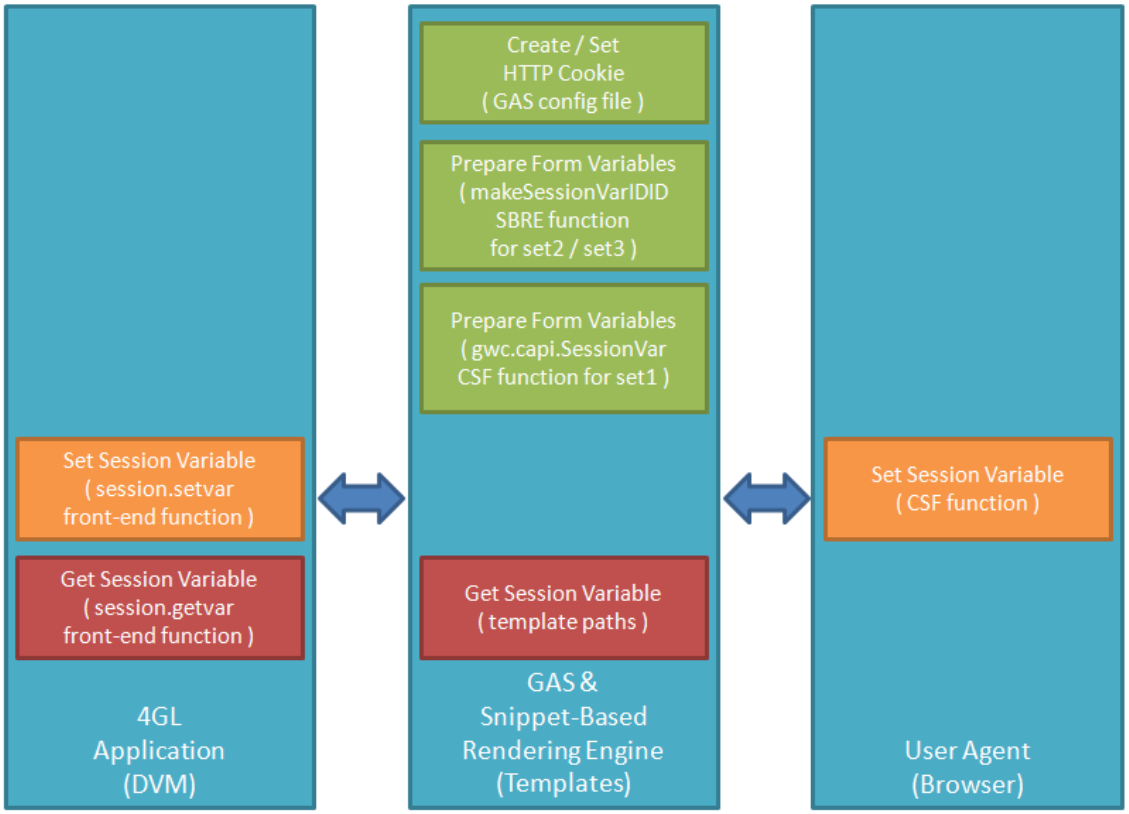
There are a variety of reasons why you would want to introduce a session variable. For example:

- You can have an application set a session variable and use it inside a template snippet, without having to design a form field or a static label in the form.
- You can place a hidden field in an HTML form that will be sent along with other form data (and fetch them from the application) without having to design a form field in the 4GL form.
- From the client-side front-end (CSF), you can use a session variable to hold data that can be used from inside a template snippet to keep a state between two page updates.
- From the browser, you can save the session variable as an HTTP cookie to hold data that can be retrieved between two runs of an application by the same user. For example, to store the users preferences.

Working with Session Variables and Cookies

The way to share session variables between two runs of an application is to persist the session variable means of HTTP cookies. Therefore, Genero must provide a way to set and get session variables to and from cookies.

Create, Set, and Get Session Variables and HTTP Cookies



A session variable or cookie can be created and updated in the following ways:

- From the GAS, an administrator can create and set HTTP Cookies in the configuration file.
- From the application, a 4GL developer can set or get session variables through front-end functions.
- From the templates (snippet-based rendering engine), a designer can prepare form variables with a snippet-based rendering engine function or CSF function and can access session variables through template paths.
- From the User Agent (browser), a request can set session variables through form variables. A session variable can be created using a Client Side Framework function.

Session Variable / HTTP Cookie synchronization

Every session variable may not need to be put into an HTTP cookie; conversely every HTTP cookie value need not be put into a session variable.

The idea is to tag each session variable that has to be set into a cookie. The snippet-based rendering engine (with the GAS) would be in charge of maintaining the cookies/session variables synchronization. On incoming HTTP requests, the engine would update corresponding session variables; on outgoing HTTP responses, the engine would set changed cookies. Between the request and the response, the 4GL developer can update session variables. Therefore, updating an HTTP cookie is indirectly done by updating its session variables.

Build HTTP Cookies in the Configuration File

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.

You declare cookies for an application within the configuration file.

Examples of cookie declarations

```
<!-- session cookie -->
<HTTP_COOKIE Id="cookie1">
  <VARIABLE Id="var1" />
  <VARIABLE Id="var2" />
  <VARIABLE Id="var3" />
  <VARIABLE Id="var4" />
</HTTP_COOKIE>
<!-- persistent cookie that applies only to this application -->
<HTTP_COOKIE Id="cookie2" Expires="date" Domain="domain">
  <VARIABLE Id="var5" />
  <VARIABLE Id="var6" />
</HTTP_COOKIE>
<!-- secure persistent cookie with default variable value and constant
value -->
<HTTP_COOKIE Id="cookie3" Expires="date" Domain="domain" Secure="TRUE"
HttpOnly="TRUE">
  <VARIABLE Id="var7" />
  <VARIABLE Id="var8">Initial value</VARIABLE>
  <CONSTANT Id="constant1">A value</CONSTANT>
</HTTP_COOKIE>
```

Note: All cookies and all associated session variables will be shared between all applications.

Setting session variables with front-end functions

A set of front-end functions allows you to dynamically set and get session variables from within your Genero application. These front-end functions are:

- **session.getvar(*varname*):** Return the value of session variable called ***varname***. Return an empty string if ***varname*** doesn't exist.
- **session.setvar(*varname* , *value*):** Set value ***value*** to session variable called ***varname*** and return 1 if successful, 0 otherwise.

  Note: Setting a variable to an empty string is equivalent to deleting the variable.

Setting session variables with a snippet-based rendering engine function

To create a session variable (when there is no existing function with the same name), a snippet-based rendering engine function is provided.

```
makeSessionVarIDID(varName, varValue)
```

In the template itself, you would add this function using a line such as the following:

```
<input type="hidden" gwc:attributes="name
makeSessionVarIDID('var1','value1')" />
```

When you use this function, the GAS first builds a page with a session variable, however it is not created in the application context yet. When the form is submitted by the User Agent to the GAS, the engine creates the session variable.

Note: Setting a variable to an empty string is equivalent to deleting the variable.

Setting session variables with Client Side Framework API

To create a session variable, a CSF function is provided.

```
gwc.capi.SessionVar(varName, varValue)
```

In the template itself, you would add this function using a line such as the following:

```
<input type="button" onclick="gwc.capi.SessionVar('var1','value1')" />
```

When you use this function, the CSF submits the variable to the GAS. The session variable is immediately created in the application context.

Note: Setting a variable to an empty string is equivalent to deleting the variable.

Access session variables using template paths

Session variables are held by a collection. As a result, there is a collection path to iterate through the full collection of variables, as well as a selector path to access a session variable by name.

```
application/meta/variables
application/meta/variable[<name>]
application/meta/variable/XDID
application/meta/variable/name
application/meta/variable/value
application/meta/variable/readOnly
```

See also: The ApplicationMetaInformation object and The Variables object (part of Template Paths - Application Hierarchy)

File Transfer within the GWC

When working with applications, there are security issues involved when retrieving files from or sending files to the DVM host. This is especially true for applications being delivered via a Web browser. When using the fgl_putfile() and fgl_getfile() methods, the user will be prompted to select where the file is placed on the local device (fgl_putfile) or which file to upload (fgl_getfile).

Warning! The implementation of file transfer relies on the snippet-based rendering engine first introduced with GWC 2.10. For more information, see *Application Rendering*.

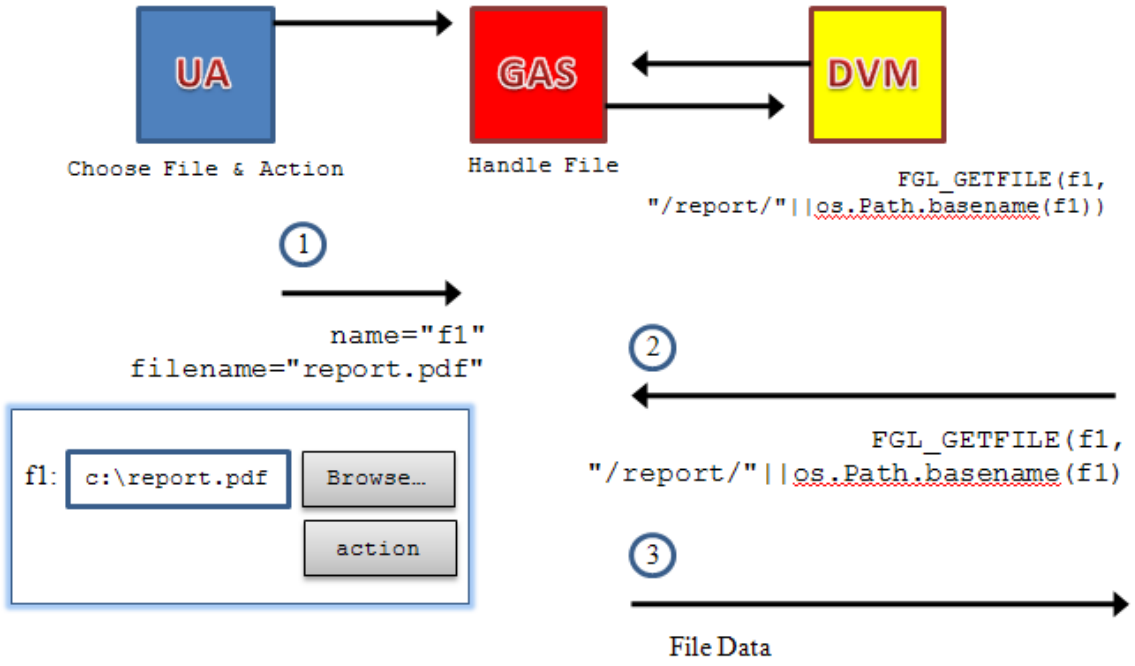
Topics

File Transfer: Uploading a File

In a Genero application, uploading a file from the front end to the application server host is handled by the fgl_getfile() built-in function. For details about this function, refer to the *Built-In Functions* topic in the *Genero Business Development Language Manual*.

Uploading a file to the DVM

The next figure illustrates the process of uploading a file to the DVM.



Notes:

1. A form definition file defines the EDIT field with the STYLE="FileUpload" attribute. The field displays with a Browse button that allows the user to locate the file. Once the file is located and selected by the user, it is transferred to the application server. The application server stores the file in the directory specified by the TEMPORARY_DIRECTORY element.
2. The user executes the action that results in a call to the FGL_GETFILE() built-in function. It requests the file from the application server.
3. The file data is transferred from the application server and saved in the directory specified by the second FGL_GETFILE() parameter.

Preparing to deploy with GWC

When using the GWC to deploy an application that includes uploading a file, verify the following:

- FileUpload.xhtml snippet exists
- Form definition file specifies FileUpload style
- Template form method set correctly

FileUpload.xhtml snippet

Uploading a file with GWC requires that the mode (snippet set) include a FileUpload.xhtml snippet. For example, in the as.xcf:

```
  <SNIPPET Id="Edit"
Style="FileUpload">$(res.path.tpl.ajax)/FileUpload.xhtml</SNIPPET>
```

Form Definition File modifications

When the application displays a form, for those fields relating to file uploads, the user should be prompted to select the file to upload. In the form definition file, add a STYLE attribute of 'FileUpload' to the field or fields where the user selects the file to upload. For example, from the .PER file:

```
  EDIT main1 = formonly.main1, STYLE="FileUpload"
```

When an EDIT field has the STYLE attribute of 'FileUpload', the FileUpload.xhtml snippet is used to render that field. All other EDIT fields continue to use the Edit.xhtml snippet for rendering.

Template modifications

For the PAGE and PDA snippet sets only, add the attribute `enctype="multipart/form-data"` to the `<form>` tag in the template file (main.xhtml). For example, if the template file states:

```
  <form method="post" id="gDialogForm" gwc:attributes="action
document/URL">
```

Then update the entry to include the attribute `enctype="multipartform-data"`:

```
  <form method="post" id="gDialogForm" gwc:attributes="action
document/URL" enctype="multipart/form-data">
```

File Transfer: Downloading a File

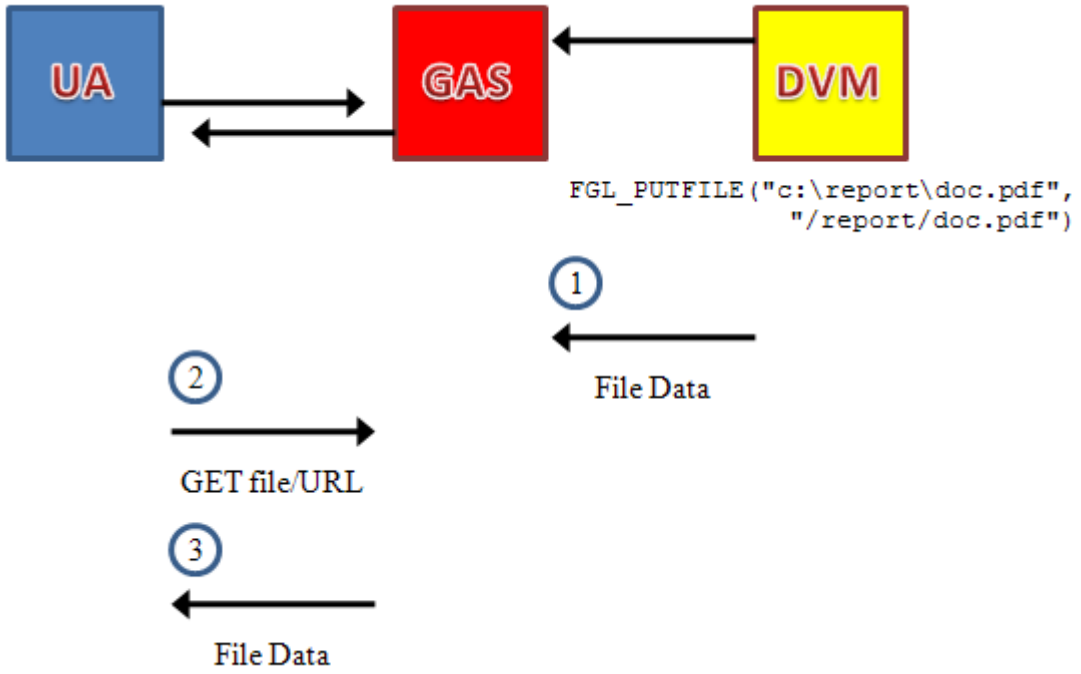
In a Genero application, downloading a file from the DVM (application server host) is handled by the fgl_putfile() built-in function. For details about this function, refer to the *Built-In Functions* topic in the *Genero Business Development Language Manual*.

When preparing to deliver an application that includes a file download via the GWC, no modifications need to be made to the source files, form definition files, templates, or snippet sets. The application, and the file download, will work as-is given the default snippets.

- Download with FGL_PUTFILE()
- Using document/bloburl

Download with FGL_PUTFILE()

The next figure illustrates the process of downloading a file to the front end host from the DVM using the FGL_PUTFILE() built-in method.



Notes:

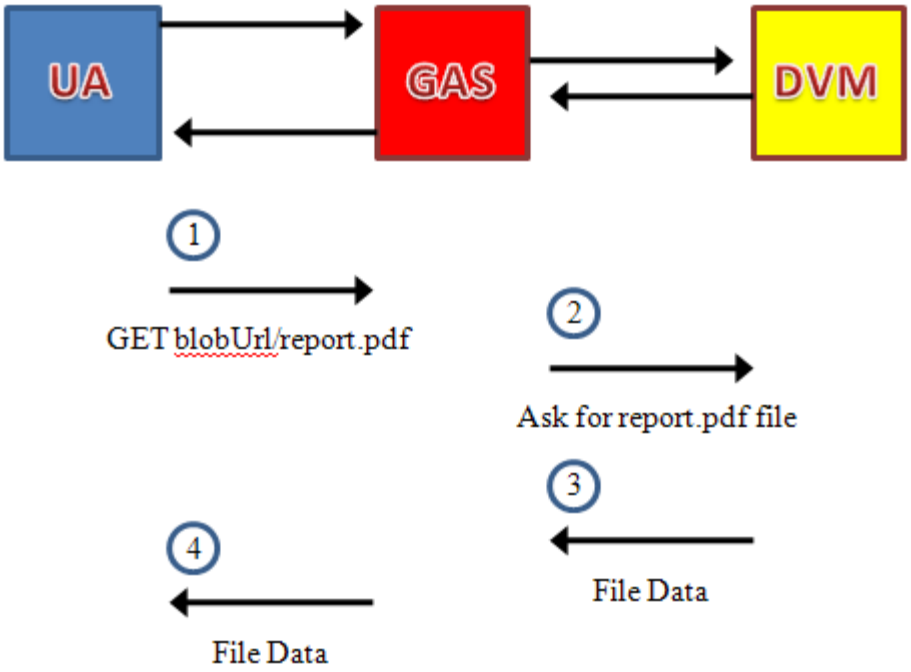
1. The FGL_PUTFILE command causes the file to be downloaded to move to the application server.
2. Retrieving a file sent by the DVM via an FGL_PUTFILE call is done using a File Transfer path (file/url). For details about File Transfer paths, see FileTransfer Object in Template Paths - Application hierarchy.
3. File data is downloaded to the front end host.

Using document/bloburl

The next figure illustrates the process of using the document/bloburl path to build an URL that requests files located on the DVM host.

To create the link, you would add code similar to the following to your template file:

```
<a gwc:attributes="href document/blobUrl + '/report.pdf'">Click to
download report</a>
```



Notes:

1. The front end requests the file using the document/blobUrl path. For details on the document/blobUrl path, see the Document object in Template Paths - Document hierarchy.
2. The GAS passes the request to the DVM.
   The DVM then searches for the requested file in the FGLIMAGEPATH directory. For information on FGLIMAGEPATH, please refer to the 'Environment Variables' topic in the *Genero Business Development Language Manual*.
3. Once the file is located, the DVM sends the file to the GAS.
4. The GAS then sends the file to the front end.

You can download any type of file that resides in the FGLIMAGEPATH directory. It is not limited to image files. A common use is when an application creates a report and wants to display a link that the user can click to display the document on the front end.

File Transfer and the GAS Configuration File

- File transfer timeout
- File upload temporary directory

For complete details on the configuration file elements referenced below, refer to the *Genero Application Server Manual*.

File Transfer Timeout

In the GAS configuration file, the `FILE_TRANSFER`'s `TIMEOUT` element determines when uploaded files are deleted. In other words, the timeout value determines how long uploaded files remain available. The files are removed from the temporary directory after the timeout period specified between the `TIMEOUT` tags elapses. The files are also removed when the GAS is shut down.

```
<FILE_TRANSFER>
  <TIMEOUT> timeout </TIMEOUT>
</FILE_TRANSFER>
```

Notes:

1. *timeout* is specified in seconds.
2. By default, the value is 600 seconds (10 minutes).

File Upload Temporary Directory

In the GAS configuration file, you specify the directory in which the uploaded files are stored. The DVM retrieves the file from this temporary directory for processing. The files remain in the temporary directory for the duration of the timeout period.

Important! This directory does not impact the fgl_getfile() destination directory and file name. It instead represents the temporary holding area between the Front End and the DVM.

```
<INTERFACE_TO_CONNECTOR>
  [...]
  <TEMPORARY_DIRECTORY> dir </TEMPORARY_DIRECTORY>
  [...]
</INTERFACE_TO_CONNECTOR>
```

Note:

1. By default, the `TEMPORARY_DIRECTORY` element is set to:
   `<TEMPORARY_DIRECTORY>$(res.path.tmp)</TEMPORARY_DIRECTORY>`
   The default value of the resource `$(res.path.tmp)` is $FGLASDIR/tmp, where the value of $FGLASDIR is dependant on the operating system.

Troubleshooting FAQ

- EDIT field rendering
- Forms statement error -8067
- Forms statement error -8066

EDIT field rendering

Issue: The EDIT field does not render with a browse button.

Solution: Ensure two items: the EDIT field is defined with STYLE="FileUpload", and the snippet set includes the FileUpload.html snippet. See File Transfer: Uploading a File for details.

Forms statement error -8067

Issue: When attempting to upload a file, the following error displays:
Event(Time="772.338228', Type='VM error data') / FORMS statement error number -8067. \012Could not read source file for file transfer.\012

Solution: You must specify `enctype="multipart/form-data"` in the template's `FORM` tag. See Template Modifications for details.

Forms statement error -8066

Issue: When attempting to upload a file, the following error displays:
Event(Time="15.928500", Type='VM error data') / Program stopped at 'test.4gl', line number 27.\012FORMS statement error number -8066.\012Could not write destination file for file transfer.\012</Event>

Solution: Check that the path is correct or that you have the permissions to write in the directory you upload the file to.

Understanding the Snippet-Based Rendering Engine

Topics

What is the Snippet-Based Rendering Engine?

Prior to Genero Web Client 2.10, there was a single rendering engine known as the built-in rendering engine. As the name implies, how an application was rendered was "built in" to the rendering engine. Starting with GWC 2.10, however, the GWC comes with a new rendering engine: a snippet-based rendering engine has been added, while the legacy built-in rendering engine remains for backwards compatibility.

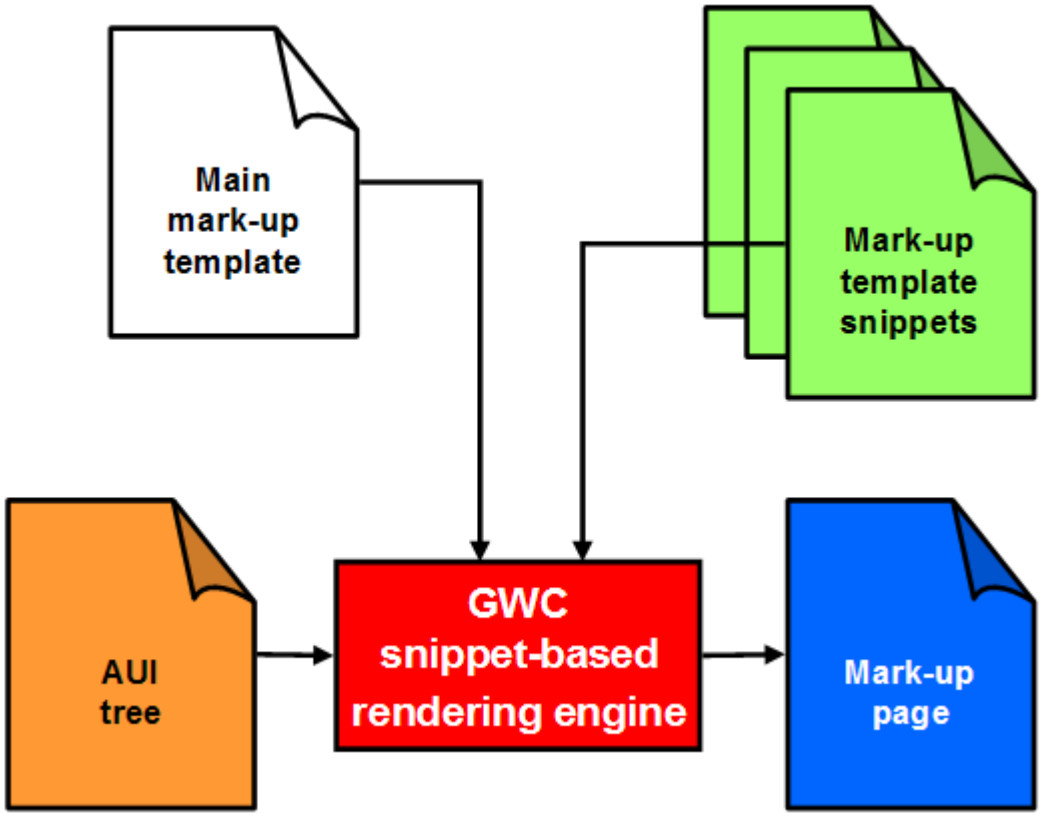
The snippet-based rendering engine allows application developers and designers to fully customize the rendering of a Genero application user interface in a web browser through the use of externally-defined template snippets. With these snippets, you can adapt the output for any kind of web browsers, from the simplest PDA to the best JavaScript-enabled desktop browser. In addition, device-specific mark-up language can be added to connect external devices (such as barcode readers) to Genero applications.

By default, the GWC defaults to using the snippet-based rendering engine. The rendering engine selected is defined by an OutputMap, and the OutputMap is chosen based on the browser signature or is explicitly specified in the application's URL. See How Browser-Based Themes, Templates, and Snippet Sets work for the GWC for more information.

Understanding Snippet-Based Rendering

Snippet-based rendering relies upon the GWC object model. The GWC object model ensures that all available data regarding the application is used to render the page properly. Sources for the data include the application's abstract user interface (AUI) , the document data (holds information about the rendering of the current document such as the URL to send the form back to, rendering errors, and so on), and server data (gives access to static information about the server, such as the application server version number).

The GWC object model exposes this data through properties. These properties are accessible by means of a path notation within GWC expressions; the GWC expressions held by GWC template attributes.



The GWC reads the application's AUI tree, which is the abstract definition of the application's current user interface. Some of the objects in the object model point to a well-defined entity within the AUI tree (such as a Window object, a Form object, and Edit object, and so on). Other objects in the object model correspond to an entity that is used by other AUI objects (such as the GridLayout component, created each time a Grid, ScrollGrid, or Group object appear in the AUI tree). The objects in this object model are wrapped by GWC UI Components.

The rendering of a GWC UI Component is driven by the template snippets. The template snippets are read dynamically at runtime. A template snippet is parsed within the context of the associated component. The main template controls the overall rendering for the application page.

The template and snippet files provided for use with the snippet-based rendering engine are either XHTML or XSL (in comparison with the HTML templates used with the legacy built-in rendering issue).

- XHTML is merely an adaptation of HTML to be XML well-formatted; XHTML template and snippet files are XML well-formatted documents. When a snippet is using XHTML, then you can use gwc template language (gwc tags) within the snippet file.

- XSLT snippets use standard XSLT; you cannot use gwc tags within this file. (Right now, only used to provide a snippet to handle the StyleList node and subnodes from the AUI tree.)

NOTE: The rendering engine will not load XML data referenced in snippets such as XML schemas or DTDs. As a result, the rendering engine will not be able to control or validate the snippet content against these schemas or use entities defined in these schemas. Therefore, snippets should not embed references such as **é** as this could result in unwanted behavior.

Rendering Configuration

An application's configuration ultimately decides whether an application is rendered using the built-in rendering engine or the snippet-based rendering engine. This section identifies those parts of the Genero Application Server configuration file that determine how the application is rendered.

Output Drivers

The OUTPUT_DRIVER element (found within WEB_APPLICATION_RENDERING_COMPONENT elements) defines the three different rendering engine options. From the Genero Application Server configuration file:

```
01  <WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.gwc">
02    <OUTPUT_DRIVER>GWC</OUTPUT_DRIVER>
03  </WEB_APPLICATION_RENDERING_COMPONENT>
04  <WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.gwc2xhtml">
05    <OUTPUT_DRIVER>GWC2</OUTPUT_DRIVER>
06  </WEB_APPLICATION_RENDERING_COMPONENT>
07  <WEB_APPLICATION_RENDERING_COMPONENT Id="cpn.rendering.xslt">
08    <OUTPUT_DRIVER>XSLT10</OUTPUT_DRIVER>
09  </WEB_APPLICATION_RENDERING_COMPONENT>
```

There are three output driver options:

- GWC (Line 02) - When this output driver is specified, the legacy built-in rendering engine is used.
- GWC2 (Line 05) - When this output driver is specified, the snippet-based rendering engine is used.
- XSLT10 (Line 08) - This output driver was created to enable customers to deploy Web applications to specific handheld devices prior to the implementation of the snippet-based rendering engine. You should not use this output driver unless instructed so by Genero support personnel.

Important: The output driver that should be used is GWC2. the others are available for legacy application compatibility.

Themes / Snippet Sets

A WEB_APPLICATION_THEME_COMPONENT defines the theme (the set of templates and snippets) to be used by the GWC when rendering an application that requests that theme. A theme designed for the built-in rendering engine only contains TEMPLATE elements (as SNIPPET elements are ignored by the built-in rendering engine), while a theme designed for the snippet-based rendering engine contains both TEMPLATE and SNIPPET elements.

For example, the AJAX theme includes both TEMPLATE and SNIPPET elements, with various TEMPLATE elements defining the overall presentation of the application on a page and various SNIPPET elements for each AUI object that could be displayed by a form. As an application developer or designer, you can customize the XHTML snippet files to customize an AUI object.

```
01  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.ajax.gwc">
...
07    <TEMPLATE
Id="_default">$(res.path.tpl.ajax)/main.xhtml</TEMPLATE>
08
09    <SNIPPET Id="Window">$(res.path.tpl.ajax)/Window.xhtml</SNIPPET>
10    <SNIPPET Id="Menu">$(res.path.tpl.ajax)/Menu.xhtml</SNIPPET>
11    <SNIPPET
Id="MenuAction">$(res.path.tpl.ajax)/MenuAction.xhtml</SNIPPET>
12    <SNIPPET Id="Dialog">$(res.path.tpl.ajax)/Dialog.xhtml</SNIPPET>
13    <SNIPPET Id="Action">$(res.path.tpl.ajax)/Action.xhtml</SNIPPET>
14    <SNIPPET Id="Form">$(res.path.tpl.ajax)/Form.xhtml</SNIPPET>
15    <SNIPPET Id="VBox">$(res.path.tpl.ajax)/VBox.xhtml</SNIPPET>
16    <SNIPPET Id="HBox">$(res.path.tpl.ajax)/HBox.xhtml</SNIPPET>
17    <SNIPPET Id="Group">$(res.path.tpl.ajax)/Group.xhtml</SNIPPET>
18    <SNIPPET Id="Table">$(res.path.tpl.ajax)/Table.xhtml</SNIPPET>
19    <SNIPPET Id="Grid">$(res.path.tpl.ajax)/Grid.xhtml</SNIPPET>
20    <SNIPPET
Id="ScrollGrid">$(res.path.tpl.ajax)/ScrollGrid.xhtml</SNIPPET>
21    <SNIPPET
Id="GridLayout">$(res.path.tpl.ajax)/GridLayout.xhtml</SNIPPET>
22    <SNIPPET
Id="FormField">$(res.path.tpl.ajax)/FormField.xhtml</SNIPPET>
23    <SNIPPET Id="Folder">$(res.path.tpl.ajax)/Folder.xhtml</SNIPPET>
24    <SNIPPET Id="Edit">$(res.path.tpl.ajax)/Edit.xhtml</SNIPPET>
25    <SNIPPET
Id="DateEdit">$(res.path.tpl.ajax)/DateEdit.xhtml</SNIPPET>
26    <SNIPPET
Id="ButtonEdit">$(res.path.tpl.ajax)/ButtonEdit.xhtml</SNIPPET>
27    <SNIPPET
Id="TextEdit">$(res.path.tpl.ajax)/TextEdit.xhtml</SNIPPET>
28    <SNIPPET Id="Label">$(res.path.tpl.ajax)/Label.xhtml</SNIPPET>
29    <SNIPPET
Id="ComboBox">$(res.path.tpl.ajax)/ComboBox.xhtml</SNIPPET>
30    <SNIPPET Id="Button">$(res.path.tpl.ajax)/Button.xhtml</SNIPPET>
```

```
31    <SNIPPET
Id="CheckBox">$(res.path.tpl.ajax)/CheckBox.xhtml</SNIPPET>
32    <SNIPPET
Id="RadioGroup">$(res.path.tpl.ajax)/RadioGroup.xhtml</SNIPPET>
33    <SNIPPET Id="Image">$(res.path.tpl.ajax)/Image.xhtml</SNIPPET>
34    <SNIPPET
Id="StaticLabel">$(res.path.tpl.ajax)/StaticLabel.xhtml</SNIPPET>
35    <SNIPPET
Id="StaticImage">$(res.path.tpl.ajax)/StaticImage.xhtml</SNIPPET>
36    <SNIPPET Id="Slider">$(res.path.tpl.ajax)/Slider.xhtml</SNIPPET>
37    <SNIPPET
Id="SpinEdit">$(res.path.tpl.ajax)/SpinEdit.xhtml</SNIPPET>
38    <SNIPPET
Id="TimeEdit">$(res.path.tpl.ajax)/TimeEdit.xhtml</SNIPPET>
39    <SNIPPET
Id="ProgressBar">$(res.path.tpl.common)/ProgressBar.xhtml</SNIPPET>
40    <SNIPPET Id="HLine">$(res.path.tpl.ajax)/HLine.xhtml</SNIPPET>
41    <SNIPPET
Id="HBoxTag">$(res.path.tpl.ajax)/HBoxTag.xhtml</SNIPPET>
42    <SNIPPET
Id="TopMenu">$(res.path.tpl.ajax)/TopMenu.xhtml</SNIPPET>
43    <SNIPPET
Id="TopMenuGroup">$(res.path.tpl.ajax)/TopMenuGroup.xhtml</SNIPPET>
44    <SNIPPET
Id="TopMenuCommand">$(res.path.tpl.ajax)/TopMenuCommand.xhtml</SNIPPET>
45    <SNIPPET
Id="TopMenuSeparator">$(res.path.tpl.ajax)/TopMenuSeparator.xhtml</SNIP
PET>
46    <SNIPPET
Id="ToolBar">$(res.path.tpl.ajax)/ToolBar.xhtml</SNIPPET>
47    <SNIPPET
Id="ToolBarItem">$(res.path.tpl.ajax)/ToolBarItem.xhtml</SNIPPET>
48    <SNIPPET
Id="ToolBarSeparator">$(res.path.tpl.ajax)/ToolBarSeparator.xhtml</SNIP
PET>
49    <SNIPPET
Id="StartMenu">$(res.path.tpl.ajax)/StartMenu.xhtml</SNIPPET>
50    <SNIPPET
Id="StartMenuGroup">$(res.path.tpl.ajax)/StartMenuGroup.xhtml</SNIPPET>
51    <SNIPPET
Id="StartMenuCommand">$(res.path.tpl.ajax)/StartMenuCommand.xhtml</SNIP
PET>
52    <SNIPPET
Id="StartMenuSeparator">$(res.path.tpl.ajax)/StartMenuSeparator.xhtml</
SNIPPET>
53    <SNIPPET
Id="EndingPage">$(res.path.tpl.common)/EndingPage.xhtml</SNIPPET>
54    <SNIPPET
Id="FileTransfert">$(res.path.tpl.common)/FileTransfer.xhtml</SNIPPET>
55    <SNIPPET
Id="StyleList">$(res.path.tpl.common)/StyleList.xsl</SNIPPET>
56  </WEB_APPLICATION_THEME_COMPONENT>
```

Notes

1. Line 01 specifies the component's Id: "cpn.theme.ajax.gwc". An application definition can inherit the settings of this component by specifying this Id value in the Using attribute in its THEME element.
2. Line 07 defines the main template file for this component.
3. Lines 09 - 55 provide the template snippets for this component.

For information on customizing a template or snippet file, see Customizing Templates and Snippets.

Output Maps

As discussed above, how an application is rendered by the Genero Web Client depends on two components: the RENDERING component (which defines which rendering engine is used) and the THEME component (which specifies which template and snippet files to use).

An Output Map gives the ability to group together a single RENDERING component and a single THEME component into a named Output Map, which can then be specified for an application. By using the Using attribute, you can specify a WEB_APPLICATION_RENDERING_COMPONENT and a WEB_APPLICATION_THEME_COMPONENT, previously defined within the COMPONENT_LIST section of the Genero Application Server configuration file.

Output Maps are defined within an APPLICATION element (defined within the APPLICATION_LIST section). For example, the GWC abstract application defines various MAP elements; an application that specifies this application as its PARENT application will be able to use one of these Output Maps, which in turn specifies the RENDERING and THEME to use when rendering the application. You can allow or deny some OutputMap by setting the Allowed attribute accordingly.

```
01  <!--This is the default application for GWC-->
02  <APPLICATION Id="defaultgwc" Parent="defaultwa" Abstract="TRUE">
03    <TIMEOUT Using="cpn.gwc.timeout.set1"/>
04    <PICTURE Using="cpn.gwc.picture"/>
05    <OUTPUT Rule="UseGWC">
06      <MAP Id="DUA_Symbol-WC" Allowed="TRUE">
07        <RENDERING Using="cpn.rendering.xslt"/>
08        <THEME Using="cpn.theme.default.gwc">
09           <TEMPLATE Id="_default">$(res.theme.symbol-
wc.stylesheet)</TEMPLATE>
10        </THEME>
11      </MAP>
12      <MAP Id="DUA_GWC" Allowed="TRUE">
13        <RENDERING Using="cpn.rendering.gwc"/>
14        <THEME Using="cpn.theme.default.gwc"/>
15      </MAP>
16      <MAP Id="DUA_AJAX_HTML" Allowed="TRUE">
17        <RENDERING Using="cpn.rendering.gwc2html" />
18        <THEME Using="cpn.theme.ajax.gwc" />
19      </MAP>
```

```
20      <MAP Id="DUA_AJAX" Allowed="TRUE">
21        <RENDERING Using="cpn.rendering.gwc2xhtml" />
22        <THEME Using="cpn.theme.ajax.gwc" />
23      </MAP>
24      <MAP Id="DUA_PAGE_HTML" Allowed="TRUE">
25      <RENDERING Using="cpn.rendering.gwc2html" />
26      <THEME Using="cpn.theme.page.gwc" />
27      </MAP>
28      <MAP Id="DUA_PAGE" Allowed="TRUE">
29        <RENDERING Using="cpn.rendering.gwc2xhtml" />
30        <THEME Using="cpn.theme.page.gwc" />
31      </MAP>
32      <MAP Id="DUA_PDA" Allowed="TRUE">
33        <RENDERING Using="cpn.rendering.gwc2" />
34        <THEME Using="cpn.theme.pda.gwc" />
35      </MAP>
36    </OUTPUT>
37  </APPLICATION>
```

Notes

1. Line 02, the application Id stated defaultgwc. An application that specfies defaultgwc as its Parent will inherit these settings.
2. Line 05, the Output Rule is to "UseGWC", meaning that this application is to be displayed using the Genero Web Client.
3. Lines 06 - 35 define the various Output Map options made possible (Allowed="TRUE") to an application that inherits the configuration of this abstract application.
4. Lines 06 - 11 define the DUA_Symbol-WC Output Map. Do not use this map unless instructed by Four J's support.
5. Lines 12 - 15 define the DUA_GWC Output Map. Its RENDERING element specifies "cpn.rendering.gwc", which maps to the GWC output driver, which in turn informs the Genero Web Client to use the **built-in** rendering engine to render the application (as discussed in the previous section). It also specifies that "cpn.theme.default.gwc" as the theme.
6. Lines 16 - 19 define the DUA_AJAX_HTML Output Map. Its RENDERING element specifies "cpn.rendering.gwc2html", which maps to the GWC output driver, which in turn informs the Genero Web Client to use the **snippet-based** rendering engine to render the application (as discussed in the previous section).
   Tip: The DUA_AJAX_HTML Output Map is designed to be the default Output Map for Internet Explorer.
7. Lines 20 - 23 define the DUA_AJAX_HTML Output Map. Its RENDERING element specifies "cpn.rendering.gwc2xhtml", which maps to the GWC output driver, which in turn informs the Genero Web Client to use the **snippet-based** rendering engine to render the application (as discussed in the previous section).
   Tip: The DUA_AJAX_HTML Output Map is designed to be the default Output Map for Mozilla FireFox and other browsers.
8. Lines 24 - 27 define the DUA_PAGE Output Map. Its RENDERING element specifies "cpn.rendering.gwc2", which maps to the GWC output driver, which in turn informs the Genero Web Client to use the snippet-based rendering engine to render the application (as discussed in the previous section).
   Tip: The DUA_PAGE_HTML Output Map is designed as a reference

implementation. It provides a basic implementation of each snippet, illustrating the behavior of the snipped-based rendering engine and providing a reference implementation for testing purposes. Its does not use the Client Side Framework. It is recommended that you use the DUA_AJAX or DUA_PDA Output Maps instead.

9. Lines 28 - 31 define the DUA_PAGE Output Map. Its RENDERING element specifies "cpn.rendering.gwc2", which maps to the GWC output driver, which in turn informs the Genero Web Client to use the snippet-based rendering engine to render the application (as discussed in the previous section).
   Tip: The DUA_PAGE_HTML Output Map is designed as a reference implementation. It provides a basic implementation of each snippet, illustrating the behavior of the snipped-based rendering engine and providing a reference implementation for testing purposes. Its does not use the Client Side Framework. It is recommended that you use the DUA_AJAX or DUA_PDA Output Maps instead.
10. Lines 32 - 35 define the DUA_PDA Output Map. Its RENDERING element specifies "cpn.rendering.gwc2", which maps to the GWC output driver, which in turn informs the Genero Web Client to use the snippet-based rendering engine to render the application (as discussed in the previous section).
    Tip: The DUA_PDA Output Map is designed to be the default Output Map for browsers on hand-held devices.

For complete details on all elements of the Genero Application Server configuration file, refer to the GAS Configuration Reference.

Why are some widgets partially rendered?

The default rendering is accomplished with CSS and javascript. If you lack one of the two features, the rendering may be incorrect.

- Check that your browser meets the browser requirements and that it supports javascript.
- Check your installed files, especially the directory "web/fjs" on the application server (AS) side.
- Check that CSS and javascript files are reachable on the AS side.
  - With direct connection, type the URL in your browser - http://<server>:6394/fjs/defaultTheme/genero.css
  - With connection through a web server, apache for example, use http://<server>/cgi-bin/fglccgi/fjs/default/genero.css
- Check that your templates have `application/connectorURI` in front of each reference to a file on the application server. For example, gwccomponents.css is on the application server, the reference to this file in the default template is:

```
<link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/gwccomponents.css'"
type="text/css" title="Default Theme"/>
```

User Interface Customization Options

There are a variety of options available that allow you to customize the application interface delivered by the Genero Web Client:

- Customize with Genero presentation styles
- Customize with CSS
- Customize templates and snippets
- The User Interface and JavaScript

You can use all four methods when customizing your application; the methods are not exclusive. Best practices state that you should customize first with Genero Presentation Styles, then with CSS, then with modification of the template and snippet files, and finally with JavaScript.

Customize with Genero Presentation Styles

Genero Presentation Styles allow you to define a set of decoration properties to be used for graphical objects. Presentation Styles are provided to centralize attributes related to the user interface elements. Genero presentation styles provide a centralized customization for all application displayed by the Genero Desktop Client (GDC), Genero Web Client (GWC), or Genero Java Client (GJC). Typical presentation attributes define font properties and foreground and background colors. Some presentation attributes will be specific to a given class of widgets.

Presentation Styles are defined in a **.4st** resource file which is distributed with other runtime files.

For more information:

- Customize the User Interface with Genero Presentation Styles

---

Customize with CSS

Cascading Style Sheets (CSS) provide a simple mechanism for adding styles (such as fonts, colors, and spacing) to Web documents. You will typically use CSS when you want to make simple interface changes, such as using different colors.

For more information:

- Customize the User Interface with Cascading Style Sheets (CSS)
- Template CSS Reference

Customize Templates and Snippets

Templates and snippet sets provide you with a mechanism for customizing the rendering of the HTML displayed in the browser. By customizing the template and snippet files, you can change the layout of the application and the structure of the widgets.

For more information:

- How snippets render the application interface
- Customizie the User Interface with Templates and Snippets

Customize with Javascript

JavaScript provides an interaction with the engine. JavaScript can enable you to create your own widgets.

For more information:

- Customize the User Interface with JavaScript

Customize the User Interface with Genero Presentation Styles

Starting with GWC 2.10, the Genero Application Server generates a CSS for those applications that currently use Genero Presentation Styles (defined in a .4st file).

Topics

How GWC generates CSS from Presentation Styles

Genero presentation styles use a .4st file to provide centralized styles for use across the various Genero application front-ends. It is assumed that you have created your .4st file that defines the styles you want to apply to the application. This file defines the set of attributes related to the decoration of user interface elements.

In addition to having a presentation style file created, your application must also call the appropriate .4st file using the ui.Interface.loadStyles() method.

The DVM creates the AUI Tree. This AUI tree includes the StyleList node, which contains child Style nodes. These child Style nodes contain StyleAttribute nodes, where each StyleAttribute node contains a name (attribute name) and value.

The GWC rendering engine converts these StyleAttributes into CSS based upon the XSLT document defined for the selected theme. For example, the default "cpn.theme.ajax.gwc" theme has the following SNIPPET element defined:

```
    <SNIPPET
Id="StyleList">$(res.path.tpl.common)/StyleList.xsl</SNIPPET>
```

This file, the StyleList.xsl file, is an XSLT file that defines how the .4st file is translated into the appropriate CSS entry.

Customizing how the GWC transforms your Presentation Styles

While the GWC can process the entries in the .4st file, not all .4st entries are translated into **valid** CSS entries by default.

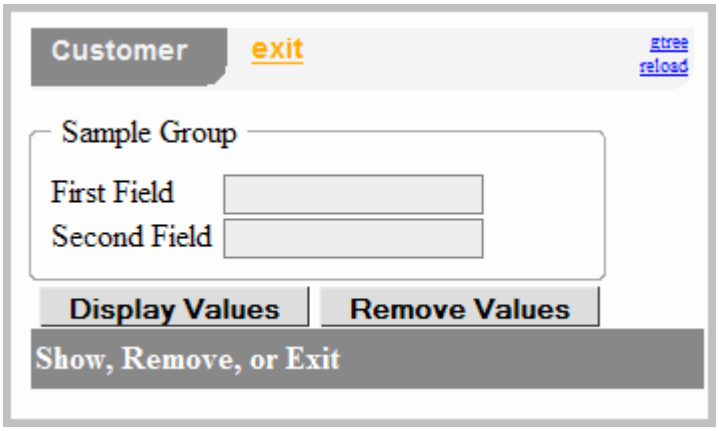
In this example, the .4st file used a color "lightMagenta". For example, to define a background color for the Edit field, we created a .4st file as follows:

```
<StyleList>
  <Style name="Edit">
    <StyleAttribute name="backgroundColor" value="lightMagenta"/>
  </Style>
</StyleList>
```

If the GWC generated a CSS entry for the application as follows:

```
.gEdit {
    background-color: lightMagenta;
}
```

When the application surfaces in the browser, the background color for the fields would not be light magenta. The reason behind this is that "lightMagenta" is not a valid color for CSS.



Note:

1. With the default StyleList.xsl file, GWC makes an automatic conversion of the following color names to RGB values: green, darkOlive, lightTeal, lightOrange, lightMagenta, darkTeal, lightRed, darkYellow, lightYellow. As such, the color 'lightMagenta' is translated into a valid CSS color.

To map your own RGB value to the 'lightMagenta' color, modify the StyleList.xsl file to change the value "lightMagenta" to the appropriate RGB color value before displaying the application in a browser. Modifying this file consists of three steps:

Step 1: Create a copy of the default StyleList.xsl file. By creating a copy of the file, you avoid having the file overwritten when you upgrade the GWC. For this example, let's name this file StyleListCustom.xsl.

Step 2: Within the WEB_APPLICATION_THEME_COMPONENT in the GAS configuration file, modify the SNIPPET element for the StyleList to reference your copy of the file.

```
<SNIPPET
Id="StyleList">$(res.path.tpl.common)/StyleListCustom.xsl</SNIPPET>
```

Step 3: Open the StyleListCustom.xsl file and create a new entry that sets the "lightMagenta" color to the desired RGB value.

```
<xsl:template name="translateColor">
  <xsl:choose>
    <xsl:when test="@value='green'">#00ff00</xsl:when>
    <xsl:when test="@value='lightMagenta'">#ffc0ff</xsl:when>
    [...]
    <xsl:otherwise><xsl:value-of select="@value"/></xsl:otherwise>
</xsl:template>
```

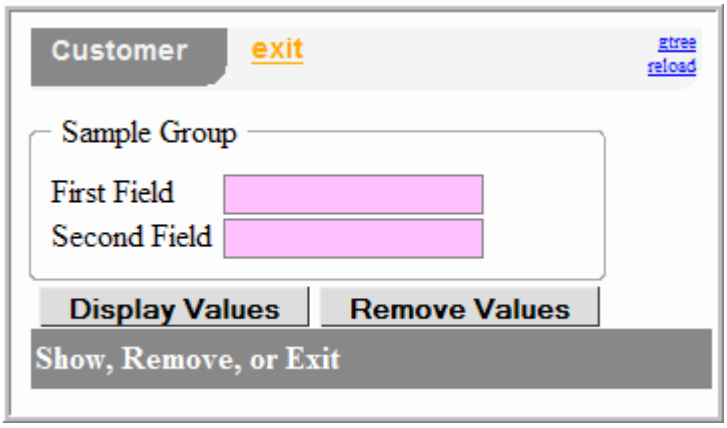
Notes:

- To create an entry we copied the line above defining the translation for the value 'green' and modified as required.
- To find the hexadecimal name that is valid for CSS, we looked up the valid color names and hex names on the W3C site: http://www.w3schools.com/css/css_colornames.asp
- This file is written in XSLT. To make modifications to this file, it would be very useful to understand and be able to work with XSLT. The site w3schools.com provides online training for XSLT.

After this entry has been made to the style list file and the changes saved, the appropriate CSS is generated when the application is run. You can view the source created and see that the CSS entry has changed to the following:

```
.gEdit {
    background-color: #ffc0ff;
}
```

This is valid CSS and is able to be interpreted by the browser.



What takes priority: Presentation Styles or existing CSS files

If you have the same attribute defined both in the CSS stylesheets (such as the gwccomponents.css) and in the .4st file, the .4st file will take priority.

For example, if we add a background color to the gwccomponents.css file:

```
.gEdit {
border-width: 1px;
background-color: aqua;

}
```

We then also have the background color for the Edit field defined in our .4st file:

```
<StyleList>
  <Style name="Edit">
    <StyleAttribute name="backgroundColor" value="#7FFF00"/>

  </Style>
</StyleList>
```

The rendering engine will translate the .4st into a CSS entry, and this entry will take precedence over the value defined in the CSS stylesheet.

Generating CSS from Presentation Styles - Limitations

Limitations exist regarding:

- Common attributes
- Pseudo selectors
- TTY attributes
- Browser limitations

Common Attributes

Only the common attributes provided by the Genero Presentation Styles are translated into CSS. For other presentation styles (such as positioning), customization must be done within the snippet files. For more information on the Genero Presentation Styles, refer to the Presentation Styles topic in the *Genero Business Development Language Manual*. .

Pseudo Selectors

Genero presentation styles include pseudo selectors, where a style is applied only when some conditions are fulfilled.

```
<Style name="Edit:focus" >
```

Pseudo selectors, however, are NOT translated into CSS by the Genero Web Client. In order to handle pseudo selectors, you must include style paths within the snippet file itself.

For more information on pseudo selectors defined in the presentation styles, refer to the Pseudo Selectors paragraph within the Presentation Styles topic in the *Genero Business Development Language Manual*.

TTY attributes

The GWC does not handle the entries for (TTY-based) display attributes defined in either the .4GL or the form specification file, such as FONTPITCH, BOLD, REVERSE, BLINK, UNDERLINE, INVISIBLE, BLACK, and so on.

Browser Limitations

There are some difference between browsers on what the browser supports in terms of CSS. For example, Internet Explorer does not support background color on Edit fields.

Shortcuts style['allInlines4ST'] and style['allClasses4ST']

syle['allInlines4ST'] is equivalent to:

```
(style['textColor']?' color:'+colorToRGB(style['textColor'])+';':'')+
```

```
  (style['backgroundColor']?' background-
color:'+colorToRGB(style['backgroundColor'])+';':'')+
  (style['fontSize']?' font-size:'+style['fontSize']+';':'')
```

syle['allClasses4ST'] is equivalent to:

```
  (style['border']?' gBorder_'+style['border']:'') +
  (style['fontFamily']?' gFontFamily_'+style['fontFamily']:'') +
  (style['fontWeight']?' gFontWeight_'+style['fontWeight']:'') +
  (style['textDecoration']?'
gTextDecoration_'+style['textDecoration']:'') +
  (style['fontStyle'] ?' gFontStyle_'+style['fontStyle'] :'')
```

Customize the User Interface with Cascading Style Sheets (CSS)

CSS is one of the fastest and the simplest way to change your application's look. Most generated HTML elements have classes and containers (<DIV>) so you can change the widget's look and place the elements where you want. This section describes how you can modify existing or add new CSS. Understanding this chapter requires knowledge of CSS. We use CSS 2 standards described on the W3C site. For more information on CSS, refer to the tutorials at http://www.w3schools.com.

Topics

- Default CSS
- Adding New Styles
- Override Existing Styles
- Applying a Style to a Widget (Form Object)

For information about using existing .4st presentation style files, click here.

We assume that you have read the section How the GWC Uses Web Technologies and understand the principles.

Default CSS

The Genero Web Client provides default CSS files. Each set of snippets have their own CSS. Which CSS file is used is based on the Style Sheet link provided by the theme's template file. For example, if the application is using the AJAX theme (the DUA_AJAX Output Map), then the template being used is the main.xhtml template located in the directory $FGLASDIR/tpl/set1.

Within this template file:

```
01 <head>
02  <title gwc:condition="application/ui" gwc:marks="title
[application/ui/CID,window/name
    + ' - ' + application/ui/text + ' - ' +
document/URL]">Title</title>
03  <meta http-equiv="Content-Type" gwc:attributes="content

XPathConfig('/APPLICATION/OUTPUT/MAP[@Id=../@DUA]/RENDERING/MIME_TYPE/t
ext()')+';
    charset='+document/encoding" />
04  <script type="text/javascript" gwc:attributes="src
application/connectorURI+'/fjs/gwccomponents.js'"
```

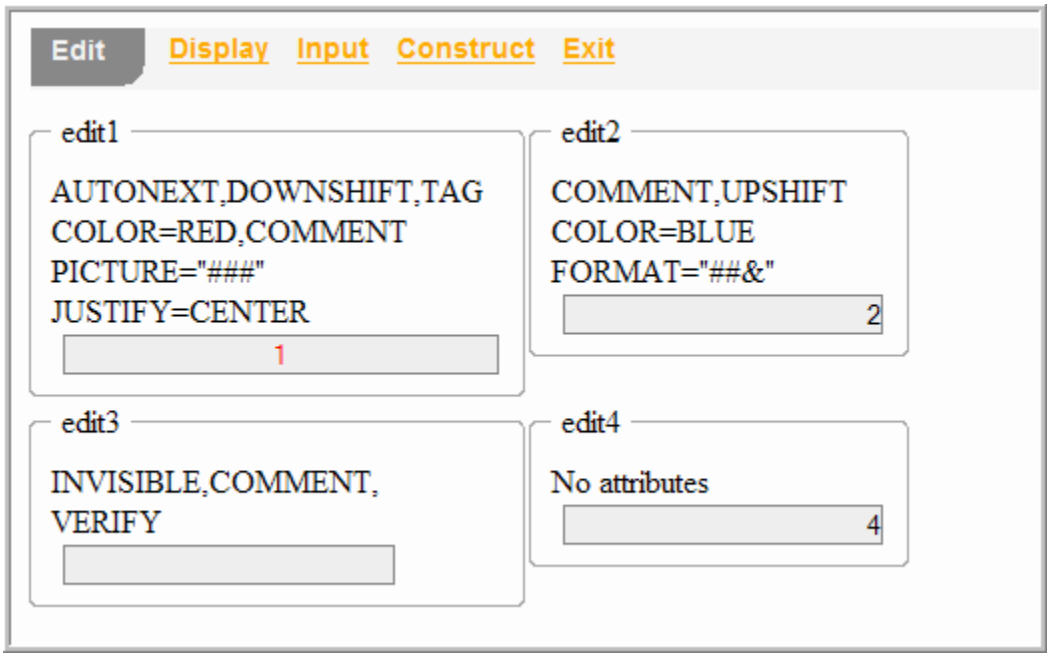
```
    defer="defer"> </script>
05  <script type="text/javascript" gwc:attributes="src
application/connectorURI+'/fjs/gwccore.js'"
    defer="defer"> </script>
06  <link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/gwccomponents.css'"
    type="text/css" title="Default Theme"/>
07  <link rel="shortcut icon" gwc:attributes="href
application/connectorURI+'/favicon.ico'"
    type="image/x-icon" />
08  <script type="text/javascript" gwc:content="'var gDialog=' +
document/dialog" />
09  <script type="text/javascript" gwc:content="'var gElementMarkTree='
+
    document/marks" />
10  <script type="text/javascript" gwc:content="'var gComponentList=' +
    document/components" />
11  <script type="text/javascript" gwc:condition="application/ui"
gwc:content="'var
    gDbdate=\'' + application/ui/dbdate + '\'" />
12  <!--script gwc:content="'alert(\''+application/state+'\');'"/-->
13  <!-- the next element will be replaced by the current 4GL window
based on the
    associated window snippet -->
14  <!-- if the 4GL window has a specific style, a snippet configured
with that
    style will be used -->

15  <style type="text/css" gwc:condition="application/ui/StyleList"
gwc:content="application/ui/StyleList"/>
16 </head>
```

The CSS used for applications being rendered using the AJAX theme is defined by the <LINK> element defined by line 07 in the example above. When you resolve the references, this link maps to the file $FGLASDIR/web/fjs/**gwccomponents.css**.

In the CSS reference section, you have the available styles for each widget. GWC styles are prefixed by the letter "g". Note that not all styles listed in the CSS Reference section have a default defined in the default genero.css file.

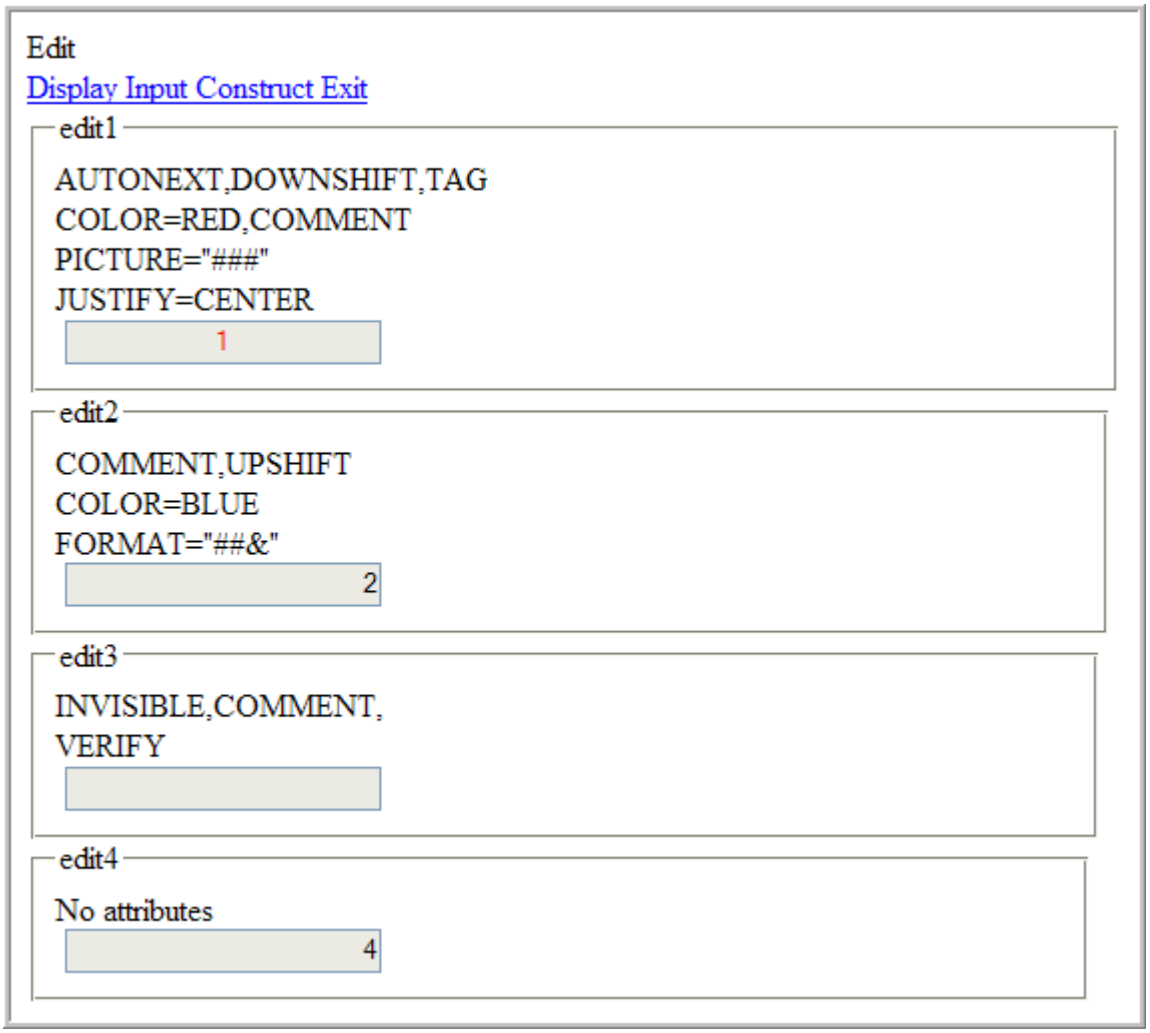
Example using CSS

The following screenshot is the Edit demonstration program using the default template **main.xhtml**, which specifies the **gwccomponents.css** style sheet.



Example without using CSS

The following screenshot is the same program (the Edit demonstration program) without a CSS applied. Although the application is fully functional, the appearance is not as nice as the program displayed using the default CSS.



Adding New Styles

When you define a new style or redefine an existing style, you can either use `LINK` tags to include styles declarations defined in an external file (an external CSS) or you can declare the style directly in your HTML template using `STYLE` tags. When style information is read by the browser, if a style is defined multiple times and/or in multiple locations, the one with the highest weight is used. For information regarding the weight between various styles, refer to the documentation provided on the W3C site.

Include a Custom Style Sheet

After creating a new style sheet in an external file, use `LINK` tags to include the style declarations defined in the external file. The `LINK` tag is added to the template. In the

code snippet below, styles defined in **mystyles.css** override styles defined in **genero.css** in accordance with the priorities described in CSS standards.

```
    ...
11  <link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/gwccomponents.css'"
    type="text/css" title="Default Theme"/>
12  <link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/mystyles.css'"
    type="text/css"/>
    ...
```

When the second link is added for the additional stylesheet, notice that we do not provide a second title attribute. Browsers such as FireFox use these titles to allow the user to select between the two defined stylesheets by the title rather than combining the two stylesheets. By eliminating the title attribute for the link, the resulting CSS is the combination of the two stylesheets. For selectors defined in both stylesheets, those selector attributes defined in the second stylesheet (line 12 above) take precedence over those selector attributes defined in the first stylesheet (line 11 above).

Adding a STYLE tag within the template

Use STYLE tags to declare a new style directly in the template file. This style will only be available to applications that use this template file.

```
    ...
04  <link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/gwccomponents.css'"
    type="text/css" title="Default Theme"/>
    ...
05  <style type="text/css" gwc:condition="application/ui/StyleList"
gwc:content="application/ui/StyleList"/>
05  <style type="text/css">
06  .mystyle {
07     width: 100%;
08     background-color: beige;
09  }
10  </style>
```

Overriding existing styles

you can override defined styles. To override defined styles, you have to give the same selector and add new styles or redefine existing styles.

For example, here is the style declaration from the default Genero CSS for the AJAX theme (**gwccomponents.css**) for the selector `.gDialog`. This selector is for a dialog object.

```
.gDialog,
.gMenu {
  float: left;
  clear: both;
  width: 95%;
  padding: 0px 16px 2px 0px;
  margin: 0px 2px 10px 2px;
  -moz-border-radius-bottomright: 16px;
  background-color: #F4F4F4;
  }
```

To add a new style for this selector -- for example, a background color -- simply add the CSS style `background-color` for this selector in either a custom CSS file or in a `STYLE` tag in your template file:

```
.gDialog{
   background-color: #FF0000;
}
```

As this is a new style, it will be added to the style information read from the **gwccomponents.css** file for this selector. And as the selector attribute `backgroud-color` is defined twice (initially in the **gwccomponents.css** file and again in your style declaration), the one with the higher weight will be used.

Applying a Style to a Widget (Form Object)

You can add your own styles for any widget (form object) by adding the `STYLE` attribute to the widget in the form definition file. (.per file or .4fd file). The **STYLE** attribute value is transmitted to the HTML **class** attribute.

```
01  EDIT f001=FORMONLY.field1, STYLE="style1 style2";
```

Generated HTML (part of)

```
<input type="text" id="ge29" title="" size="17" value="" style=""
class="gField
gInherit gDisabled gEdit gcstyle1 gcstyle2 gEdit_gcstyle1
gEdit_gcstyle2" readonly="readonly"/>
```

Corresponding CSS

```
  .gcstyle1 {
     border: 1px solid #00FF00;
     background-color: #FF0000;
  }

  .gcstyle2 {
     width: 100%;
```

```
}
```

Important! While the STYLE was defined as "style1" and "style2" for the widget (form object), the CSS must preface these style names with "gc" (for .name) and "g" (for name, like Edit)

For more information on the technical detail behind this specific topic, see Relating Styles, Classes, and Selectors.

Template CSS Reference

You can use Cascading Style Sheets (CSS) to customize the look and feel of your application.

Topics

CSS Syntax

CSS uses a specific syntax to define the style of HTML elements. This section presents the various syntax options.

Genero CSS Selectors

This section lists Genero CSS selectors by category (Containers, FormFields, Dialogs, and more). Not all selectors are valid for each of the three modes (AJAX, PAGE, PDA); you must view the generated HTML to see which selectors are valid for the mode (template and snippet sets) being used to display your application.

CSS Syntax

CSS uses a specific syntax to define the style of HTML elements.

Syntax

```
selector {
    style [...]
}
```

Notes

1. *selector* is a path defining on which HTML tags the styles have to be applied
2. *style* is a CSS style property as defined in the W3C site

Here is an excerpt of the selector syntax used with the Genero Web Client default theme:

Selector	Description
`BODY`	This selector applies to any `BODY` tag.
`#gWorkspace`	This selector applies to any tag having *gWorkspace* for id.

`.gGrid`	This selector applies to any tag having *gGrid* for class.
`.gMenu SPAN`	This selector applies to any `SPAN` tag which is a descendant of a tag having *gMenu* for class.
`INPUT.queryZone`	This selector applies to any `INPUT` tag having *queryZone* for class.
`.gToolBar .hover *`	This selector applies to any tag which is a descendant of a tag having *hover* for class and which is a descendant of a tag having *gToolBar* for class.

You can merge common styles configurations by grouping selectors in a comma separated list:

```
selector1 , selector2 , selector3 {
    styles [...]
}
```

For a complete reference for selector syntax, refer to the W3C site.

Template CSS

This section provides a list of selectors recognized by the Genero Web Client. It is divided into the following sections:

- Containers CSS: GRID, TABLE, SCROLLGRID, GROUP, FOLDER, PAGE, HBOX, VBOX
- FormFields CSS: FormField Box, EDIT, TEXTEDIT, BUTTON, BUTTONEDIT, DATEEDIT, CHECKBOX, COMBOBOX, RADIOGROUP, LABEL, Construct
- Dialog CSS: MENU, DIALOG, TOPMENU, TOOLBAR, MESSAGE, ERROR
- Other CSS

Notes

1. Not all selectors are valid for each of the three modes (AJAX, PAGE, PDA); you must view the generated HTML to see which selectors are valid for the mode (template and snippet sets) being used to display your application.
2. You can create your own selectors by adding classes to your snippet files.

Containers CSS

- GRID

- TABLE
- SCROLLGRID
- GROUP
- FOLDER
- PAGE
- HBOX
- VBOX

GRID

Selector	Description
`.gGrid`	The `SPAN` tag containing the grid
`.gGridLine`	A line of the grid

TABLE

Selector	Description
`.gTableBox`	The `SPAN` tag containing the table
`.gTable`	The `TABLE` tag
`.gTable col.gHidden`	A hidden table column
`.gTable TH` `.gTable THEAD TR TH`	A table header cell
`.gTable TR`	A table row
`.gTable TD`	A table cell
`.gTable TD *`	Any table cell descendant
`.gTable TD` `.gCurrentField`	The field having the focus in the table
`.gTable` `INPUT.gTableHeader`	A column input header; used to sort the table
`.gTable` `.disabledTableHeader`	A disabled table column header
`.gTable .disabled`	A disabled field of the table
`.gTable .gSortAsc`	An ascending sorted table column
`.gTable .gSortDesc`	A descending sorted table column
`.gTable` `.gCurrentRow *`	Any descendant of the currently selected row
`.gTable` `.gButtonEdit`	A ButtonEdit widget in the table

`.gTable .gAction`	An action item (ex: the button of a ButtonEdit widget) in the table
`.gTable .activeButtonEdit .gAction`	The action item of the currently active ButtonEdit widget in the table
`.gTable .activeButtonEdit .gButtonEdit`	The input part of the currently active ButtonEdit widget in the table
`.gFill .gTable`	A Grid Table once the JavaScript layout function has finished its processing

---

SCROLLGRID

Selector	Description
`.gScrollGridBox`	The `SPAN` tag containing the scrollgrid
`.gScrollGrid`	The scrollgrid itself
`.gHLineBox HR`	A horizontal line in the scrollgrid

GROUP

Selector	Description
`.gGroupBox`	The `SPAN` tag containing the group
`.gGroup .gGroupBox .gGroup`	The container box of a group
`.gGroupTitle .gGroupBox .gGroupTitle`	The title of a group

FOLDER

Selector	Description
`.gFolder`	The `DIV` tag containing the folder pages

PAGE

Selector	Description
`.gFolder .gPageHeader`	A page header
`.gFolder .gPage`	A folder header
`.gFolder .selectedPageHeader`	The currently selected page header
`.gFolder .selectedPage`	The currently selected page

HBOX

Selector	Description
`.gHBox`	A HBOX container
`.gHBox TD`	A HBOX column

VBOX

Selector	Description
`.gVBox`	A VBOX container
`.gVBoxLine .gVBox TD`	A VBOX line

FormFields CSS

- FormField Box
- EDIT
- TEXTEDIT

- BUTTON
- BUTTONEDIT
- DATEEDIT
- CHECKBOX
- COMBOBOX
- RADIOGROUP
- LABEL
- Construct

FormField Box

Selector	Description
.gFormFieldBox	The SPAN tag containing the formfield
.gCurrentField	The FormField having the input
.gJustifyCenter	A FormField having the JUSTIFY attribute set to *center*
.gJustifyLeft	A FormField having the JUSTIFY attribute set to *left*
.gJustifyRight	A FormField having the JUSTIFY attribute set to *right*
.gShiftDown	A FormField having the DOWNSHIFT attribute set
.gShiftUp	A FormField having the UPSHIFT attribute set
.gNoEntry	A FormField having the NOENTRY attribute set
.gNotNull	A FormField having the NOT NULL attribute set
.gRequired	A FormField having the REQUIRED attribute set
.gTypeByte	A FormField having the BYTE target type
.gTypeChar	A FormField having the CHAR target type
.gTypeDate	A FormField having the DATE target type
.gTypeDatetime	A FormField having the DATETIME target type
.gTypeDecimal	A FormField having the DECIMAL target type
.gTypeFloat	A FormField having the FLOAT target type
.gTypeInteger	A FormField having the INTEGER target type
.gTypeInterval	A FormField having the NTERVAL target type
.gTypeMoney	A FormField having the MONEY target type
.gTypeSmallfloat	A FormField having the SMALLFLOAT target type
.gTypeSmallint	A FormField having the SMALLINT target type
.gTypeString	A FormField having the STRING target type
.gTypeText	A FormField having the TEXT target type
.gTypeVarchar	A FormField having the VARCHAR target type
.gVerify	A FormField having the VERIFY attribute set

EDIT

Selector	Description
`.gEdit`	An Edit widget
`.gFill .gEdit`	An Edit widget once the Javascript layout function has finished its processing

---

TEXTEDIT

Selector	Description
`.gTextEdit`	A TextEdit widget
`.gScrollbarHorizontal`	A TextEdit widget having the `SCROLLBARS` attribute set to *horizontal*
`.gScrollbarVertical`	A TextEdit widget having the `SCROLLBARS` attribute set to *vertical*
`.gFill .gTextEdit`	An TextEdit widget once the Javascript layout function has finished its processing

---

BUTTON

Selector	Description
`.gButtonBox`	The `SPAN` tag containing the button
`.gButtonBox .gAction`	The image or text for this button
`.pressedButtonBox`	A pressed button
`.gFill .gButtonBox`	A Button once the Javascript layout function has finished its processing
`.gFill .gHBoxTag .gButtonBox`	A Button in a Grid HBox tag once the Javascript layout function has finished its processing

---

BUTTONEDIT

Selector	Description
.gButtonEdit	A ButtonEdit widget
.gFill .gButtonEdit	An ButtonEdit widget once the JavaScript layout function has finished its processing

---

DATEEDIT

Selector	Description
.gDateEdit	A DateEdit widget
.gFill .gDateEdit	A DateEdit widget once the JavaScript layout function has finished its processing
.calendar	The calendar widget
.calendar THEAD .nav	The calendar widget's navigation bar
.calendar TD	A calendar widget cell
.calendar THEAD .nav .info	The calendar widget's date
.calendar THEAD .days TD	The calendar widget's days bar
.calendar THEAD .days .we	Weekend days in the calendar widget's days bar
.calendar TBODY TD	A calendar widget's day number cell
.calendar TBODY .hover	A calendar widget's day number cell having the mouse over it
.calendar TBODY .today	Today's cell in the calendar widget
.calendarIcon	The DateEdit widget's calendar icon

CHECKBOX

Selector	Description
.gCheckBox .gFormFieldBox .gCheckBox	A CheckBox widget

.gTable TD .gCheckBox	A CheckBox widget in a table
.nullState	A CheckBox widget having no state
.checkedState	A checked CheckBox widget
.uncheckedState	An unchecked CheckBox widget

COMBOBOX

Selector	Description
.gComboBox	A ComboBox widget
.comboboxEdit	The ComboBox input field
.comboboxButton	The ComboBox button
.comboboxList	The ComboBox list of values
.comboboxList DIV	A ComboBox list item
.comboboxList DIV.over	The ComboBox list item having the focus
.comboboxList DIV.selected	The currently selected ComboBox list item
.gCurrentField .comboboxEdit	The ComboBox having the input
.disabled .comboboxEdit	A disabled ComboBox input field
.disabled .comboboxButton	A disabled ComboBox button
.gQuery .comboboxEdit	A ComboBox input in construct mode
.gFill .comboboxEdit	A ComboBox input once the JavaScript layout function has finished its processing
.gFill .gHBoxTag .comboboxEdit	A ComboBox input in a Grid HBox tag once the JavaScript layout function has finished its processing

---

RADIOGROUP

Selector	Description
.gRadioGroup	A RadioGroup widget

`.gOrientationHorizontal DIV`	An option of a RadioGroup widget having its `ORIENTATION` attribute set to *horizontal*

LABEL

Selector	Description
`.gLabel`	A form label
`.gFill .gLabelBox LABEL`	A static Label once the JavaScript layout function has finished its processing
`.gFill .gHBoxTag .gLabelBox LABEL`	A static Label in a Grid HBox tag once the JavaScript layout function has finished its processing

---

Construct

Selector	Description
`INPUT.queryZone`	A formfield in construct mode
`INPUT.currentQueryZone`	The formfield in construct mode having the focus

Dialog CSS

- MENU
- DIALOG
- TOPMENU
- TOOLBAR
- MESSAGE
- ERROR

MENU

Selector	Description
`.gMenu`	The `DIV` tag containing the menu
`.gMenu SPAN`	The menu title
`.gMenu UL`	The menu actions container
`.gMenu LI`	A menu action
`.gMenu LI.hover`	A menu action having the mouse cursor over it

`.gMenu INPUT`	The image or text associated to the menu action
`.gMenu LI.gCurrentAction INPUT`	The image or text for the current menu action
`.gMenu LI.gHidden`	A hidden menu action
`.gStyleDialog IMG`	The image associated to a menu having the `STYLE` attribute set to *Dialog*

DIALOG

Selector	Description
`.gDialog`	The `DIV` tag containing the action panel
`.gDialog UL`	The action panel container
`.gDialog LI`	An action
`.gDialog LI.hover`	An action having the mouse cursor over it
`.gDialog LI.gHidden`	A hidden action
`.gDialog INPUT`	The text associated to the action

TOPMENU

Selector	Description
`.gTopMenu`	The `DIV` tag containing the TopMenu
`.gTopMenu UL`	The container for the TopMenu list of top groups
`.gTopMenu LI`	A TopMenu top group
`.gTopMenu UL UL`	A TopMenu group's items lis
`.gTopMenu LI LI`	A TopMenu group item
`.gTopMenu UL UL UL`	A sub-group's items list
`.gTopMenu label`	The TopMenu text
`.gTopMenu .gAction`	The image or text for a TopMenu command
`.gTopMenu .hover`	The TopMenu item having the mouse cursor over it
`.gTopMenu HR`	A TopMenu separator
`.gTopMenu LI.gHidden`	A hidden TopMenu item

TOOLBAR

Selector	Description
`.gToolBar`	The `DIV` tag containing the ToolBar
`.gToolBar UL`	The container for the ToolBar items
`.gToolBar LI`	A ToolBar item
`.gToolBar HR`	A ToolBar separator
`.gToolBar INPUT`	The image or text for a ToolBar item
`.gToolBar .hover`	The ToolBar item having the mouse cursor over it
`.gToolBar .hover *`	The image or text for a ToolBar item having the mouse cursor over it
`.gToolBar .pressed`	A pressed ToolBar item
`.gToolBar LI.gHidden`	A hidden ToolBar item

MESSAGE

Selector	Description
`#gMessage`	The `P` tag containing the message

ERROR

Selector	Description
`#gError`	The `P` tag containing the error message

Other CSS

Selector	Description
`#gDialogForm`	The HTML form containing the workspace
`#gForm`	The container of the application's current form
`#gForm .gHidden`	Any hidden elements in the form
`#gFormTable`	The `TABLE` tag containing the form
`#gForm-div`	The `DIV` tag containing the form
`#gPanel`	The `TD` tag containing the action panel

`BODY`	The `BODY` tag of the page
`.defaultButton`	An image for which the resource has not been found
`.disabled` `.disabled OPTION`	A disabled element of the application
`.gCurrentCell` `.disabled`	A disabled element of a screen array
`SELECT.gCurrentField`	The ComboBox widget having the focus
`.gStretchX`	An Image widget having the `STRETCH` attribute set to *X*
`.gStretchY`	An Image widget having the `STRETCH` attribute set to *Y*
`.gAutoScale`	An Image widget having the `AUTOSCALE` attribute set\
`.gHLineBox HR`	A horizontal line
`.gFill .gHLineBox HR`	A horizontal line once the JavaScript layout function has finished its processing
`.gFill .gHBoxTag` `.gHLineBox HR`	A horizontal line in a Grid HBox tag once the JavaScript layout function has finished its processing

Customize the User Interface with Templates and Snippets

The HTML that is sent to the browser is dependant on the templates and snippets used by the rendering engine. To change the HTML that is generated, you can change the template and snippets that produce the HTML.

Topics

- Writing a Custom Snippet
- Example of Snippet Customization
- Using your Custom Snippet
- Do I have to restart the GAS to see the result of changes in my template or snippet files?

We assume that you have read the section How the GWC Uses Web Technologies and understand the principles. This section describes how you modify the default GWC rendering to customize your Web application. Understanding this chapter requires knowledge of CSS and JavaScript. Refer to the tutorials at http://www.w3schools.com to get an overview of these technologies.

Writing a Custom Snippet

To change how a specific interface object is rendered by the snippet-based rendering engine, you modify the snippet itself.

To do this, you need to:

1. Identify the snippet that is associated with the interface object that you wish to modify.
2. Identify the language that the snippet is written in. We know that the snippets have template language (for that provides the path to the interface object(s), but the snippets may also include JavaScript and HTML (as with the AJAX theme - Set 1) or may simply be in HTML (as with the PDA theme - Set 3).
3. Create a copy of the snippet and make your modifications to the copy. Do not modify the snippet that comes installed with the product. If you modify the snippets provided with the product, you will lose your work when you upgrade the product. You can place the copy in a directory of your choosing; when you define the resource for your custom snippet you will be able to specify the path to the snippet file. See Using your Custom Snippet for more information.

Warning! By default, the Genero Application Server caches the template and snippet files; you must either restart the application server to use the latest template and snippet files, or you must start the application server using the --development option. See Installing and Starting the GWC for more information.

Modify a Snippet Example

In this example, the Image snippet is being modified to display differently depending on whether the file being displayed is a Flash file (with a .swf extension) or another type of image file.

Example AJAX Mode (Set 1 Snippets)

For AJAX Mode, we want to customize the Image snippet file so that it uses an HTML object tag instead of an image tag, as the object tag can handle the SWF. We then want to provide conditional code: if it is a Flash File, do *x,* if it is not a Flash File, do *y*.

Before Customization

```
01  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02  <html xmlns:gwc="http://www.4js.com/GWC">
03
04  <!-- the head element is ignored -->
05  <head>
06    <meta http-equiv="Content-Type" content="text/html; charset=utf-
8" />
07    <title>Image snippet</title>
08  </head>
09  <body>
10    <!-- the template snippet is the content of the gwc:snippet-root
element -->
11    <gwc:snippet-root>
12      <img
13        gwc:condition="hidden!=1"
14        gwc:attributes="
15          src value ? ImageURI(value) : null; title comment;
16          class (hidden!=2?'':' gHidden');
17        "
18      />
19    </gwc:snippet-root>
20  </body>
21  </html>
```

After Customization

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
03  <html xmlns:gwc="http://www.4js.com/GWC">
04
05  <!-- the head element is ignored -->
06  <head>
07    <meta http-equiv="Content-Type" content="text/html; charset=utf-
8" />
08    <title>Image snippet</title>
```

```
09  </head>
10  <body>
11    <!-- the template snippet is the content of the gwc:snippet-root
element -->
12    <gwc:snippet-root>
13      <img
14        gwc:condition="(hidden!=1) &amp;&amp; !contains(value,
'.SWF')"
15        gwc:attributes="src value ? ImageURI(value) : null; title
comment;class
            (hidden!=2?'':' gHidden');"
16        style="max-width:480px;margin:10px;padding:20px;background-
color:black;"
17      />
18      <object gwc:condition="!hidden &amp;&amp; contains(value,
'.SWF')" width="480px"
          style="margin:10px;">
19        <param name="movie" gwc:attributes="value ImageURI(type ==
'Image' ? value :
          image)"/>
20        <embed width="480px" gwc:attributes="src ImageURI(value)"/>
21      </object>
22
23    </gwc:snippet-root>
24  </body>
25  </html>
```

Notes

1. In Line 14, the gwc:condition instruction handles files that are NOT flash files (files without an SWF extension).
2. In Line 18, the gwc:condition instruction handles files that are flash files (files with an SWF extension).

Example PDA Mode (Set 3 Snippets)

For the PDA example, when a Flash file is found, we simply want the application to inform the user that their device does not support this type of file.

Before Customization

```
01  <?xml version="1.0"?>
02  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
03  <html xmlns:gwc="http://www.4js.com/GWC" xml:lang="fr">
04  <head>
05    <meta http-equiv="Content-Type" content="text/html; charset=utf-
8" />
06    <title>Image snippet</title>
07  </head>
08  <body>
09    <gwc:snippet-root>
10      <img gwc:condition="!hidden"
11           gwc:attributes="src ImageURI(type == 'Image' ? value :
```

```
image)
12              ;width (autoscale ? (contains(width, 'px') ? width :
number(width)*6 ) : null)
13              ;height (autoscale ? (contains(height, 'px') ? height :
number(width)*10 ) : null)
14              ;title comment
15              ;alt comment
16              ;class 'gImage gStyle_'+translate(class, ' -/\#.=+()[]',
'____________') +
17                (' gColor_'+style['textColor']) +
18                (tag ? ' gTag_'+translate(tag, ' -/\#.=+()[]',
'____________') : '')"/>
19    </gwc:snippet-root>
20  </body>
21  </html>
```

After Customization

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
03  <html xmlns:gwc="http://www.4js.com/GWC" xml:lang="fr">
04  <head>
05    <meta http-equiv="Content-Type" content="text/html; charset=utf-
8" />
06    <title>Image snippet</title>
07  </head>
08  <body>
09    <gwc:snippet-root>
10      <img gwc:condition="!hidden &amp;&amp; !contains(value,
'.SWF')"
11        width="200px"
12        gwc:attributes="src ImageURI(type == 'Image' ? value : image)
13          ;title comment
14          ;alt comment
15          ;class 'gImage gStyle_'+translate(style, ' -/\#.=+()[]',
'____________') +
16            (color ? ' gColor_'+color:'') +
17            (tag ? ' gTag_'+translate(tag, ' -/\#.=+()[]',
'____________') : '')"/>
18      <div style="border:1px solid orange;
19                  background-color:#F4F4F4;
20                  text-align:center;"
21           gwc:condition="!hidden &amp;&amp; contains(value,
'.SWF')">
22                  This animation cannot be displayed<br/>
23                  on this device.</div>
24    </gwc:snippet-root>
25  </body>
26  </html>
```

Notes

1. In Line 10, the gwc:condition instruction handles files that are NOT flash files (files without an SWF extension).

2. In Line 21, the gwc:condition instruction handles files that are flash files (files with an SWF extension).

Using your Custom Snippet

Once you have customized a snippet, you then need to make the snippet available.

- Specify a custom snippet for a specific application
- Specify a snippet for all applications

Specify a customized snippet for a specific application

If you only want to apply your customized snippet on a per application basis, you would override the resources for the snippets within the application's configuration.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <APPLICATION Parent="defaultgwc">
03    <EXECUTION>
04      <PATH>/home/srcdir/app1</PATH>
05      <MODULE>app1</MODULE>
06    </EXECUTION>
07    <OUTPUT>
08      <MAP Id="DUA_AJAX">
09        <THEME>
10          <SNIPPET
Id="Image">/home/srcdir/app1/Snippets/Set1_Image.xhtml</SNIPPET>
11        </THEME>
12      </MAP>
13      <MAP Id="DUA_PDA>
14        <THEME>
15          <SNIPPET
Id="Image">/home/srcdir/app1/Snippets/Set3_Image.xhtml</SNIPPET>
16        </THEME>
17      </MAP>
18    </OUTPUT>
19  </APPLICATION>
```

Specify a customized snippet across all applications

If you want to have your customized snippet apply across all of your applications, then you want it to be part of the defined sets for the various application themes (AJAX, PAGE, PDA).

You have three options:

Option 1: You can alter how the snippet is defined within each of the relevant WEB_APPLICATION_THEME_COMPONENTS

```
01   ...
02  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.ajax.gwc">
03       ...
04    <SNIPPET
Id="RadioGroup">$(res.path.tpl.ajax)/RadioGroup.xhtml</SNIPPET>
05    <SNIPPET
Id="Image">/home/srcdir/app1/Snippets/Set1_Image.xhtml</SNIPPET>
06    <SNIPPET
Id="StaticLabel">$(res.path.tpl.ajax)/StaticLabel.xhtml</SNIPPET>
07       ...
08  </WEB_APPLICATION_THEME_COMPONENT>
09   ...
10  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.pda.gwc">
11       ...
12    <SNIPPET
Id="RadioGroup">$(res.path.tpl.pda)/RadioGroup.xhtml</SNIPPET>
13    <SNIPPET
Id="Image">/home/srcdir/app1/Snippets/Set3_Image.xhtml</SNIPPET>
14    <SNIPPET
Id="StaticLabel">$(res.path.tpl.pda)/Label.xhtml</SNIPPET>
15       ...
16  </WEB_APPLICATION_THEME_COMPONENT>
17   ...
```

Option 2: You can create new WEB_APPLICATION_THEME_COMPONENTS, then modify the auda.xrd to reference these components. To create new WEB_APPLICATION_THEME_COMPONENTS, you would copy and paste the content of the existing components and provide your copies with new names.

```
01  ...
02  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.ajax.gwc">
03    ...
04    <SNIPPET
Id="RadioGroup">$(res.path.tpl.ajax)/RadioGroup.xhtml</SNIPPET>
05    <SNIPPET Id="Image">$(res.path.tpl.ajax)/Image.xhtml</SNIPPET>
06    <SNIPPET
Id="StaticLabel">$(res.path.tpl.ajax)/StaticLabel.xhtml</SNIPPET>
07    ...
08  </WEB_APPLICATION_THEME_COMPONENT>

09  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.ajax2.gwc">
10    ...
11    <SNIPPET
Id="RadioGroup">$(res.path.tpl.ajax)/RadioGroup.xhtml</SNIPPET>
12    <SNIPPET
Id="Image">/home/srcdir/app1/Snippets/Set1_Image.xhtml</SNIPPET>
13    <SNIPPET
Id="StaticLabel">$(res.path.tpl.ajax)/StaticLabel.xhtml</SNIPPET>
14    ...
15  </WEB_APPLICATION_THEME_COMPONENT>

16  ...
17  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.pda.gwc">
18    ...
19    <SNIPPET
Id="RadioGroup">$(res.path.tpl.pda)/RadioGroup.xhtml</SNIPPET>
```

```
20    <SNIPPET Id="Image">$(res.path.tpl.pda)/Image.xhtml</SNIPPET>
21    <SNIPPET
Id="StaticLabel">$(res.path.tpl.pda)/Label.xhtml</SNIPPET>
22    ...
23  </WEB_APPLICATION_THEME_COMPONENT>
24  <WEB_APPLICATION_THEME_COMPONENT Id="cpn.theme.pda2.gwc">
25    ...
26    <SNIPPET
Id="RadioGroup">$(res.path.tpl.pda)/RadioGroup.xhtml</SNIPPET>
27    <SNIPPET
Id="Image">/home/srcdir/app1/Snippets/Set3_Image.xhtml</SNIPPET>
28    <SNIPPET
Id="StaticLabel">$(res.path.tpl.pda)/Label.xhtml</SNIPPET>
29    ...
30  </WEB_APPLICATION_THEME_COMPONENT>
31  ...
```

Option 3: You could replace the image snippet files in the respective template directories. For this solution, you would name your custom snippet file the same name as the snippet file provided during the installation.

If you choose either solution 1 or 2, when you upgrade the GWC you will have to modify the GAS configuration file to re-apply your changes. If you choose solution 3, you will have to replace the relevant snippet files with your custom versions each time you do an upgrade.

Do I have to restart the GAS to see the result of changes in my template or snippet files?

By default, the Genero Application Server daemon (gasd) keeps the template and snippet files in its cache. When developers make changes to a template or snippet file, the change is not recognized until the Genero Application Server daemon is restarted.

To change this default behavior, you can specify the `--development` flag. This flag is only valid for the Genero Application Server provided with the GWC install package.

For more information on starting the Genero Application Server daemon and the various command options, please refer to the *Genero Application Server Manual*.

Customize the User Interface with JavaScript

This section discusses the implementation of the user interface with JavaScript. We assume that you have read the section How the GWC Uses Web Technologies and understand the principles. This help topic will not teach you how to write JavaScript code. For an introduction to working with JavaScript, refer to the Learn JavaScript tutorial at http://www.w3schools.com.

Topics:

- Client-Side Framework (CSF)
- gwc:marks template instruction
- Reserved functions
- Component styles
- Client-side framework API
- Toolkit API

Warning! We recommend leaving the default JavaScript files unchanged; Modifying default JavaScript files is not supported. To add your own JavaScript, create a new JavaScript file and reference it in the main template.

Client-Side Framework (CSF)

The client-side framework (CSF) is the part of the GWC that runs on the client side (browser) when in AJAX mode.

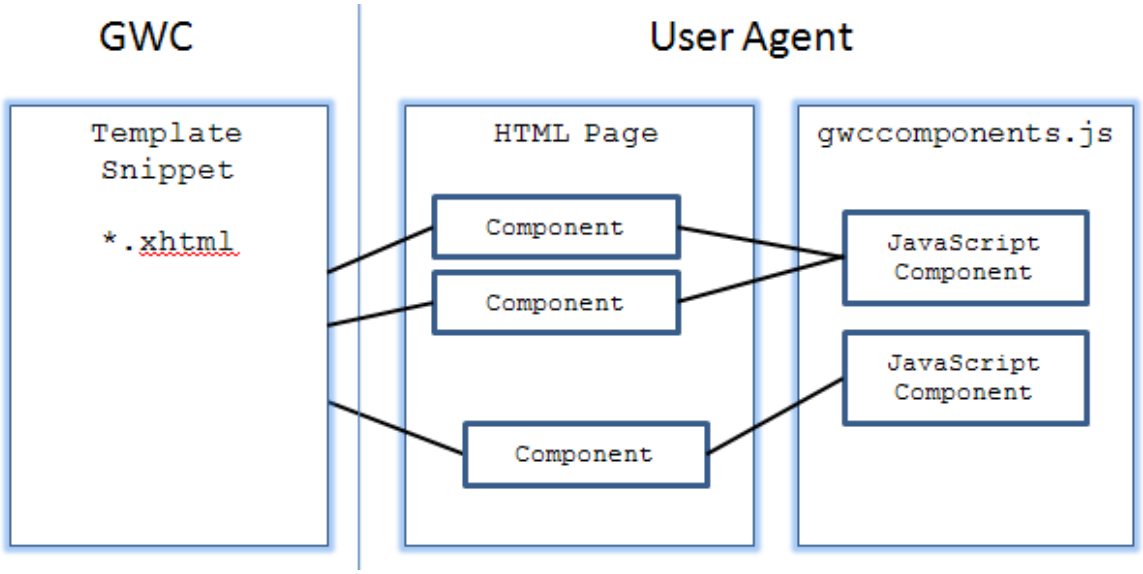
The primary goal of the CSF is to manage application life. The primary goals of the client-side JavaScript is to provide objects that allow scripts to interact with the user, control the web browser, and alter the document content that appears within the web browser window. JavaScript eases the interaction with users and refines the application's design.

---

gwc:marks template instruction

The `gwc:marks` template instruction is a convenient way to identify an HTML element, send data, or send events from GWC template snippets to the client-side framework (CSF).

Data in the `gwc:marks` template instruction is sent in a JavaScript structure, not in the xHTML document.



The `gwc:marks` template instruction is a generic tool provided by the GWC. The CSF makes its own usage of this tool.

Syntax

```
<tag gwc:marks="mkp expr [; ...] " ...>
...
</tag>
```

Notes:

1. *mkp* is the name of the mark
2. *expr* is the data sent to the CSF

Relating Components to JavaScript Objects

Each component that has a CID template path has a related entry in the `gwc.componentSet` in gwccomponents.js. For example, the Edit component (defined in the template snippet Edit.xhtml) is related to the object `gwc.componentSet.Edit` (defined in gwccomponents.js).

Providing the expression

When the `gwc:marks` instruction is used, the CSF expects either a component identifier (CID) with data, or null.

Because the CSF needs to link the gwc:marks instruction to the right GWC JavaScript component in gwccomponents.js, the component identifier (CID) must be provided with the data. The CID is the unique identifier of an instance of a component in the HTML page. Because of this, you need to use an array within the expression, with the CID at the first place. For example:

```
01 gwc:marks="mkp [CID, data1, ..., dataN]"
```

The other allowed value as expression is null. This informs the CSF to remove the mark.

```
01 gwc:marks="mkp null"
```

To send data from a template path to a JavaScript function, use this:

```
01 gwc:marks="currentFieldStyle [CID,'gCurrent'+type]"
```

When a mark is created or removed, the related function is called in gwcComponents.js. The call can be treated as an event.

For example, the following shows how to use the marker as an event only (without any data).

```
01 gwc:marks="modifiable isModifiable ? [CID] : null"
```

The JavaScript function related to the mark name is called when the page is loaded the first time, and subsequently when the expression is changing.

On the JavaScript side

On the JavaScript side, the data is stored in a JavaScript Array object, and the function with the same name as the mark is called.

The function looks like this: `function( polarity, eid, CID, component, data )`

1. *polarity* is the reason why the function is called. polarity is true if the mark is added and false if the mark is removed (data of the mark is null).
2. *eid* is the HTML id of the element where the gwc:mark is placed on.
3. *CID* is the component identifier of the template snippet that defined the mark.
4. *component* is the JavaScript object directly related to the component. This object can be used to store temporary data related to the component.
5. *data* is the expression that is provided to the mark.

For example, consider the following hypothetical `gwc:marks` instruction.

```
01 gwc:marks="someTableInfo [CID,offset,pageSize,size];
```

This instruction, when placed in the snippet Table.xhtml, will cause the CSF to call the following function in gwccomponents.js:

```
01 gwc.componentSet.Table = {
02
03   someTableInfo: function( polarity, eid, cid, component, data ) {
04
05     var offset = data[1];
06     var pageSize = data[2];
07     var size = data[3];
```

```
08      ...
09      }
10   }
```

Notice that the mark name and the function prefix name (`someTableInfo`) match.

Note:

1. When the expression is changing, the related JavaScript function is called twice: the first time with polarity == false, and the second time with polarity == true (as if it was removed and then added).

Reserved Functions

Some optional reserved function of the component can be called by the client-side framework (CSF). These functions should not be used as a mark name.

Reserved Function	Description
`GetValue( component )`	This function is called by the CSF when it needs to know the current value of the component. The function returns the current value of the component, or undefined (a JavaScript keyword) if there is no value.
`currentField( polarity, component )`	This function is called by the CSF when the field gets or loses the focus. The polarity argument is true if the component *receives* the focus, and the polarity argument is false if the component *loses* the focus.
`currentCell( polarity, component )`	This function is called by the CSF when the matrix cell gains or loses the focus. The polarity argument is true if the matrix cell *receives* the focus, and the polarity argument is false if the matrix cell *loses* the focus.
`currentTable( polarity, component )`	This function is called by the CSF when the table is made current. The polarity argument is true if the table *receives* the focus, and the polarity argument is false if the table *loses* the focus.
`currentRow( polarity, component, rowIndex )`	This function is called by the CSF when the row of the table having focus has changed. If polarity is false, rowIndex is the index of the row that *loses* the focus. If polarity is true, rowIndex is the index of the row that *receives* the focus.

Example:

```
01 gwc.componentSet.Edit = {
02   ...
```

```
03   GetValue:function( component ) {
04
05     var eid = gwc.core.state.FirstMarkEid( component.cid, 'field' );
06     if ( eid != undefined )
07       return gwc.tk.IdToElement(eid).value;
08   }
09   ...
```

Component Styles

It is possible to select a different JavaScript component according to the widget's style specified in the form definition file (.PER or .4FD).

If a style is defined, the name of the component is: `componentName + '_' + styleName`

Note:

1. The function that creates the final component name is at the top of gwccomponents.js

Example:

In the .PER file, the ATTRIBUTES section defines an EDIT field as follows:

```
01 ...
02 ATTRIBUTES
03 EDIT edit1 = formonly.edit1, STYLE = "FileUpload";
04 ...
```

In the gwccomponents.js file, the component is:

```
01 gwc.componentSet.Edit_FileUpload = {
02 ...
```

This EDIT widget is transformed to define the file upload widget.

Client-Side Framework API

The client-server framework (CSF) API provides a set of functions to manage links between marks and components.

The following list is a subset of the available functions.

Reserved Function and variables	Description
`gwc.core.state.FirstMarkEid( CID, markName )`	This function returns the HTML id of the first element of the component CID that has the markName name.
`gwc.core.state.FirstMarkData( CID, markName )`	This function returns the mark data (the JavaScript Array) of the component CID that has the markName name.
`gwc.core.state.MarkData( CID, markName, eid )`	This function returns the only mark data based that has CID as component identifier, markName as mark name and eid as HTML element id.
`gwc.core.state.ComponentByCid( CID )`	This function returns the component object that has CID as identifier.
`gwc.capi.SessionVar( varName, varValue )`	This function set the session variable named varName with value varValue.
`gwc.cfg.localeDateStrings`	If the entry gwc.cfg.localeDateStrings is not defined (see snippet main.xhtml), the CSF tries to detect automatically the name of the days and months in the current browser locale.

ToolKit API

The toolkit API is a collection of general purpose JavaScript functions. The purpose of this toolkit is to provide browser-independent functions. These functions are stored in `gwc.tk.*`

Note:

1. These functions change according to the needs of the client-server framework. As a result, it is not possible to exhaustively list them.

Example:

This example sets the CSS class 'myRedColor' on the HTML element that has the id 'myId'

```
01 gwc.tk.AddClass( gwc.tk.IdToElement('myId'), 'myRedColor' );
```

Front End Protocol

The Front End Protocol is a set of instructions sent by the User Agent to the Genero Application Server.

Summary

- Exclusive/Inclusive forms
- Usage

Exclusive/Inclusive forms

User Agents send form data to form data processing agents as a sequence of *control-name* and *control-value*. For example, the browser sends pairs of *name=value* to the Genero Application Server. Most controls have names but the value submitted is not always the one expected. Inclusive form is introduced to workaround the default behaviour.

Syntax

Exclusive form

```
control-name=control-value
```
that is
```
<prefix> [ container ] = [ [ ss/prefix ] <value> ]
```

Inclusive form

```
control-name/control-value=
```
that is
```
<prefix> [ container ] [ / [ ss/prefix ] <value> ] =
```

Where

- *prefix* is a character to identify the type of action.
- *container* is the component identifier.
- *ss* is a subprefix.
- *value* is the value expected by the Genero Application Server which is not always the default value that a User Agent sends.

To build the control-name use the template paths `IDID` (inclusive form) or `XDID` (exclusive form).
Facilities are available as functions in the Front End Protocol Functions section.

Example

Exclusive form

```
<input type="radio" gwc:attributes="name XDID; value ID;... />
```
produces
```
<input type="radio" name="a" value="89"... />
```

Inclusive form

```
<input type="submit" gwc:attributes="name IDID"... />
```
produces
```
<input type="submit" name="a/89"... />
```

Usage

The Front End Protocol is used to enhance user agents that have "poor" interactivity like Personal Digital Assistants.

Example of a ComboBox value selection helper
A ComboBox (<select> html widget) is redesigned as a set of buttons (input of type submit). The Select widget is submitted in exclusive form. The Button has a name but the value represents the displayed text of the button and not the value expected from a ComboBox.

```
<p gwc:omit-tag="true" gwc:repeat="cb_item items">
  <input type="submit" gwc:attributes="disabled
isModifiable?NULL:'disabled'; name makeValueIDID(id,cb_item/name);
value cb_item/text"/>
</p>
```

Add this sample in the $FGLASDIR/set2/ComboBox.xhtml. Instead of selecting the ComboBox value, click on the button to select a value in the ComboBox list.

Tutorial - Working with the Genero Web Client

This tutorial provides you with some hands-on experience in running and customizing a Web application delivered by the Genero Web Client front-end.

Topics

Step 0: Run the tutorial demo application

The Genero Application Server installs with a demo application known as the Contact List (Card Manager) application. The default configuration provided in the GAS configuration file allows you to run this demo application without any additional configuration.

To run the demo application, start the GAS and then enter the following URL in a browser:

http://localhost:6394/wa/r/demo/Card

Note: For the purpose of this tutorial, it is assumed that you are connecting to the GAS directly. If your browser is on a different host than the GAS, you will have to replace "localhost" with the name or IP address of the GAS host. Likewise, if you have changed the port number from the installation default of 6394, you will have to modify the port number accordingly.

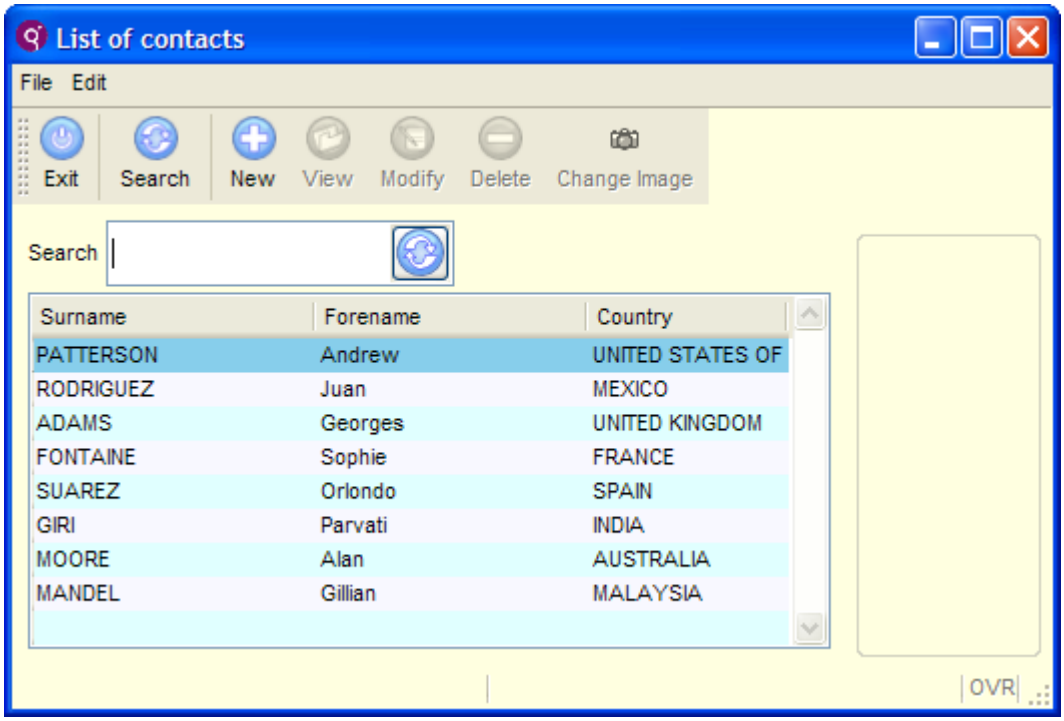


Figure 0-1: The application displayed with the Genero Desktop Client

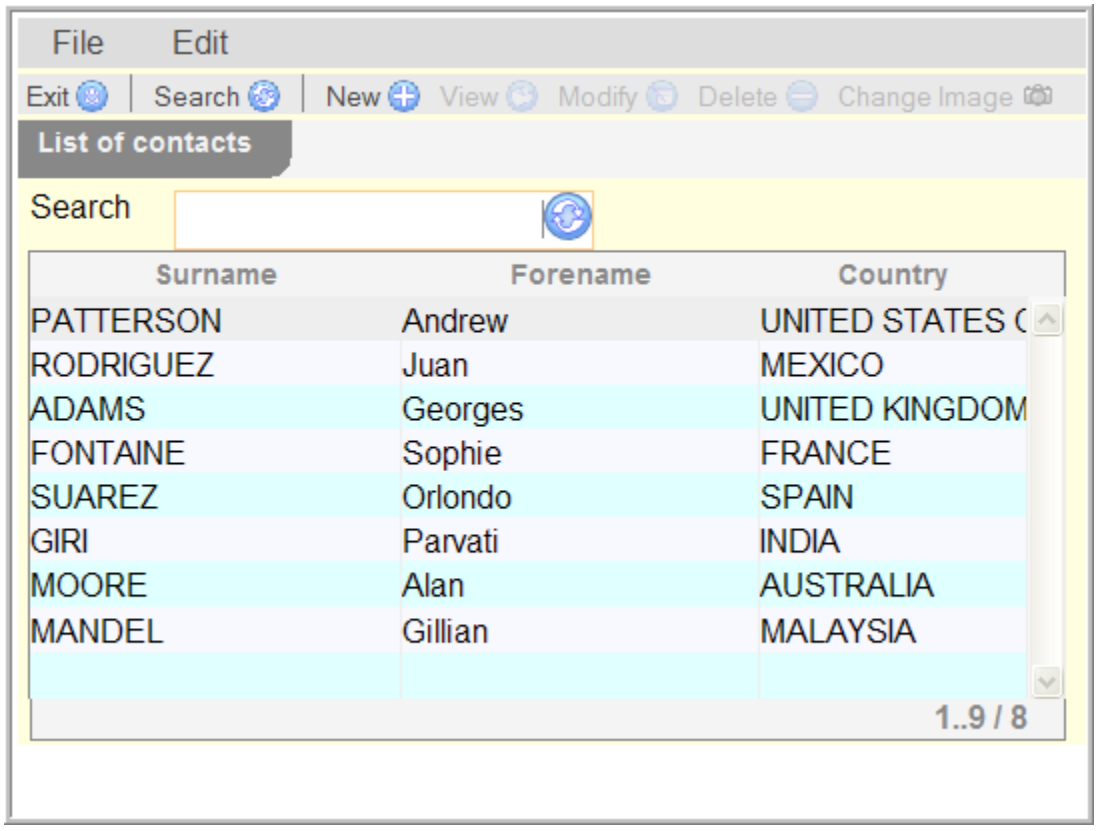


Figure 0-2: The application displayed by the GAS with the Genero Web Client

Things to observe:

- Colors are kept (they match the colors of the application delivered in GDC)
- The action panel positioning is different from where it sits with the GDC. The GDC has the action panel along the right-hand side of the application, whereas the GWC displays the action panel across the top of the application window, under the toolbar.

Card.xcf

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.x
sd">
 <EXECUTION>
  <PATH>$(res.path.demo.app)/card/src</PATH>
  <MODULE>card.42r</MODULE>
 </EXECUTION>
</APPLICATION>
```

Step 1: Use a custom snippet file

For the next part of this tutorial, a custom snippet file has been created to display image widgets for this application.

Enter the following URL in a browser:

http://localhost:6394/wa/r/demo/CardStep1

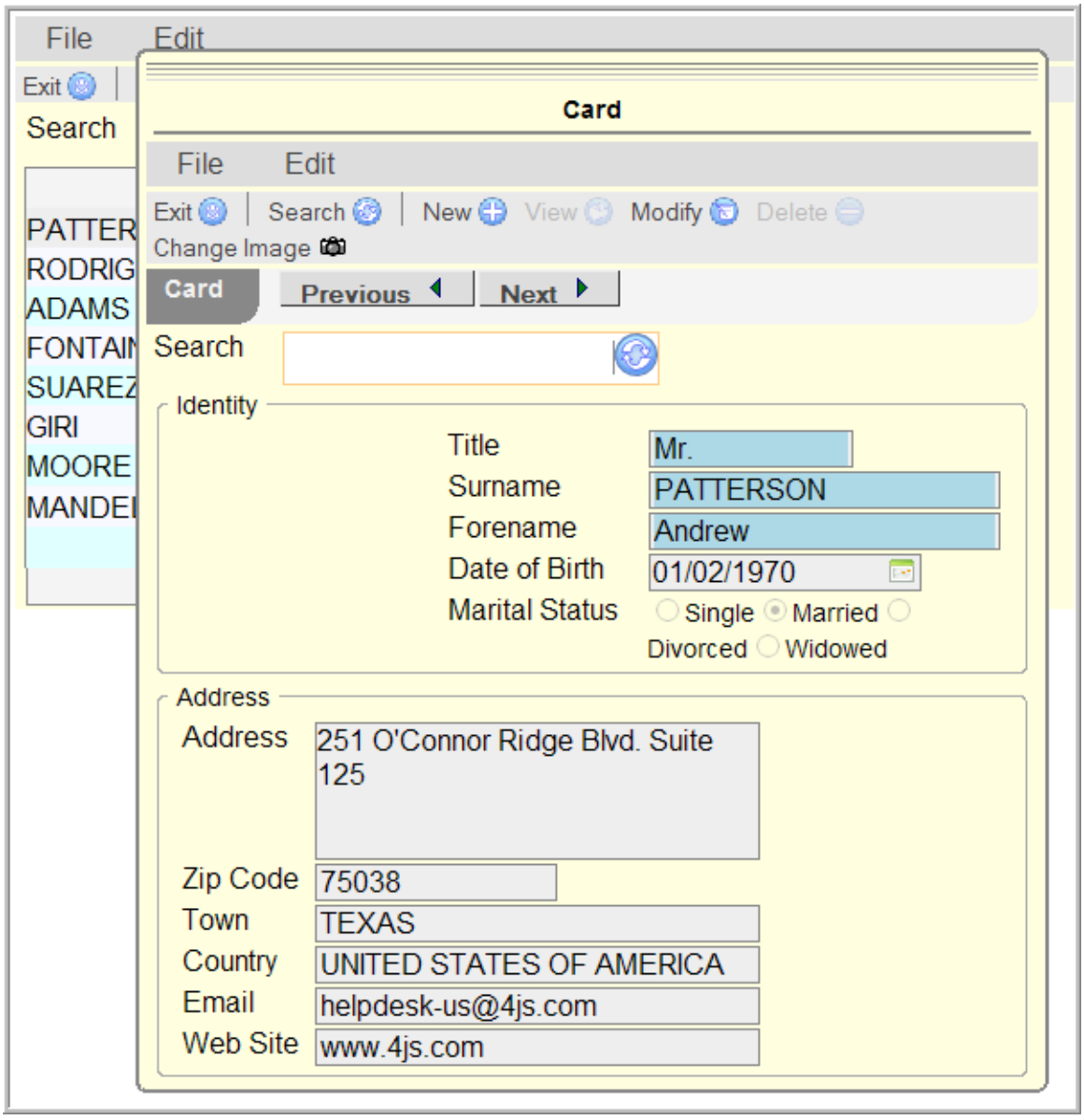


Figure 1-1: Contact displayed prior to referencing the custom Image snippet in the application configuration file.

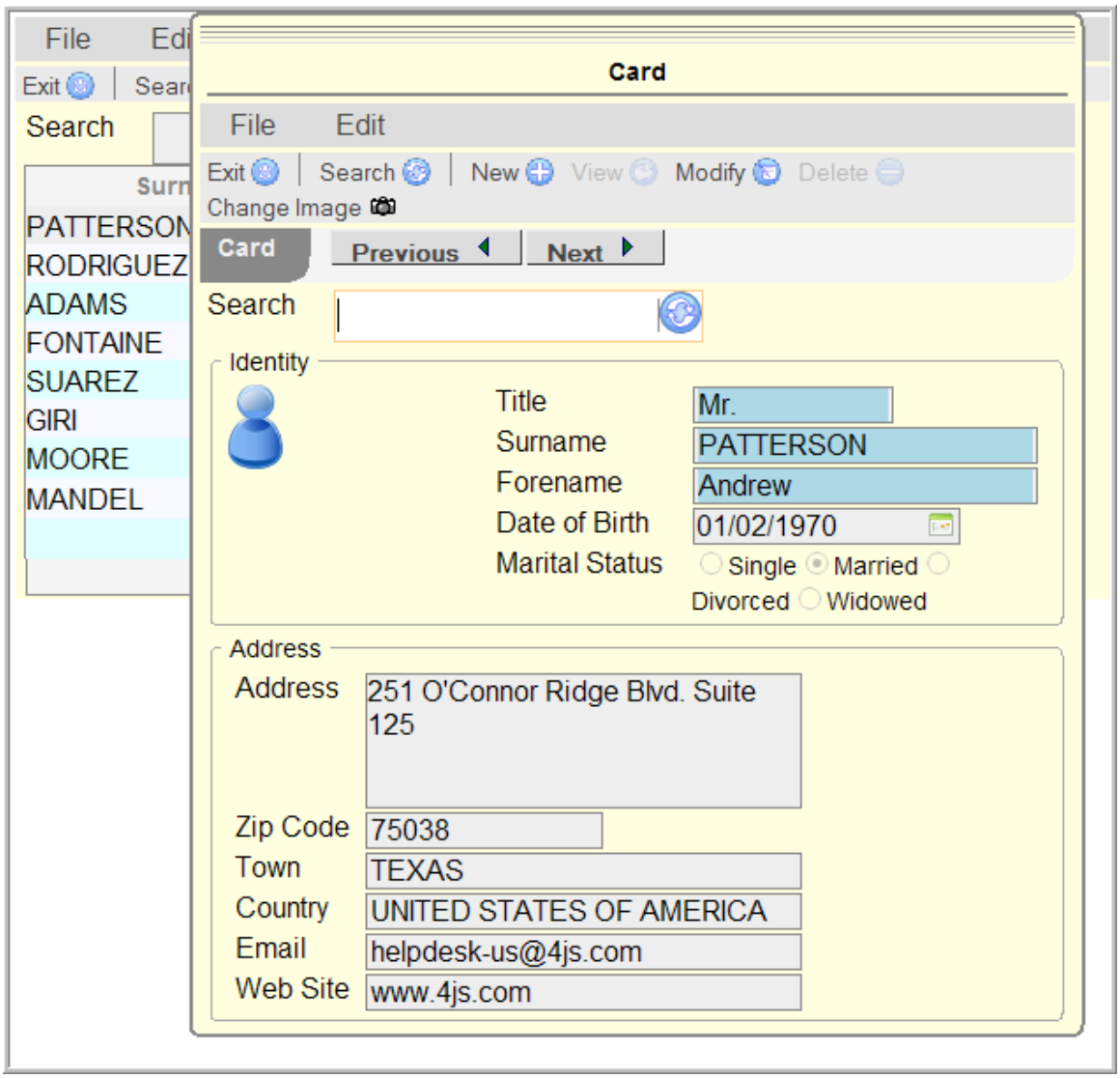


Figure 1-2: Contact displayed after referencing the custom Image snippet in the application configuration file.

Thing to observe:

- An image is now displayed in the appropriate spot on the form.

Application Configuration File: CardStep1.xcf

Examine the application configuration file for this step in the tutorial.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.x
sd">
```

```
 <EXECUTION>
  <PATH>$(res.path.demo.app)/card/src</PATH>
  <MODULE>card.42r</MODULE>
 </EXECUTION>
 <OUTPUT>
  <MAP Id="DUA_AJAX">
   <THEME>
    <SNIPPET Id="Image"
Style="Picture">$(res.path.demo.app)/card/tpl/set1/Image.xhtml</SNIPPET
>
   </THEME>
  </MAP>
  <MAP Id="DUA_AJAX_HTML">
   <THEME>
    <SNIPPET Id="Image"
Style="Picture">$(res.path.demo.app)/card/tpl/set1/Image.xhtml</SNIPPET
>
   </THEME>
  </MAP>
 </OUTPUT>
</APPLICATION>
```

Notes:

- The additional configuration elements (OUPUT, MAP, THEME, SNIPPET) do the following:
  For the Output Map DUA_AJAX, the snippet to be used for the Image widget is defined by the entry in the application configuration file. For all other snippets, the entries defined in the as.xcf file for the DUA_AJAX Output Map are used. The same configuration is done for DUA_AJAX_HTML. For the rest of the document, it is considered that DUA_AJAX_HTML has the same configuration of snippets than DUA_AJAX.

Modifications to the template and snippet files

A new snippet file has been created for the rendering of the Image widgets for this application.

To identify what is different, you can compare the Image.xhtml file provided with the default AJAX theme ($FGLASDIR/tpl/set1/Image.xhtml) with the snippet file specified by the SNIPPET element in the application configuration file. Using Genero Studio's Graphical Differential tool, we can quickly identify the differences between the two files, as shown in Figure 1-3 below:

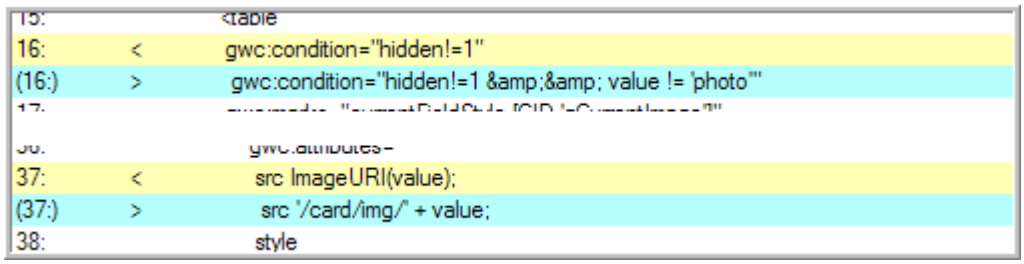


Figure 1-3: The differences between the default and custom Image snippet files, with the customized values shown in blue.

Notes:

- In Line 16, an additional condition is being checked for: the value of the formfield should not be "photo"
- In Line 37, the src tag is modified to specify a different Image location: the /card/img alias, which is defined in the as.xcf file:
  `<ALIAS Id="/card/img">$(res.path.demo.app)/card/src/photo</ALIAS>`

Usually, application pictures are configured through the picture component. In this example, the common pictures are handled by the picture component, alias "/pic", and the application images are served through alias "/card/img". Another solution could be to copy all the common picture in the application image directory and set the component picture path to alias /card/img.

```
<PICTURE>
  <PATH>$(connector.uri)/card/img</PATH>
</PICTURE>
```

In this case you do not need to customize your image snippet file.

Step 2: Customize the template and add additional CSS

For the next step in this tutorial, a header and footer have been added.

Enter the following URL in a browser:

http://localhost:6394/wa/r/demo/CardStep2

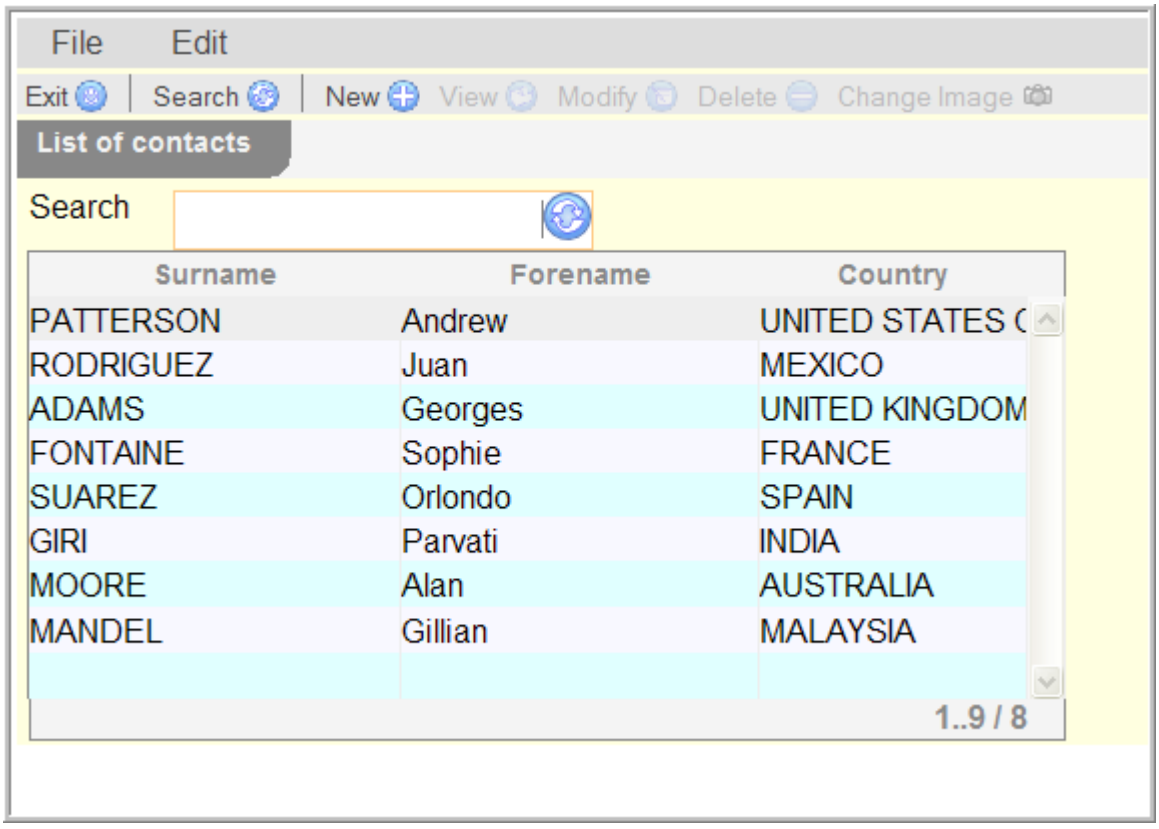


Figure 2-1: Application prior to adding header and footer.

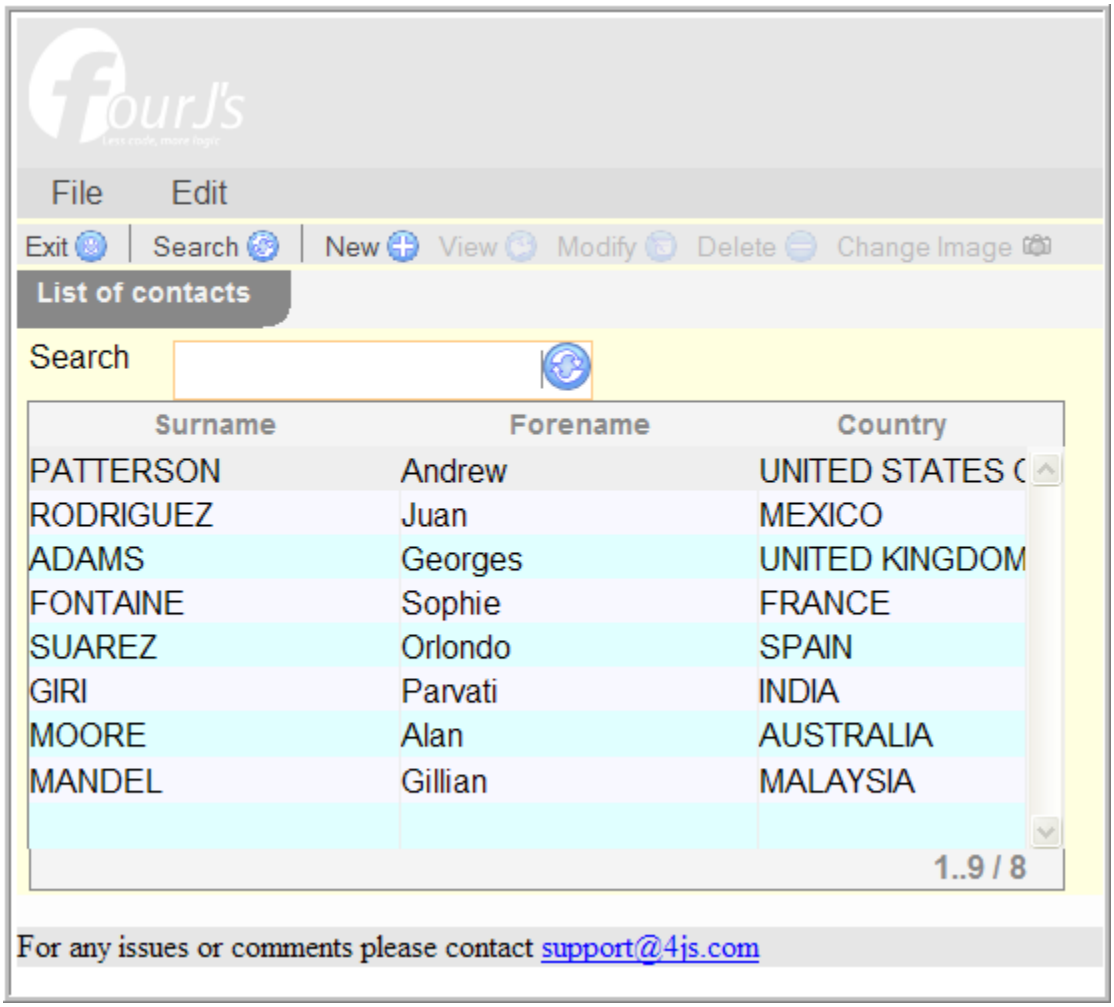


Figure 2-2: Application after adding header and footer.

Things to observe:

- Both a header and a footer have been added.
- The formatting of the look-and-feel of the header and footer is controlled by the entries provided in the latest CSS file added to the application's template.

Application Configuration File: CardStep2.xcf

Examine the application configuration file for this step in the tutorial.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.x
sd">
 <EXECUTION>
  <PATH>$(res.path.demo.app)/card/src</PATH>
```

```
  <MODULE>card.42r</MODULE>
 </EXECUTION>
 <OUTPUT>
  <MAP Id="DUA_AJAX">
   <THEME>
    <TEMPLATE
Id="_default">$(res.path.demo.app)/card/tpl/set1/main.xhtml</TEMPLATE>
    <SNIPPET Id="Image"
Style="Picture">$(res.path.demo.app)/card/tpl/set1/Image.xhtml</SNIPPET
>
   </THEME>
  </MAP>
 </OUTPUT>
</APPLICATION>
```

Notes:

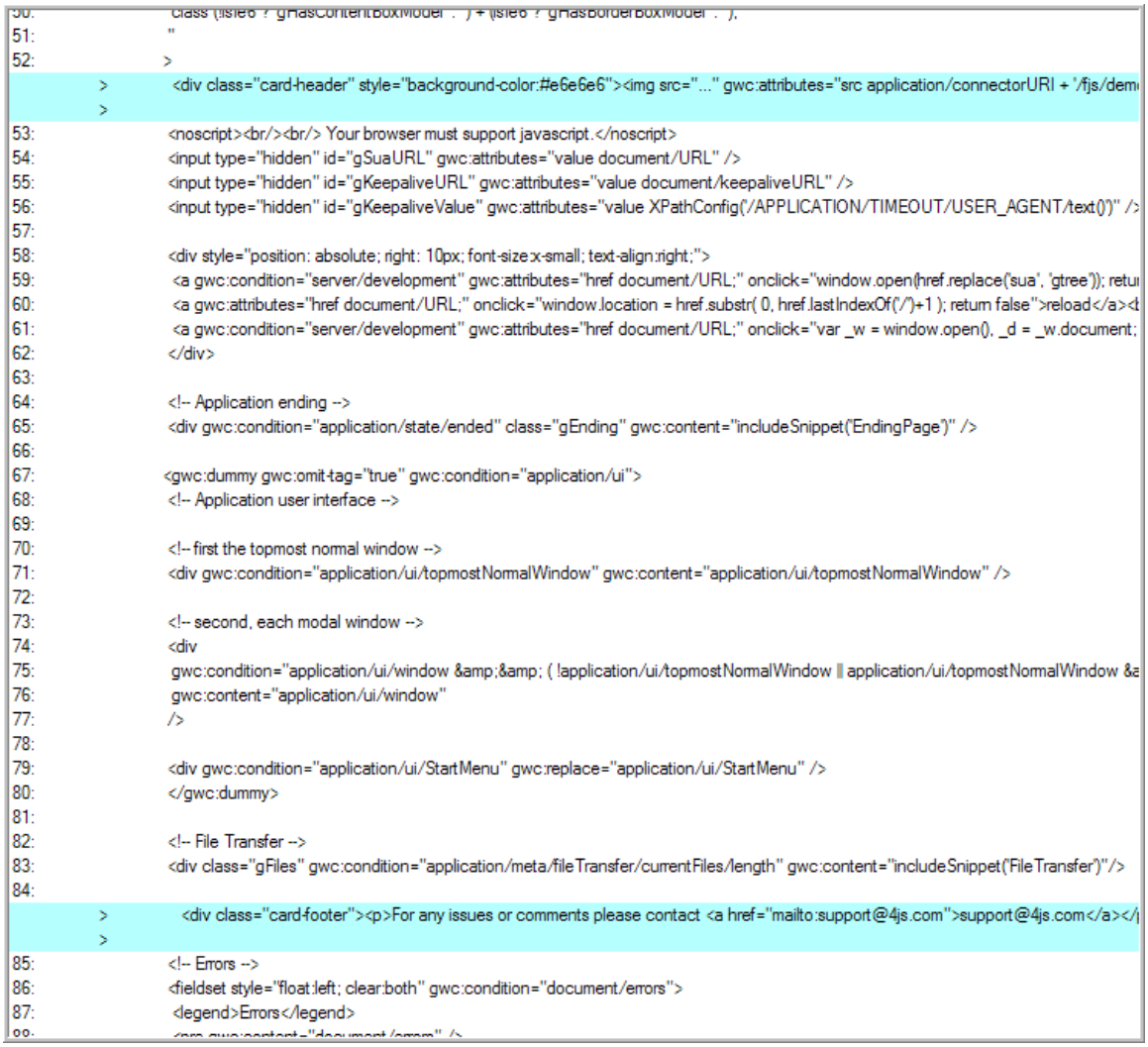
- For the Output Map DUA_AJAX, a custom template has been specified.

Modifications to the template and snippet files

Using Genero Studio's Graphical Differential tool, we can quickly identify the differences between the two files, as shown in Figure 2-3 and 2-4 below:

```
32:             <link rel="stylesheet" gwc:attributes="href application/connectorURI+'/fjs/gwccomponents.css"' type="text/css" title="Default Theme"
33:             <link rel="shortcut icon" gwc:attributes="href application/connectorURI+'/favicon.ico"' type="image/x-icon" />
34:             <script type="text/javascript" gwc:content="'var gDialog=' + document/dialog" />
35:             <script type="text/javascript" gwc:content="'var gElementMarkTree=' + document/marks" />
36:             <script type="text/javascript" gwc:content="'var gComponentList=' + document/components" />
37:             <style type="text/css" id="gStyleList"/>
          >      <link rel="stylesheet" gwc:attributes="href application/connectorURI+'/fjs/demo/card.css"' type="text/css" title="Default Theme"/>
          >      <script type="text/javascript" gwc:attributes="src application/connectorURI+'/fjs/demo/card.js"'> </script>
          >
38:           </head>
```

Figure 2-3: The addition of the custom CSS, with the customized values shown in blue.



```
50:                 class (!isIe6 ? 'gHasContentBoxModel' : '') + (!isIe6 ? 'gHasBorderBoxModel' : '');
51:                 "
52:                >
                >    <div class="card-header" style="background-color:#e6e6e6"><img src="..." gwc:attributes="src application/connectorURI + '/fjs/dem
                >
53:                <noscript><br/><br/> Your browser must support javascript.</noscript>
54:                <input type="hidden" id="gSuaURL" gwc:attributes="value document/URL" />
55:                <input type="hidden" id="gKeepaliveURL" gwc:attributes="value document/keepaliveURL" />
56:                <input type="hidden" id="gKeepaliveValue" gwc:attributes="value XPathConfig('/APPLICATION/TIMEOUT/USER_AGENT/text()')" />
57:
58:                <div style="position: absolute; right: 10px; font-size:x-small; text-align:right;">
59:                 <a gwc:condition="server/development" gwc:attributes="href document/URL;" onclick="window.open(href.replace('sua', 'gtree')); retu
60:                 <a gwc:attributes="href document/URL;" onclick="window.location = href.substr( 0, href.lastIndexOf('/')+1 ); return false">reload</a><b
61:                 <a gwc:condition="server/development" gwc:attributes="href document/URL;" onclick="var _w = window.open(), _d = _w.document;
62:                </div>
63:
64:                <!-- Application ending -->
65:                <div gwc:condition="application/state/ended" class="gEnding" gwc:content="includeSnippet('EndingPage')" />
66:
67:               <gwc:dummy gwc:omit-tag="true" gwc:condition="application/ui">
68:                <!-- Application user interface -->
69:
70:                <!-- first the topmost normal window -->
71:                <div gwc:condition="application/ui/topmostNormalWindow" gwc:content="application/ui/topmostNormalWindow" />
72:
73:                <!-- second, each modal window -->
74:                <div
75:                 gwc:condition="application/ui/window &amp;amp;amp; ( !application/ui/topmostNormalWindow || application/ui/topmostNormalWindow &a
76:                 gwc:content="application/ui/window"
77:                />
78:
79:                <div gwc:condition="application/ui/StartMenu" gwc:replace="application/ui/StartMenu" />
80:                </gwc:dummy>
81:
82:                <!-- File Transfer -->
83:                <div class="gFiles" gwc:condition="application/meta/fileTransfer/currentFiles/length" gwc:content="includeSnippet('FileTransfer')"/>
84:
                >    <div class="card-footer"><p>For any issues or comments please contact <a href="mailto:support@4js.com">support@4js.com</a></
                >
85:                <!-- Errors -->
86:                <fieldset style="float:left; clear:both" gwc:condition="document/errors">
87:                 <legend>Errors</legend>
88:                 <gwc:content="document/errors" />
```

Figure 2-4: The addition of the header and footer code to the template file, with the customized values shown in blue.

Add a header that displays a logo

To add a header to the application display, the following code is placed in the custom template file:

```
<div class="card-header" style="background-color:#e6e6e6">
<img src="..." gwc:attributes="src application/connectorURI +
'/fjs/demo/card-logo.png'" alt="Genero logo"/>
</div>
```

Notes:

- <IMG SRC="URL"> displays image from the indicated URL
- <IMG SRC="URL" ALT="***"> displays the name / description of image if image not displayed

Add a footer that includes a mail link

To add a footer to the application display, the following code is placed in the custom template file:

```
<div class="card-footer">For any issues or comments please contact
<a href="mailto:support@4js.com">support@4js.com</a></div>
```

Add a custom CSS file

To add a custom CSS file, an entry is made under the existing CSS entry and within the HEAD tags (as shown in Figure 2-3):

```
<link rel="stylesheet" gwc:attributes="href
application/connectorURI+'/fjs/demo/card.css'" type="text/css"
title="Default Theme"/>
```

For this example, we have provided a CSS for you. Before moving to the next step, you can make changes to this CSS file and view the results in the displayed application..

Step 3: Customize Snippet to vary Rendering based on Style

For the next step in this tutorial, various things are added:

First, the FormField snippet has been modified such that all fields where the REQUIRED attribute is set to true (1) are marked on the UI with an asterisk.

Second, customized Edit snippets are created to render Edit widgets based on their assigned styles. If the field is representing an email address, that field is assigned the style of "Email", and a custom Edit snippet is called to render the field as a link that, when clicked, opens a new message to the clicked-on address. If the field is representing a Web address, that field is assigned the sytle of "OpenURL", and a custom Edit snippet is called to render that field: displaying an Open button that, when clicked, opens the specified URL.

Enter the following URL in a browser:

http://localhost:6394/wa/r/demo/CardStep3

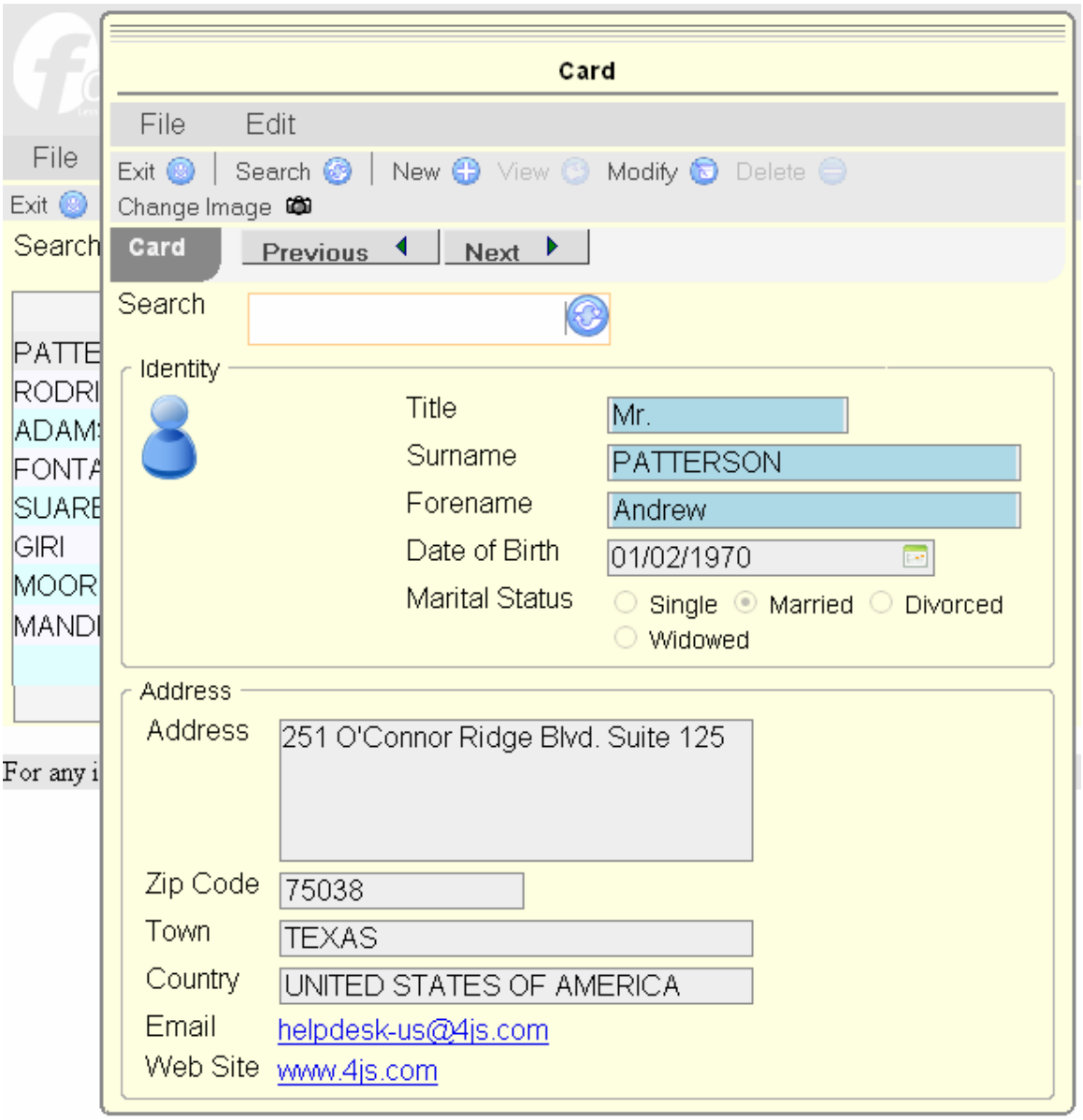


Figure 3-1: Contact displayed after referencing the customized FormField and Edit snippets in the application configuration file.
Fields such as 'Email' and 'Web Site', while still being Edit fields, are rendered differently based upon their assigned styles.

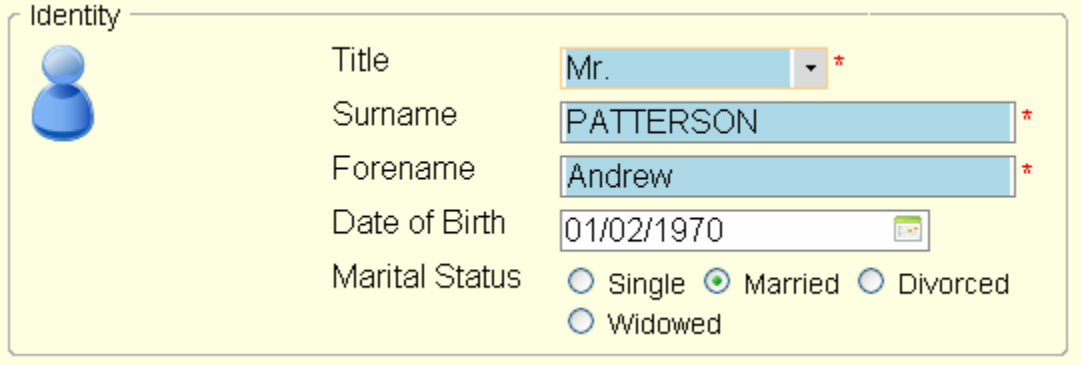


Figure 3-2: In input, required fields are marked with an asterisk

Things to observe:

- The Email field displays as a link that, when clicked, opens a new mail message using the default mail program.
- The Web Site field displays as a link also, when clicked, the specified Web Site displays.

Application Configuration File: CardStep3.xcf

Examine the application configuration file for this step of the tutorial.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.x
sd">
 <EXECUTION>
  <PATH>$(res.path.demo.app)/card/src</PATH>
  <MODULE>card.42r</MODULE>
 </EXECUTION>
 <OUTPUT>
  <MAP Id="DUA_AJAX">
   <THEME>
    <TEMPLATE
Id="_default">$(res.path.demo.app)/card/tpl/set1/main.xhtml</TEMPLATE>
    <SNIPPET Id="Image"
Style="Picture">$(res.path.demo.app)/card/tpl/set1/Image.xhtml</SNIPPET
>
    <SNIPPET
Id="FormField">$(res.path.demo.app)/card/tpl/set1/FormField.xhtml</SNIP
PET>
    <SNIPPET Id="Edit"
Style="Email">$(res.path.demo.app)/card/tpl/set1/Edit_Email.xhtml</SNIP
PET>
    <SNIPPET Id="Edit"
Style="OpenURL">$(res.path.demo.app)/card/tpl/set1/Edit_OpenURL.xhtml</
SNIPPET>
   </THEME>
```

```
  </MAP>
 </OUTPUT>
</APPLICATION>
```

Notes:

- Three snippets have been added for the DUA_AJAX Output Map for this application.
- A new SNIPPET element has been added for all FormField widgets.
- A new SNIPPET element has been added, specifying the custom snippet file to use for Edit fields associated with the style "Email".
- A new SNIPPET element has been added, specifying the custom snippet file to use for Edit fields associated with the style "OpenURL".

Modifications to the template and snippet files

Changes to the FormField snippet provide the code that places an asterisk next to required fields, when you are in input mode:

```
<span style="color:red" gwc:condition="item/isModifiable
&amp;&amp; item/isRequired &amp;&amp; item/hidden!=1"
>*</span>
```

isRequired notation is used to determine whether the required attribute is set to true, and if so an asterisk is displayed.

For the two custom Edit snippets, open the relevant files and observe the following:

For the Edit snippet for the style Email, the following has been added:

```
<a gwc:attributes="href 'mailto:'+value"
gwc:content="'mail'" />
```

For the Edit snippet for the style OpenURL, the following has been added:

```
<input type="button" value="Open" gwc:attributes="onclick
'OpenURL(\''+value+'\')'"/>
```

Step 4: Customize Snippet to use JavaScript to render the widget

For the next part of this tutorial, a custom Table snippet uses the gwc:marks instruction to make a call to a JavaScript that renders the table based on the value selected in the Page size list box.

Enter the following URL in a browser:

http://localhost:6394/wa/r/demo/CardStep4

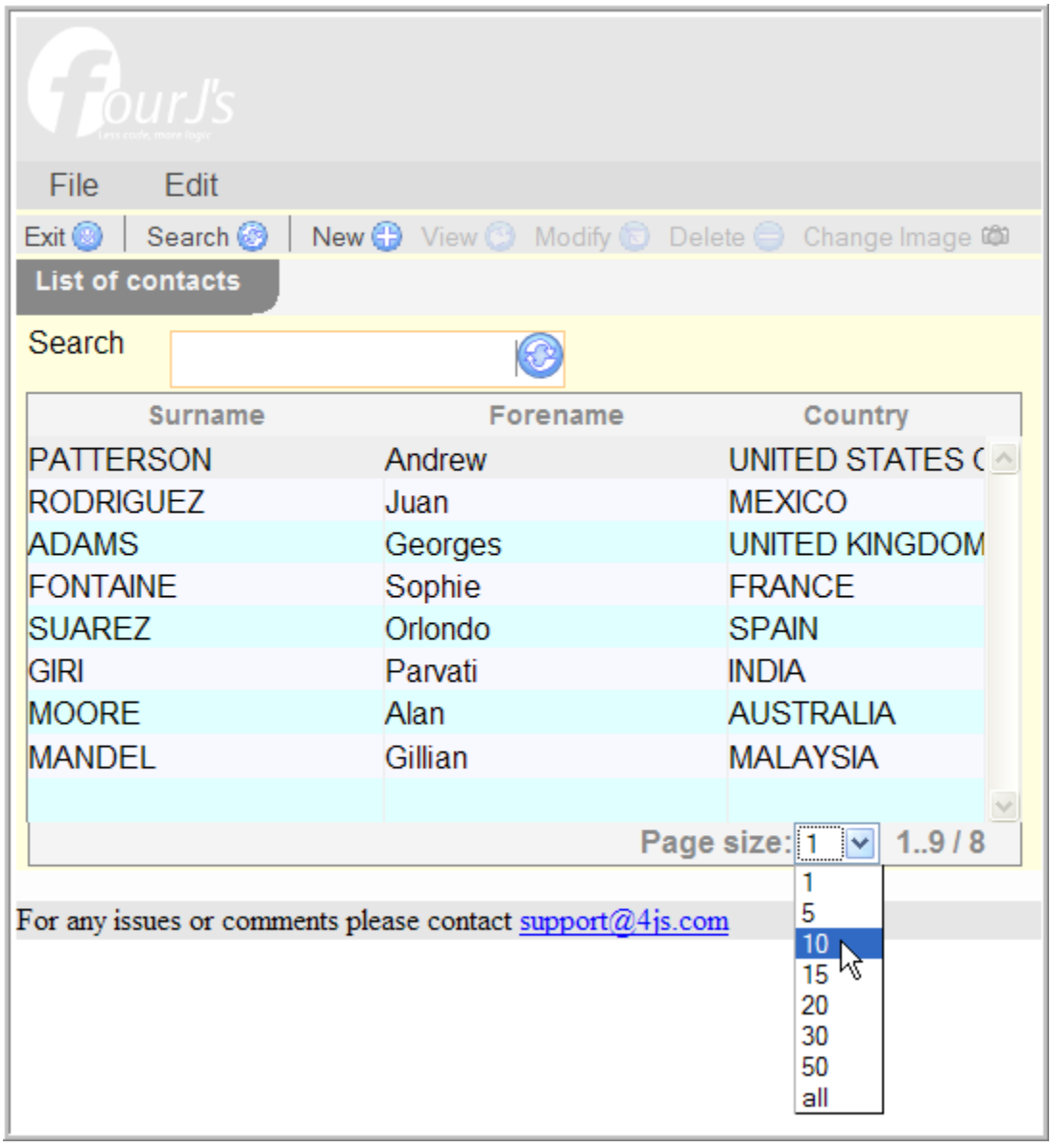


Figure 4-1: Contact List with customized Table widget

Things to observe:

- The table widget displays with a list box labeled "Page size". Selecting different values from this list box changes the number of rows displayed for the table.

Application Configuration File: CardStep4.xcf

Examine the application configuration file for this step in the tutorial. Note that a new entry has been added, specifying a custom snippet file for all Table widgets.

```
<?xml version="1.0"?>
<APPLICATION Parent="defaultgwc"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.x
sd">
 <EXECUTION>
  <PATH>$(res.path.demo.app)/card/src</PATH>
  <MODULE>card.42r</MODULE>
 </EXECUTION>
 <OUTPUT>
  <MAP Id="DUA_AJAX">
   <THEME>
    <TEMPLATE
Id="_default">$(res.path.demo.app)/card/tpl/set1/main.xhtml</TEMPLATE>
    <SNIPPET Id="Image"
Style="Picture">$(res.path.demo.app)/card/tpl/set1/ImageStep4.xhtml</SN
IPPET>
    <SNIPPET
Id="FormField">$(res.path.demo.app)/card/tpl/set1/FormField.xhtml</SNIP
PET>
    <SNIPPET Id="Edit"
Style="Email">$(res.path.demo.app)/card/tpl/set1/Edit_Email.xhtml</SNIP
PET>
    <SNIPPET Id="Edit"
Style="OpenURL">$(res.path.demo.app)/card/tpl/set1/Edit_OpenURL.xhtml</
SNIPPET>
    <SNIPPET
Id="Table">$(res.path.demo.app)/card/tpl/set1/Table.xhtml</SNIPPET>
   </THEME>
  </MAP>
 </OUTPUT>
</APPLICATION>
```

Modifications to the template and snippet files

By this point, you should easily identify that a custom snippet for the Table widget has been created and specified for this application.

Adding a select to change the number of lines displayed in a table

- In Table.xhtml, add a select item to the footer

```
<select gwc:marks="changePageSize [CID]" gwc:attributes="id
'pgsz_'+id">
  <option gwc:attributes="value makePageSizeIDID(id ,
1)">1</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
5)">5</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
10)">10</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
15)">15</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
20)">20</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
30)">30</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
50)">50</option>
  <option gwc:attributes="value makePageSizeIDID(id ,
size)">all</option>
</select>
```

makePageSizeIDID is the template function that will send the action to change the pageSize of the table
changePageSize is the new function we add to the component Table

- In card.js, handle the table pageSize resizing when the select item changes its value:

```
gwc.componentSet.Table.changePageSize = function( polarity, eid,
cid, component, data ) {
  // set an event on the object to submit the action when the
value is changed
  gwc.tk.SetEventEx(polarity, document.getElementById(eid),
'change', gwc.componentSet.Table.sentPageSize);
}
```

changePageSize is the function we declared in the Table.xhtml snippet using the **gwc:marks** instruction.
gwc.tk.SetEventEx is a function of the toolkit to handle (add/remove) events it take 4 arguments.
polarity: boolean, to add or remove the handler
elt: item on with the handler is set/removed
ev: event name
handler: function attached to the event

Warning! As with all functions in the CSF toolkit, this function is subject to changes.

```
gwc.componentSet.Table.sentPageSize = function(ev,elt){
  gwc.capi.Action(elt.value);
}
```

gwc.componentSet.Table.sentPageSize is the handler associate to the 'onchange' event of the select item.
gwc.api.Action sent an action to the engine.

- Add your custom snippet in the application configuration file

```
<SNIPPET
Id="Table">$(res.path.demo.app)/card/tpl/set1/Table.xhtml</
SNIPPET>
```

- Add your custom js to main.xhtml

```
<script type="text/javascript" gwc:attributes="src
application/connectorURI+'/demo/card.js'" defer="defer">
</script>
```

Send an action to the engine (see the custom Image.xhtml snippet)

Clicking on the picture of the contact will send the "changeimg" action to the engine and open a window to upload a new image for the contact.
Note, accepting the dialog without changing the image will set the image to blank.

- use the api **gwc.capi.Action** defined in $FGLASDIR/web/gwccore.js

  (**Warning!** These apis are subject to changes)

```
<a gwc:attributes="href
'javascript:gwc.capi.Action(gwc.core.state.ActionDidByName(
\'changeimg\') )'">[...]</a>
```

Communicate with external web sites like googlemap, geonames

To use googlemap you need to to sign up for a key at http://code.google.com/apis/maps.
You can use the current key if you are testing with localhost server.

- populate a 4GL field with javascript

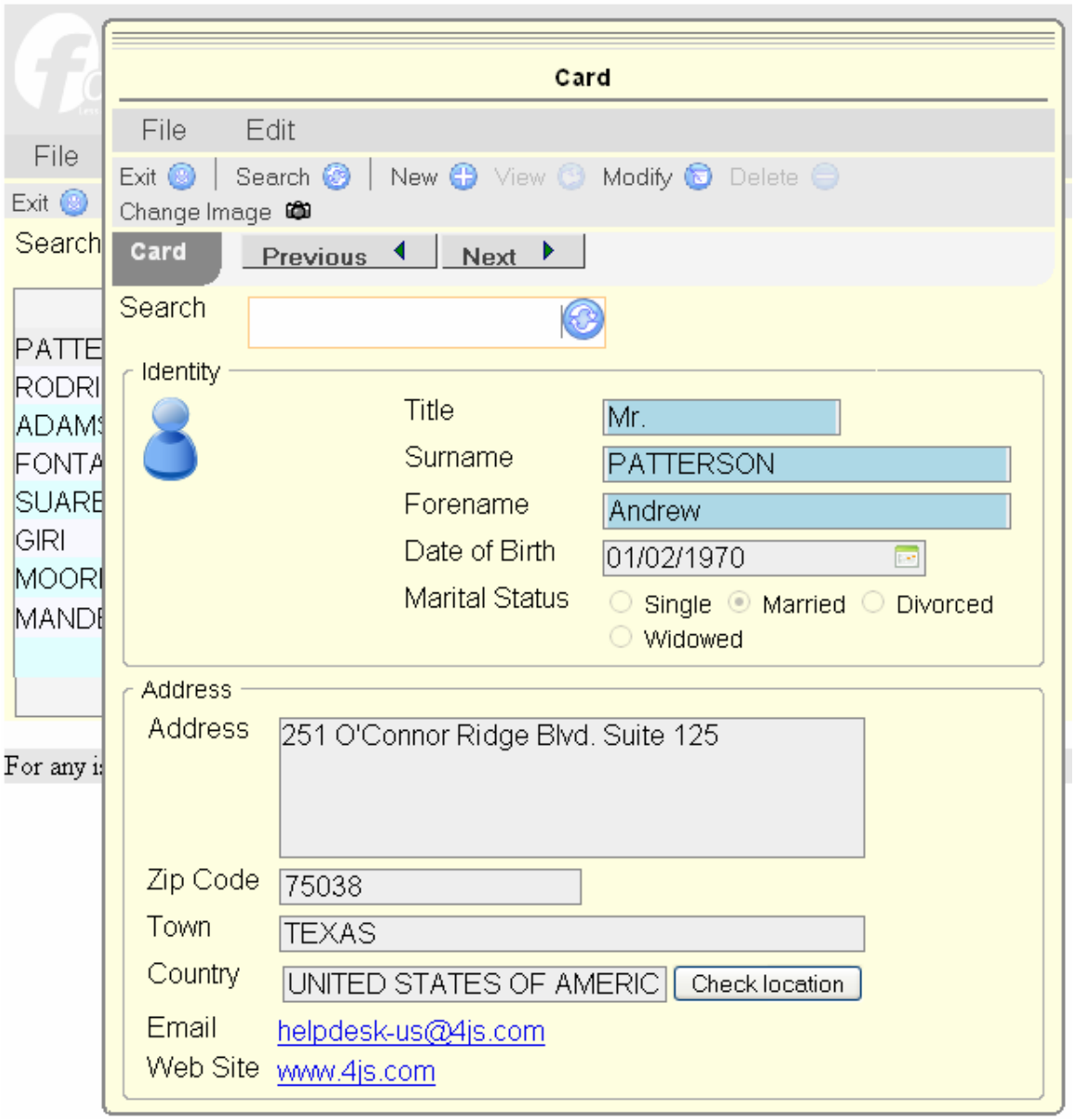


Figure 4-2: The Check Location button is added to the form.

Google Map and geonames are used to fill the country name. Clicking on the button gives the focus to the country field and opens the Google map. Once the map opens, double click on a country name to populate the country field.

Excerpt from Edit_GoogleMap.xhtml snippet:

```
<input type="button" value="Choose"
  gwc:condition="isModifiable"
  gwc:attributes="onclick 'gwc.capi.Focus(\'' + cid +
'\');' +
```

```
  'OpenMap(\''+ application/connectorURI + '/fjs/demo/card-
map.html\',\'wd_country\',\'' + value + '\')'"
/>
```

The field should have the focus, if you want the value to be taken in account by the 4GL application. To set the focus on a field, use the JavaScript api function **gwc.capi.Focus**.

Excerpt from card.js

```
function SetCountry(id,value) {
  var elt = gwc.tk.IdToElement(id);
  elt.value = value;
}
```

gwc.tk.IdToElement is a function of the toolkit that get the html object given the id of this element. The field is then filled with the value returned by geonames.

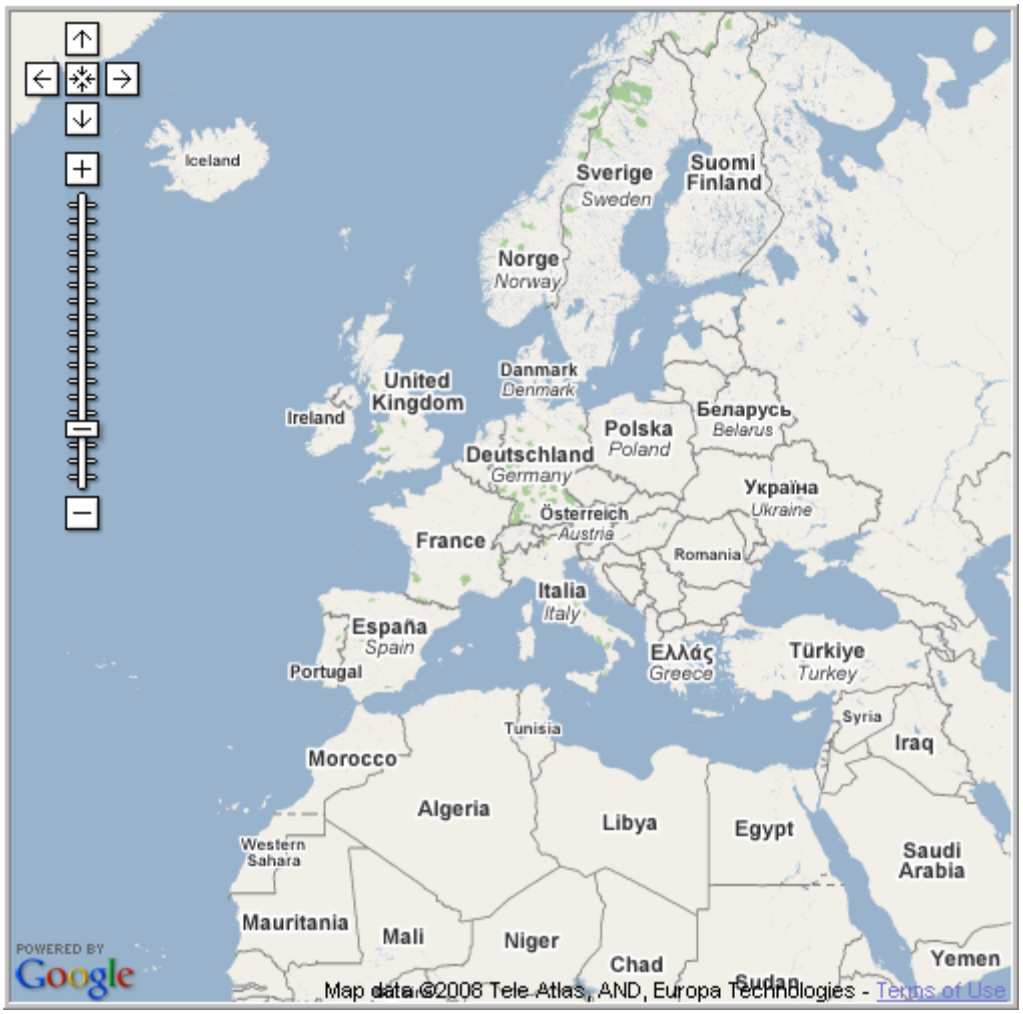


Figure 4-3: GoogleMap

- send information to other web sites

GoogleMap is used to locate and check the address. In the Edit_GoogleMap.xhtml snippet, get the value of the fields address, town and country.

```
<input type="button" value="Check location"
  gwc:condition="!isModifiable"
  gwc:define="address FormField['formonly.address']/item/value;
    town FormField['formonly.town']/item/value;
    country FormField['formonly.country']/item/value;
  "
  gwc:attributes="onclick 'ShowLocation(\''+
application/connectorURI + '/fjs/demo/card-map.html\','
    + '\'' + escapeJS(translate(address, ['\'',','] ,' ')) + '\','
    + '\'' + escapeJS(translate(town, ['\'',','] ,' ')) + '\','
    + '\'' + escapeJS(translate(country, ['\'',','] ,' ')) + '\''
    + ')'"
/>
```

In card.js, the JavaScript function ShowLocation submits the data to the map page. GoogleMap locates and displays the address on the map.

```
function ShowLocation(url,address,town,country) {
  var args = "id=&value=&address=" + address + "&town=" + town +
"&country=" + country;
  var win = window.open(url + "?" + args
,"CardMap","toolbar=no,menubar=no,location=no,height=500,width=500
");
}
```

Figure 4-4: GoogleMap locates and displays the address on a map.

Tutorial Summary

Throughout this tutorial, you have customized your application's User Interface using CSS, snippets, template files, and JavaScript.

You should have noted that at no time did you have to change the application source code.

How to Create a Breadcrumb Trail

The following code displays the list of windows, otherwise known as a "breadcrumb trail".

```
<div gwc:condition="application/ui">
  <span gwc:repeat="w application/ui/windows">
  <span gwc:condition="w/form" gwc:content="' :: ' + ( w/form/text ||
w/text || w/form/name )"/>
  <span gwc:condition="!w/form" gwc:content="' :: ' + w/name"/> </span>
</div>
```

You add this code to your template file (the main.xhtml file within the snippet set). Where you place this code in your file determines where the breadcrumb trail appears within the application interface.

How to Vary the Widget Display based on a Field Attribute

You have a form that has several fields that share the same widget type -- for example, a group of text edit fields -- however you want the rendering of the individual fields to be different based on some attribute of the field.

In this example, we provide you with an example of how you would have a snippet render the field differently based on a field attribute. A TAG attribute is added to a defined EDIT field to identify that field as one that contains a location. The value of this TAG attribute is then evaluated by a condition instruction within the snippet file and generates the appropriate code accordingly.

First, we modify the form specification file to include a TAG attribute with a value that specifies it as a location (e.g., sayhello).

```
01 EDIT edt10=formonly.edt10, SCROLL, TAG="sayhello";
```

You now have a value that you can use within the Edit.xhtml snippet file to react to the tag based on a gwc:condition instruction.

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02 <html xmlns:gwc="http://www.4js.com/GWC">
03
04 <!-- the head element is ignored -->
05 <head>
06  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
07  <title>Edit snippet</title>
08 </head>
09 <body>
10  <!--
11  (fontPitch ? ' gFontPitch_'+fontPitch : ' gFontPitch_variable') +
12  -->
13  <gwc:snippet-root>
14
15   <input type="text" readonly="readonly" class="gField gInherit"
16    gwc:marks="
17    field [CID];
18    currentFieldStyle [CID,'gCurrentField'];
19    modifiable isModifiable?[CID]:null;
20    "
21    gwc:attributes="
22      type isPassword ? 'password' : 'text';
23      class _tpl_+(isModifiable ? '':' gDisabled') +
24      ' g'+type+' '+prefix('gc',class)+'
'+prefix('g'+type+'_gc',class) +
25      (hidden!=2?'':' gHidden') +
26      (justify ? ' gJustify_'+justify : (isNumeric ? '
gJustify_right' : '')) +
27      (shift ? ' gShift_'+shift : '') +
```

```
28      ' gFontFamily_'+style['fontFamily'];
29      style (style['textColor']?' color:'+style['textColor']+';':'')
+
30      (style['backgroundColor']?' background-
color:'+style['backgroundColor']+';':'');
31      value value;
32      size width;
33      maxLength maxLength || null;
34      title comment;
35    "
36   />
37   <image gwc:condition="tag=='sayhello'" onclick="alert('hello')"
      gwc:attributes="src application/connectorURI + '/pic/accept.png'"
/>
38  </gwc:snippet-root>
39 </body>
40 </html>
```

Notes

1. In Line 37, a gwc:condition instruction specifies that if the value of the tag is "sayhello" then include the image tag specified.

How to Relate Styles, Classes, and Selectors

When defining forms, you can use the STYLE attribute to assign a style to particular form objects. This help topic examines how the STYLE attribute can ultimately result in a selector that can then be referenced by CSS.

Form Objects and Styles

When defining a form element, you can use the style attribute to specify a presentation style for the form element. For example, the following code sample from a .PER file assigns a style (named "mystyle") to the style attribute for an Edit field:

```
EDIT f01 = FORMONLY.f01, STYLE="mystyle"
```

Now that the style is set for the form element, when displaying the form within the GWC, you want to have your CSS dictate how that style displays. In other words, you want the CSS to recognize the "mystyle" style attribute and format the display accordingly.

Accessing a Style using the Class template path

When an object has the STYLE attribute defined, the result is that the "class" attribute of the GWC object is set using the value assigned to the STYLE attribute.
As a result, the style can be accessed using the template path class.

Continuing with the example started above, the snippet Edit.xhtml uses this "class" attribute

```
<input type="text"
  [...]
  gwc:attributes="
  [...]
  class [...]
  ' g'+type+' '+prefix('gc',class)+' '+prefix('g'+type+'_gc',class) +
  [...]
/>
```

This code within the snippet file generates an attribute class="... gEdit_gcmystyle gEdit gcmystyle ..." or (roughly):

```
<input type="text" class="gEdit_gcmystyle gEdit gcmystyle" />
```

Using Class values as Selectors

With the class generated by the snippet file, the class vaues `gEdit_gcmystyle`, `gcmystyle`, or `gEdit.gcmystyle` can be used as selectors.

Where to use Class values as Selectors

Depending on the HTML structure in the snippet, you can set the GWC class attribute elsewhere, so you can have other selectors.

Example

```
<div class="gEdit">
  <input type="text" class="gEdit_gcmystyle gcmystyle"/>
</div>
```

This example would provide the selector gEdit for the container of the Edit (div), and gEdit_gcmystyle or gcmystyle or gEdit gwcmystyle for the Edit itself (input).

How to Display a Label as a Hyperlink

The following example shows how to display a label on a form as an active hyperlink. While this example displays a label as a hyperlink, you can use this example as a template for making other types of widgets widget appear as hyperlinks.

Step 1: Create a snippet file that displays a label as a hyperlink (for example, LabelURL.xhtml).

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns:gwc="http://www.4js.com/GWC">

<!-- the head element is ignored -->
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 <title>Label snippet</title>
</head>
<body>
<!-- the template snippet is the content of the gwc:snippet-root
element -->
 <gwc:snippet-root>
  <a gwc:content="value" gwc:attributes="href value"/>
 </gwc:snippet-root>
</body>
</html>
```

Step 2: In the application configuration file (.xcf file), add a SNIPPET element that specifies to use the newly-created snippet file ("LabelURL.xhtml") when the form displays a label whose style property is "LabelUrl".

```
<APPLICATION Parent="defaultgwc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.10/cfextwa.x
sd">
 <EXECUTION>
  <PATH>$(res.qa.path)\howtos\LabelURL\src\4gl</PATH>
 </EXECUTION>
 <OUTPUT>
  <MAP Id="DUA_AJAX">
   <THEME>
    <SNIPPET Id="Label"
Style="LabelUrl">$(res.qa.path)\mypath\src\web\LabelURL.xhtml</SNIPPET>
   </THEME>
  </MAP>
 </OUTPUT>
</APPLICATION>
```

Step 3: For each label you want to appear as a URL, set the style property to "LabelUrl" (as shown in the .PER file example below):

```
LAYOUT (STYLE="MyStyle")
GRID g1
{
Enter a URL in the edit and click on change action
to make the URL appear as a link
[edt10 ]
[lbl01 ]
}
END
END

ATTRIBUTES
EDIT edt10=formonly.edt10, SCROLL;
LABEL lbl01=formonly.lbl01, STYLE="LabelUrl";
END
```

To run and test this example, you can use the sample .4GL code provided below.

```
MAIN

DEFINE edt10 STRING
LET edt10 = "http://www.google.com"
CLOSE WINDOW SCREEN
OPEN WINDOW w WITH FORM "sample"

INPUT BY NAME edt10 WITHOUT DEFAULTS ATTRIBUTES(UNBUFFERED)
  ON ACTION change
    DISPLAY edt10 TO lbl01
END INPUT

CLOSE WINDOW w
END MAIN
```

GAS Configuration File Overview

The Genero Application Server is configured through a configuration file. The default configuration file is the **as.xcf** file, located in the **$FGLASDIR/etc** directory. You can create a separate Genero Application Server configuration file by creating a copy of the as.xcf file and making your modifications to the copy. You can specify which configuration file to use when starting the Genero Application Server by using the "-f" option. For more information about starting the Genero Application Server, refer to the *Genero Application Server Overview* section in this manual.

The configuration file is written using XML. The document consists of elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attributes. Each attribute specification has a name and a value.

The top-most element in the Application Server configuration file is the `CONFIGURATION` element. The `CONFIGURATION` element is the root element for all Genero configuration files. 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.

Syntax:

```
<CONFIGURATION>
  <APPLICATION_SERVER>
    <RESOURCE_LIST>
      ...
    </RESOURCE_LIST>
    <COMPONENT_LIST>
      ...
    </COMPONENT_LIST>
    <INTERFACE_TO_CONNECTOR>
      ...
    </INTERFACE_TO_CONNECTOR>
    <INTERFACE_TO_DVM>
      ...
    </INTERFACE_TO_DVM>
    <LOG>
      ...
    </LOG>
    <FILE_TRANSFER>
      ...
    </FILE_TRANSFER>
    <AUTHENTICATION>
      ...
    </AUTHENTICATION>
    <APPLICATION_LIST>
      ...
```

```
    </APPLICATION_LIST>
    <SERVICE_LIST>
      ...
    </SERVICE_LIST>
  </APPLICATION_SERVER>
</CONFIGURATION>
```

Notes:

The APPLICATION_SERVER element contains the following child elements:

1. One `RESOURCE_LIST` element, containing a list of resources.
2. One `COMPONENT_LIST` element, containing a list of components.
3. One `INTERFACE_TO_CONNECTOR` element, specifying the interface between the Genero Application Server (GAS) and the GAS Connector. The connector is either the CGI connector (fglccgi), the ISAPI filter (fglcisapi), or the user agent through direct connection.
4. One `INTERFACE_TO_DVM` element, specifying the interface to the Dynamic Virtual Machine.
5. Zero or more `LOG` elements, specifying the type of information that is logged and where it is logged to.
6. Zero or more `FILE_TRANSFER` elements, specifying the directory where files are stored while being transferred between the front-end machine and the DVM.
7. Zero or one `AUTHENTICATION` element, specifying authentication parameters.
8. Zero or one `APPLICATION_LIST` element, containing a list of applications.
9. Zero or one `SERVICE_LIST` element, containing a list of Web Services.

Configuration File Element Listing

The elements contained within the Genero Application Server configuration file are listed below. The elements are described in detail within other sections of this manual; click on an element name to be taken to the page discussing that element (when viewing this manual online).

- APPLICATION_SERVER
  - RESOURCE_LIST
    - PLATFORM_INDEPENDENT
      - RESOURCE
    - WNT
      - RESOURCE
    - UNX
      - RESOURCE
  - COMPONENT_LIST
    - WEB_APPLICATION_EXECUTION_COMPONENT
      - ENVIRONMENT_VARIABLE
      - PATH
      - DVM
      - MODULE

        - AUTHENTICATION
        - PARAMETERS
    - SERVICE_APPLICATION_EXECUTION_COMPONENT
        - ENVIRONMENT_VARIABLE
        - PATH
        - DVM
        - MODULE
        - AUTHENTICATION
        - PARAMETERS
        - POOL
            - START
            - MIN_AVAILABLE
            - MAX_AVAILABLE
    - WEB_APPLICATION_TIMEOUT_COMPONENT
        - USER_AGENT
        - REQUEST_RESULT
        - DVM_AVAILABLE
    - SERVICE_APPLICATION_TIMEOUT_COMPONENT
        - DVM_AVAILABLE
        - DVM_FREE
        - REQUEST_QUEUE
        - REQUEST_RESULT
    - WEB_APPLICATION_PICTURE_COMPONENT
        - PATH
    - WEB_APPLICATION_RENDERING_COMPONENT
        - OUTPUT_DRIVER
    - WEB_APPLICATION_THEME_COMPONENT
        - TEMPLATE
- INTERFACE_TO_CONNECTOR
    - TCP_BASE_PORT
    - TCP_PORT_OFFSET
    - LIMIT_REQUEST_SIZE
    - DOCUMENT_ROOT
    - TEMPORARY_DIRECTORY
    - ERROR_DOCUMENT
    - ALIAS
- INTERFACE_TO_DVM
    - ADDRESS
    - TCP_BASE_PORT
    - TCP_PORT_RANGE
    - EXCLUDED_PORT
- LOG
    - OUTPUT
    - FORMAT
    - CATEGORIES_FILTER
    - RAW_DATA
- FILE_TRANSFER
    - TIMEOUT
- AUTHENTICATION
    - REALM
    - SERVICE_NAME

        - KEYTAB
    - APPLICATION_LIST
        - GROUP
        - APPLICATION
            - RESOURCE
            - EXECUTION
                - ENVIRONMENT_VARIABLE
                - PATH
                - DVM
                - MODULE
                - AUTHENTICATION
                - PARAMETERS
            - TIMEOUT
                - USER_AGENT
                - REQUEST_RESULT
                - DVM_AVAILABLE
            - PICTURE
                - PATH
            - OUTPUT
                - MAP
                    - RENDERING
                        - OUTPUT_DRIVER
                    - THEME
                        - TEMPLATE
    - SERVICE_LIST
        - GROUP
        - APPLICATION
            - RESOURCE
            - EXECUTION
                - ENVIRONMENT_VARIABLE
                - PATH
                - DVM
                - MODULE
                - AUTHENTICATION
                - PARAMETERS
                - POOL
                    - START
                    - MIN_AVAILABLE
                    - MAX_AVAILABLE
            - TIMEOUT
                - DVM_AVAILABLE
                - DVM_FREE
                - REQUEST_QUEUE
                - REQUEST_RESULT

Resource List - Configuration Reference

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.

RESOURCE_LIST

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 kind of variable that can be used in the configuration files and in template files. By defining and using resources, when the value of the resource needs updating, it becomes possible to modify the resource in one location - within the `RESOURCE_LIST` - and the new value is carried through the various components that reference the resource.

A resource is defined as platform-independent or platform-dependent, based on the section in which the resource is defined.

Syntax:

```
<RESOURCE_LIST>
  <PLATFORM_INDEPENDENT> [ resource] [...] </PLATFORM_INDEPENDENT>
  [ <WNT> resource [...] </WNT> ]
  [ <UNX> resource [...] </UNX> ]
</RESOURCE_LIST>
```

Notes:

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

1. One `PLATFORM_INDEPENDENT` element (required).
2. One `WNT` element (required).
3. One `UNX` element (required).

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, refer to the RESOURCE section below.

PLATFORM_INDEPENDENT

The `PLATFORM_INDEPENDENT` element containing a list of platform-independent resources, available on both Unix and Windows platforms.

Notes:

The `PLATFORM_INDEPENDENT` element may contain the following child element (described below):

1. Zero or more `RESOURCE` elements (optional).

WNT

The `WNT` element containing a list of Windows NT resources: those resources are only available on the Windows NT/2000/XP operating systems.

Notes:

The `WNT` element may contain the following child element (described below):

1. Zero or more `RESOURCE` elements (optional).

UNX

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.

Notes:

The `UNX` element may contain the following child element (described below):

1. Zero or more `RESOURCE` elements (optional).

RESOURCE

A `RESOURCE` element defines a resource, or variable, that can be used in the configuration files and in template files. It takes two attributes, an **Id** attribute and a **Source** attribute.

- The **Id** attribute provides a unique identifier for the resource. It is this unique identifier that is used elsewhere in the various configuration files to reference this resource, and therefore must be unique.
- The **Source** attribute specifies where the resource will find its value:
  - **INTERNAL** indicates that the resource value is provided inline as the content of the element.
  - **FILE** indicates that the resource value is stored in a file, and the content of the element is the path and file name of the file.
  - **ENVIRON** indicates that the resource value is the value of the environment variable specified as the content of the element.

Syntax:

```
<RESOURCE Id="resId" Source="( FILE | INTERNAL | ENVIRON ) " > resData
</RESOURCE>
```

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 FILE, *resData* is the path to this file.
   - 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:

The following example illustrates a resource defined inline.

```
<RESOURCE Id="res.dvm.wa"
Source="INTERNAL">$(res.fgldir)/bin/fglrun.exe</RESOURCE>
```

The following example illustrates a resource defined as the contents of a file. In this example, the resource is a file named generodefault.html, stored in the template directory specified by the resource res.path.tpl:

```
<RESOURCE Id="res.theme.default.gwc.template"
Source="FILE">$(res.path.tpl)/generodefault.html</RESOURCE>
```

The following example illustrates 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>
```

Component List - Configuration Reference

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.

Components are grouped by type. Each component type is discussed in its own section of this manual.

Types of components:

The COMPONENT_LIST element may contain the following child elements, or component types:

- Zero or more WEB_APPLICATION_EXECUTION_COMPONENT elements (optional).
- Zero or more SERVICE_APPLICATION_EXECUTION_COMPONENT elements (optional).
- Zero or more WEB_APPLICATION_TIMEOUT_COMPONENT elements (optional).
- Zero or more SERVICE_APPLICATION_TIMEOUT_COMPONENT elements (optional).
- Zero or more WEB_APPLICATION_PICTURE_COMPONENT elements (optional).
- Zero or more WEB_APPLICATION_RENDERING_COMPONENT elements (optional).
- Zero or more WEB_APPLICATION_THEME_COMPONENT elements (optional).

Syntax:

```
<CONFIGURATION>
  <APPLICATION_SERVER>
    ...
    <COMPONENT_LIST>
      executionComponent [...]
      applicationTimeoutComponent [...]
      serviceTimeoutComponent [...]
      pictureComponent [...]
      renderingComponent [...]
      themeComponent [...]
  </COMPONENT_LIST>
    ...
  </APPLICATION_SERVER>
</CONFIGURATION>
```

Notes:

1. The COMPONENT_LIST element does not support any attributes.

Application Execution Component - Configuration Reference

An Execution component prepares the runtime environment for an application or Web service, defining execution rules and setting the execution environment. An Execution component is referenced in an application by its unique identifier, set by the **Id** attribute.

- Web Application Execution Component
- Service Application Execution Component

WEB_APPLICATION_EXECUTION_COMPONENT

The WEB_APPLICATION_EXECUTION_COMPONENT creates a Web application execution component, which 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 a base set of execution parameters. The attribute **AllowUrlParameters** defines whether the parameters provided on the command line 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> ]
  [ <AUTHENTICATION> authtype </AUTHENTICATION> ]
  [ <PARAMETERS> parameterSettings </PARAMETERS> ]
</WEB_APPLICATION_EXECUTION_COMPONENT>
```

Notes:

The WEB_APPLICATION_EXECUTION_COMPONENT element may contain the following child elements (described below):

1. Zero or more ENVIRONMENT_VARIABLE elements (optional).
2. Zero or one PATH element (optional).
3. Zero or one DVM element (optional).
4. Zero or one MODULE element (optional).
5. Zero or one AUTHENTICATION element (optional).
6. Zero or one PARAMETERS element (optional).

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_APPLICATION_EXECUTION_COMPONENT>
```

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> ]
  [ <AUTHENTICATION> authtype </AUTHENTICATION> ]
  [ <PARAMETERS> parameterSettings </PARAMETERS> ]
  [ <POOL> poolSettings </POOL> ]
</SERVICE_APPLICATION_EXECUTION_COMPONENT>
```

Notes:

The SERVICE_APPLICATION_EXECUTION_COMPONENT element may contain the following child elements (described below):

1. Zero or more ENVIRONMENT_VARIABLE elements (optional).
2. Zero or one PATH element (optional).
3. Zero or one DVM element (optional).
4. Zero or one MODULE element (optional).
5. Zero or one AUTHENTICATION element (optional).
6. Zero or one PARAMETERS element (optional).
7. Zero or one POOL element (optional).

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>
```

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**.

The CGI transmits the environment variables it receives from the web server to the GAS. Each environment variable is prefixed by FGL_WEBSERVER_ for the DVM environment

---

PATH

The `PATH` element specifies the current working directory for the application module.

Usage Example:

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

DVM

The `DVM` element specifies the name of the Dynamic Virtual Machine you want to use to start and run the application. Typically this value is **fglrun** for Unix Systems (UNX) and **fglrun.exe** for Windows NT/2000/XP (WNT).

Usage Example:

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

MODULE

The `MODULE` element specifies the application module name (the name of the **.42r** module you want to run). If omitted, the Genero Application Server 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>
```

AUTHENTICATION

The `AUTHENTICATION` element specifies the type of authentication to be used for the application. At this time, only Kerberos is supported.

While this element can be specified as part of an execution component, it is typically defined at the application level.

Usage Example:

```
<AUTHENTICATION>KERBEROS</AUTHENTICATION>
```

PARAMETERS

The PARAMETERS element specifies the parameters to provide on the DVM command line. To enable **URL** parameters, in the EXECUTION tag, set the **AllowUrlParameters** attribute of the to TRUE.

Syntax:

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

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 configuration file.

POOL

The POOL element sets the limitations regarding the number of Virtual Machines that are attached to a Web Service. You specify three values within a POOL element: the number of Virtual Machines to start when the Genero Application Server starts, the minimum number of Virtual Machines to have alive while the Genero Application Server is running, and the maximum number of Virtual Machines to have alive while the Genero Application Server is running.

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> ]
</POOL>
```

Notes:

The POOL element may contain the following child elements (described below):

1. Zero or one START elements (optional).
2. Zero or one MIN_AVAILABLE element (optional).
3. Zero or one MAX_AVAILABLE element (optional).

Example:

```
<POOL>
  <START>5</START>
  <MIN_AVAILABLE>3</MIN_AVAILABLE>
  <MAX_AVAILABLE>10</MAX_AVAILABLE>
</POOL>
```

In this example, 5 Virtual Machines are started to service the Web service when the Genero Application Server starts; the number can fall as low as 3 Virtual Machines or raise as high as 10 Virtual Machines. For more information on setting service pool elements, refer to the Service Pool section of the Deployment Architecture for Web Services topic.

START

The START element specifies the number of Virtual Machines to start for this Web Service when the Genero Application Server starts.

Constraint:

START <= MAX_AVAILABLE

MIN_AVAILABLE

The MIN_AVAILABLE element specifies the minimum number of available Virtual Machine 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, based on the setting of DVM_FREE, the number of DVMs can decrease to reach MIN_AVAILABLE.

Constraint:

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

MAX_AVAILABLE

The `MAX_AVAILABLE` element specifies the maximum number of available Virtual Machines to be attached to a Web Service.

Constraints:

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

Application Timeout Component - Configuration Reference

Timeout components provide instruction on how the Genero Application Server (GAS) or Web service responds after a specific period of time has passed, providing mechanisms to react to time-based events (such as user inactivity or the time it takes to start a new application). Each timeout element specifies the number of seconds to wait prior to having the application server perform the task related to the timeout element. A timeout is fired when the specific condition is met.

Timeout elements can be defined for either:

- Web applications (timeouts relating to the Front End - GDC, GJC, or GWC)
- Web services

Within the `COMPONENT_LIST` element of the Genero Application Server configuration file, you can define timeout components (a `WEB_APPLICATION_TIMEOUT_COMPONENT` or `SERVICE_APPLICATION_TIMEOUT_COMPONENT`) that can later be referenced when configuring an application, in order to set the timeout values for the application. Individual timeout elements can then be overwritten within the application's configuration itself.

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 by an application, providing that application with a base 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> ]
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

Notes:

The `WEB_APPLICATION_TIMEOUT_COMPONENT` element may contain the following child elements (described below):

1. Zero or one USER_AGENT element (optional).
2. Zero or one REQUEST_RESULT element (optional).
3. Zero or one DVM_AVAILABLE element (optional).

Example:

```
<WEB_APPLICATION_TIMEOUT_COMPONENT Id="cpn.wa.timeout.set1">
  <USER_AGENT>300</USER_AGENT>
  <REQUEST_RESULT>240</REQUEST_RESULT>
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
</WEB_APPLICATION_TIMEOUT_COMPONENT>
```

In this example, the **Id** value - **cpn.wa.timeout.set1** - can be referenced when defining an application. When an application references a component by its **Id** value, it inherits the settings defined by that component.

USER_AGENT timeout (user inactivity)

The User Agent timeout is used to identify a period of user inactivity. The `USER_AGENT` element specifies the number of seconds to wait for a client request before assuming that the Front End client has died (based on user inactivity). After the specified period passes without the client emitting a further request, the Genero Application Server asks the Virtual Machine to properly close the application.

Under normal operation, the Front End client sends a GET request to the Genero Application Server immediately after receiving a response. Therefore, 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 DVM to shut down.

With the Genero Desktop Client Active X and Genero Java Client, the USER_AGENT timeout usually does not expire. When the user closes the application, the DVM handling that application is properly shut down.

With the Genero Web Client, however, the USER_AGENT timeout 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. When the user does not properly exit the application, however, the DVM remains alive and continues to consume a license 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. Because of the USER_AGENT timeout, however, the USER_AGENT timeouts and Genero Application Server unilaterally close the socket to the DVM, which causes the DVM to shut down and the license to be released.

Usage Example:

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

In this usage example, the User Agent timeout is set to 300 seconds.

REQUEST_RESULT timeout (transaction pending)

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 Virtual Machine 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 may take a little 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 240 seconds.

Usage Example:

```
<REQUEST_RESULT>240</REQUEST_RESULT>
```

In this usage example, the Request Result timeout is set to 240 seconds.

DVM_AVAILABLE timeout (DVM startup time)

The DVM Available timeout provides a delay for the DVM to start and be available. The DVM_AVAILABLE element specifies how long (in seconds) the Genero Application Server allows for the DVM to start.

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."

The DVM_AVAILABLE timeout is only applicable when you start an application or you launch sub process in interactive mode (IN FORM MODE). If you run the sub process in background (IN LINE MODE), the DVM_AVAILABLE timeout is not applicable.

Usage Example:

```
<DVM_AVAILABLE>10</DVM)AVAILABLE>
```

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

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 Genero Application Server 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>
  <DVM_FREE>dvmFreeTimeOut</DVM_FREE>
  <REQUEST_QUEUE>rqTimeOut</REQUEST_QUEUE>
  <REQUEST_RESULT>rrTimeOut</REQUEST_RESULT>
</SERVICE_APPLICATION_TIMEOUT_COMPONENT>
```

Notes:

The `SERVICE_APPLICATION_TIMEOUT_COMPONENT` element may contain the following child elements (described below):

1. Zero or one DVM_AVAILABLE element (optional).
2. Zero or one DVM_FREE element (optional).
3. Zero or one REQUEST_QUEUE element (optional).
4. Zero or one REQUEST_RESULT element (optional).

Example:

```
<SERVICE_APPLICATION_TIMEOUT_COMPONENT Id="cpn.ws.timeout.set1">
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
  <DVM_FREE>10</DVM_FREE>
  <REQUEST_QUEUE>10</REQUEST_QUEUE>
  <REQUEST_RESULT>240</REQUEST_RESULT>
</SERVICE_TIMEOUT_COMPONENT>
```

In this example, the **Id** value - **cpn.ws.timeout.set1** - can be referenced when defining an Web service. When a Web service references a component by its **Id** value, it inherits the settings defined by that component.

DVM_AVAILABLE timeout (DVM startup time)

The DVM Available timeout provides a delay for the DVM to start and be available. The `DVM_AVAILABLE` element specifies how long (in seconds) the Genero Application Server allows for the DVM to start.

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."

Usage Example:

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

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

DVM_FREE timeout

The DVM Free timeout is used to shut down a DVM that has no request to process. The DVM_FREE element specifies how long (in seconds) a DVM waits with no request to process before determining whether to shut down.

How the DVM reacts when it has no request to process is dependant on the settings for the Web services pool of DVMs. After the DVM_FREE timeout expires, the DVM is shut down if the number of running DVMs is greater than MIN_AVAILABLE. If a DVM has no request to process and the number of DVMs is greater than MAX_AVAILABLE, the DVM does not wait for the DVM_FREE timeout to expire but instead shuts down immediately.

Usage Example:

```
<DVM_FREE>10</DVM_FREE>
```

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

REQUEST_QUEUE timeout

The Request Queue timeout prevents a request from waiting indefinitely for a DVM to be made available. The REQUEST_QUEUE element specifies how long (in seconds) a request waits for a DVM to be available to handle the request. The DVM can either be an existing DVM or a new DVM started to handle the request. If a new DVM cannot be started and all other DVMs are busy processing other requests, the request will be rejected after the REQUEST_QUEUE timeout expires and an error message is logged: "REQUEST_QUEUE timeout expires". This prevents a request from being in the request queue indefinitely.

Note: The pool of requests and the pool of DVMs are independent. A request is processed as long as a DVM is available to process the request. If a request comes in and there are no available DVMs to process that request, the system attempts to start a new DVM. If a new DVM cannot be started and the DVM_AVAILABLE timeout expires, the request remains in the pool of requests until either a DVM becomes available to process the request or the REQUEST_QUEUE timeout expires, at which time the request is notified that it could not be processed.

Usage Example:

```
<REQUEST_QUEUE>10</REQUEST_QUEUE>
```

In this usage example, the Request Queue timeout is set to 10 seconds.

REQUEST_RESULT timeout

The Request Result timeout prevents a request from processing indefinitely. The `REQUEST_RESULT` element specifies the maximum amount of time (in seconds) for the DVM to process the request. If the time expires, GAS closes the connection and logs an error.

Usage Example:

```
<REQUEST_RESULT>240</REQUEST_RESULT>
```

In this usage example, the Request Result timeout is set to 240 seconds.

Web Application Picture Component - Configuration Reference

The Picture component defines how images are served. A Picture component is referenced in an application by its unique identifier, set by the **Id** attribute.

WEB_APPLICATION_PICTURE_COMPONENT

The `WEB_APPLICATION_PICTURE_COMPONENT` element specifies the directory from which images are served. It takes an attribute Id, which specifies the unique identifier for this Picture component. It is this unique identifier that is referenced by an application, providing that application with the location of its image directory.

You need to specify only the alias to the picture directory. The mapping between the alias and the physical directory where the images are stored is defined in the `INTERFACE_TO_CONNECTOR` element. For more information on setting up aliases, refer to the *Aliases* section of this manual.

Syntax:

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="resID">
 <PATH>$(connector.uri)aliasPath</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Notes:

1. *resID* is the unique identifier for this picture component definition.
2. $(connector.uri) is the resource for the Web server directory.
   If you are using a direct connection, the resource **$(connector.uri)** is empty.
   If you connect through an Apache web server, **$(connector.uri)** is replaced by **/cgi-bin/fglccgi/**, assuming your URL is **http://*WebServer*/cgi-bin/fglccgi/wa/r/*AppID***.
   If **$(connector.uri)** is not specified in the picture path, the web server is searched for the images.
3. *aliasPath* is an alias path, mapping to the physical directory that stores the image files. You define the alias in the `INTERFACE_TO_CONNECTOR` element.

Example:

```
<WEB_APPLICATION_PICTURE_COMPONENT Id="cpn.picture">
  <PATH>$(connector.uri)/fjs/pics</PATH>
</WEB_APPLICATION_PICTURE_COMPONENT>
```

Note: The Front End clients use **$(pictures.uri)** in their templates to access the pictures. This corresponds to the picture component path: **$(connector.uri)/fjs/pics**.

The path to the pictures must specify an alias. The alias would be defined in the INTERFACE_TO_CONNECTOR element:

```
<INTERFACE_TO_CONNECTOR>
  <TCP_BASE_PORT>6300</TCP_BASE_PORT>
  <TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
  <DOCUMENT_ROOT>$(res.path.docroot)</DOCUMENT_ROOT>
  <ALIAS Id="/fjs/pics">$(res.path.pics)</ALIAS>
</INTERFACE_TO_CONNECTOR>
```

Tip: When creating your html pages, use the absolute alias path to html objects like images or JavaScript files; for example, use **/fjs/pic/accept.png** rather than **../pic/accept.png**. This saves time when moving from a development environment (direct connection to the GAS) to a production environment (connection through a web server).

PATH

The PATH element specifies the URL for the directory where the images reside. This URL typically consists of the Web server directory resource combined with an alias for the image directory.

Refer to the discussion under WEB_APPLICATION_PICTURE_COMPONENT for more information on defining the PATH element.

Application List Reference (Defining Applications)

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. This section of the documentation outlines the rules for writing the XML that defines an application; for information about the general process of defining applications and groups, refer to the Adding Applications for *Genero Web Client* or *Genero Desktop Client* section of this manual.

For information about configuring for Web applications, refer to the *Defining Web Services* section of this manual.

APPLICATION_LIST

The `APPLICATION_LIST` element provides a list of groups and Web applications (for those Web applications defined within the Genero Application Server configuration file). It takes an attribute **MaxLicenseConsumption**, which takes an integer specifying the maximum number of licenses that can be consumed.

Syntax:

```
<APPLICATION_LIST MaxLicenseConsumption=numlicenses />
  [ <GROUP element.> ] [...]
  [ <APPLICATION ....> ] [...]
  ...
</APPLICATION_LIST
```

Notes:

The `APPLICATION_LIST` element may contain the following child elements (described below):

1. Zero or more `GROUP` elements (optional).
2. Zero or more `APPLICATION` elements (optional).

Example:

```
<APPLICATION LIST MaxLicenseConsumption=10 />
  <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>
```

GROUP

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>
```

Notes:

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>
```

APPLICATION

An APPLICATION element defines an application. It takes up to four attributes:

- **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.
- **MaxLicenseConsumption** - A non-negative Integer specifying the maximum license consumption.
- **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.

When you define an application, you can specify the following elements (described below):

- Zero or more `RESOURCE` elements (optional).
- Zero or one `EXECUTION` elements (optional).
- Zero or one `TIMEOUT` elements (optional).
- Zero or one `PICTURE` elements (optional).
- Zero or one `OUTPUT` elements (optional).

Example:

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

For more information on the process of defining applications, refer to *Adding Applications*.

RESOURCE

The `RESOURCE` element defines a resource available for this application. For more information on defining resources, refer to the *Resources* section in this manual.

EXECUTION

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` 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 the parameters provided

on the command line should be ignored ("FALSE", default value) or provided to the DVM ("TRUE").

Settings defined locally within the `EXECUTION` element override settings defined in included execution components.

Possible execution elements include:

- Zero to many `ENVIRONMENT_VARIABLE` elements.
- Zero or one `PATH` element.
- Zero or one `DVM` element.
- Zero or one `MODULE` element.
- Zero or one `AUTHENTICATION` element.
- Zero or one `PARAMETERS` element.

For more information on defining execution elements, refer to *Setting the Execution Environment* section of this manual.

Usage Examples:

```
<EXECUTION Using="cpn.wa.execution.local" />
<EXECUTION Using="cpn.wa.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI>1</ENVIRONMENT_VARIABLE>
</EXECUTION>
<EXECUTION Id="params" AllowUrlParameters="TRUE">
  <PATH>/home/examples/BuiltIn/Arguments</PATH>
</EXECUTION>
```

TIMEOUT

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.

Possible timeout elements include:

- Zero or one `USER_AGENT` element.
- Zero or one `REQUEST_RESULT` element.
- Zero or one `DVM_AVAILABLE` element.

For more information on setting timeout values, refer to *Application Timeouts*.

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>
```

PICTURE

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.picture" />
<PICTURE>
  <PATH>$(connector.uri)/iiug/images</PATH>
</PICTURE>
```

For more information on specifying a picture component or a path, see *Web Application Picture Component - Configuration Reference*.

OUTPUT

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, refer to *Automatic Discovery of User Agent*.

The `OUTPUT` element can contain the following child element (described below):

- Zero or more MAP elements.

MAP

The `MAP` element is a combination of a rendering mechanism and a theme. It takes an 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. Common values are: DUA_WML11, DUA_HTML32, DUA_HTML40, DUA_DHTMLIE55, DUA_CHTML10 and DUA_FORMSML11. The list can be extended by custom choices.

It may also specify an optional attribute **Allowed**, which specifies whether this map will be used in this context or not. Possible values are TRUE (allowed) or FALSE (not allowed). For example, this can be used to forbid some application to use WML if they were not designed to.

The `MAP` element can contain the following child elements (described below):

- Zero or one RENDERING element.
- Zero or one THEME elements.

RENDERING / OUTPUT DRIVER

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` 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.

Usage Examples:

```
<RENDERING Using="cpn.rendering.wa" />
<RENDERING>
  <OUTPUT_DRIVER>JFE36</OUTPUT_DRIVER>
</RENDERING>
```

For more information on output drivers and rendering components, refer to *Application Rendering*.

THEME / TEMPLATE

The THEME elment defines the theme to be applied to the application. It takes an optional attribute **Using**, in which the unique identifier of a predefined WEB_APPLICATION_THEME_COMPONENT element can be specified.

The THEME element may contain TEMPLATE child elements, specifying the template(s) 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.

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 on defining templates and theme components, refer to *Defining a Theme Component*.

Service List - Configuration Reference

For a Web service 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. This section of the documentation outlines the rules for writing the XML that defines an application; for information about the general process of defining applications and groups, refer to the *Adding a Web Service Application* section of this manual.

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). It takes an attribute **MaxLicenseConsumption**, which takes an integer specifying the maximum number of licenses that can be consumed by all Web Services specified in the list of services contained within the `SERVICE_LIST` tags.

Syntax:

```
<SERVICE_LIST MaxLicenseConsumption="numLicenses">
  [ <GROUP ...> [...] ]
  [ <APPLICATION ...> [...] ]
</SERVICE_LIST>
```

Notes:

The `SERVICE_LIST` element may contain the following child elements (described below):

1. Zero or more `GROUP` elements (optional).
2. Zero or more `APPLICATION` elements (optional).

Example:

```
<CONFIGURATION>
  <APPLICATION SERVER>
    ...
    <SERVICE_LIST MaxLicenseConsumption="10">
      ...
    </SERVICE_LIST>
  </APPLICATION_SERVER>
</CONFIGURATION>
```

WARNING! 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.

GROUP

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>
```

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>
```

APPLICATION

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. The APPLICATION element can have up to four attributes defined:

- **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.
- **MaxLicenseConsumption** - A non-negative Integer specifying the maximum license consumption.

- **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.

Note: With the release of Genero 2.0, Web services are named applications as they host several Web services in one DVM.

When you define a Web service application, you can specify the following elements (described below):

- Zero or more `RESOURCE` elements (optional).
- Zero or one `EXECUTION` element (optional).
- Zero or more `TIMEOUT` elements (optional).

Example:

```
<APPLICATION Id="webapp" Parent="abswebapp">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)</PATH>
    <MODULE>webapp.42r</MODULE>
  </EXECUTION>
  <TIMEOUT> </TIMEOUT>
</APPLICATION>
```

RESOURCE

The `RESOURCE` element defines a resource available for this application. For more information on defining resources, refer to the *Resources* section in this manual.

EXECUTION

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.

Possible execution elements include:

- Zero or more ENVIRONMENT_VARIABLE elements.
- Zero or one PATH element.
- Zero or one DVM element.
- Zero or one MODULE element.
- Zero or one AUTHENTICATION element.
- Zero or one POOL element.

For more information on defining execution elements, refer to *Setting the Execution Environment*.

Usage Examples:

```
<EXECUTION Using="cpn.ws.execution.local" />
<EXECUTION Using="cpn.ws.execution.local">
  <ENVIRONMENT_VARIABLE Id="FGLGUI>1</ENVIRONMENT_VARIABLE>
</EXECUTION>
```

TIMEOUT

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.

Possible timeout elements include:

- Zero or one DVM_AVAILABLE element.
- Zero or one DVM_FREE element.
- Zero or one REQUEST_QUEUE element.
- Zero or one REQUEST_RESULT element.

For more information on setting timeout parameters, refer to *Application Timeouts.*

Usage Examples:

```
<TIMEOUT Using="cpn.ws.timeout.set1" />
```

```
<TIMEOUT>
  <DVM_AVAILABLE>10</DVM_AVAILABLE>
  <DVM_FREE>10</DVM_FREE>
  <REQUEST_QUEUE>10</REQUEST_QUEUE>
  <REQUEST_RESULT>240</REQUEST_RESULT>
</TIMEOUT>
```

Template Language Reference for the Snippet-Based Rendering Engine

Genero Web Client provides a template language to create dynamic templates. The template instructions, expressions, and paths allowed in the template file is linked to the rendering engine that processes the template. This reference is for those templates that will be rendered by the snippet-based rendering engine, the default rendering engine (starting with Genero Web Client 2.10). See *Application Rendering* for a discussion on application rendering and rendering options.

Topics

Due to the amount of information provided, each topic is covered on a separate page in the online version of this manual.

Template instructions and the GWC namespace

Genero Web Client instruction specify the kind of operations you can perform. This section presents the syntax for the Genero Web Client namespace and lists valid template instructions and valid syntax, and identifies the weight (processing priority) of each template instruction.

Template expressions

Genero Web Client template expressions are elements Genero Web Client template instructions can manipulate.

Template paths

Genero Web Client template paths provide access to Genero Web Client objects, such as the application server, Web server, and Genero application elements.

Template functions

Genero Web Client provides a snippet developer toolbox through a set of functions, suchs as logical iteration, number formatting, type conversion, snippet specific functions, and so on.

Template Instructions

Topics

- Genero Web Client namespace
- Template Instructions

Genero Web Client Namespace

A Genero Web Client template instruction is prefixed by "gwc". The Genero Web Client processes the template instruction and generates new HTML code.

Syntax

```
<tag gwc:instruction="expression" ... >
...
</tag>
```

Notes

1. `tag` is an HTML tag
2. *instruction* is any template instruction
3. *expression* is a template expression

Template Instructions

Element Instructions

GWC Template Instruction	Description
`gwc:snippet-root`	Delimit a snippet.

Attribute Instructions

GWC Template Instruction	Description
`gwc:define`	Define a variable to be used in the current tag or the children tags.
`gwc:condition`	Test the condition.

gwc:repeat	Repeat the children tags.
gwc:replace	Replace the entire tag.
gwc:attributes	Dynamically change an attribute of the current HTML tag.
gwc:content	Replace the text between the tags.
gwc:omit-tag	Suppress the surrounding tag after instructions in the children tags have been processed.
gwc:marks	Associate custom data to the element

With Genero Web Client instructions, as with XML, you can write attributes in any order. The processing of instructions, however, is interpreted in the order listed above, from the highest priority to the lowest. For example, once gwc:replace has been executed, there is no material to achieve the gwc:content processing.

---

snippet-root instruction

Replace the XHMTL DIV element by the rendering of the component. Snippet-root instructions are typically found in snippet XHTML files. See Application Rendering for more information on snippet files.

Syntax

```
<gwc:snippet-root>
  ...
</gwc:snippet-root>
```

Example

```
<gwc:snippet-root>
  <input type="text" readonly="readonly" class="gField gEdit"
   gwc:marks="
   field [CID];
   currentFieldStyle [CID,'gCurrentField'];
   modifiable isModifiable?[CID]:null;
   "
   gwc:attributes="
   class _tpl_+(isModifiable ? '':' gDisabled')+(color?'
gColor_'+color:'')+(hidden!=2?'':' gHidden');
   value value;
   size width;
   maxLength maxLength || null;
   title comment;
   "
  />
</gwc:snippet-root>
```

This defines the rendering of an Edit field.

define instruction

The define instruction declares a local variable that can be used in the HTML tag (the current element) and in its child elements. This variable is undefined outside of the current element.

Syntax

```
<tag gwc:define="var expression [; ...]" ...>
...
</tag>
```

Notes

1. *var* is the variable name
2. *expression* is a template expression

Example

```
<div gwc:define="i menu">
```

i is set to the current item of a menu.

---

condition instruction

If the expression is TRUE, render the element. If the expression is FALSE, remove the tag and its children.

Syntax

```
<tag gwc:condition="expression" ...>
...
</tag>
```

Notes

1. *expression* is a template expression
2. When using expressions in a `gwc:condition` instruction, verification is done on the value of the expression. The condition is true if the following test is true:

Expression Value Type	Test
Boolean	Expression value is true.

Numeric	Expression value is not 0. (zero)
String	String is not empty or "0". (zero)
Template path with identified node	The identified element exists (*i.e.* `formfield[edt1]`).
Template path with value	The value is not empty if it is a string, or 0 if it is a numeric or boolean value (*i.e.* `formfield[edt1]/id`).

---

repeat instruction

For a specified collection, iterate through the collection and repeat the element, once per item in the collection.

Syntax

```
<tag gwc:repeat="elt eltList">
    statements
</tag>
```

Notes

1. *eltList* is the list of elements to loop on (the collection of elements)
2. *elt* is an element of *eltList*
3. *statements* are statements repeated for each element *elt* in the collection *eltList*
4. See repeat template paths for special template paths used with a `gwc:repeat` instruction

Example

```
<div gwc:repeat="item menu/actions">
  <a href="..." gwc:condition="item/text"
     gwc:define="text item/text"
     gwc:attributes="href string:${document/URL}?${item/id}"
     gwc:content="text">Input</a>
</div>
```

Displays the text of each action of a menu as an HTML link.

Repeat instruction template paths

The following table lists special template paths for use with a `gwc:repeat` instruction. For more information on template paths, see Template Paths for the Snippet-Based Rendering Engine.

Template Path	Type	Component	AUI object
`repeat`	object		
`repeat/elt`	object		

Template Path	Type	Description	Attribute Type
`repeat/elt/length`	attribute	Get the length of the repeat sequence (the number of elements in the sequence).	number
`repeat/elt/index`	attribute	Get the index of the current element in the sequence, starting with 0 (zero).	number
`repeat/elt/first`	attribute	Test if the current repeat element is the first element of the sequence	boolean
`repeat/elt/last`	attribute	Test if the current repeat element is the last element of the sequence	boolean
`repeat/elt/even`	attribute	Test if the current repeat element is an even element of the sequence	boolean
`repeat/elt/odd`	attribute	Test if the current repeat element is an odd element of the sequence	boolean

Notes

1. *elt* is a repeat instruction element.

---

replace instruction

Replace the HTML tag (current element and its children) with the value of a template expression.

Syntax

```
<tag gwc:replace="expression" ...>
...
</tag>
```

Notes

1. *expression* can be any template expression

Example

```
<div gwc:replace="window"></div>
```

The `DIV` tag is replaced by the application window.

---

attributes instruction

This instruction dynamically sets a value to an attribute of the current tag. If the attribute does not exist, it will be created. If it exists, its value will be changed.

Syntax

```
<tag gwc:attributes="att expression [; ...] " ...>
...
</tag>
```

Notes

1. *att* is the attribute name
2. *expression* is the attribute value. Notice there is no equal sign (=) between the attribute name and the value being assigned to it.

Example

```
<form action="..." method="post" gwc:attributes="action document/URL ;
method string:get">
```

The action attribute is set to the document URL and the method attribute to get.

---

content instruction

Replace the element content (between the HTML tags) with the value of a template expression.

Syntax

```
<tag gwc:content="expression" ...>
...
</tag>
```

Notes

1. *expression* can be any template expression

Example

```
<div id="gForm-div" gwc:content="form" />
```

The content of the div tag is set to the generated code of the current form.

---

omit-tag instruction

If the expression evaluates to TRUE, replace the element by its content. If the expression is FALSE, remove the element.

Syntax

```
<tag gwc:omit-tag="expression"...>
...
</tag>
```

Notes

1. *expression* is any valid expression

Example

```
<div gwc:content="formfield[edt1]/value" gwc:omit-tag="true"></div>
```

This displays the value of the form field "edt1" and removes the `DIV` tag.

---

marks instruction

Associates data with the current element, in order to be used by JavaScript on the client side.

Syntax

```
<tag gwc:marks="mkp expression [; ...] " ...>
   ...
</tag>
```

Notes

1. *mkp* is the mark-up element name
2. *expr* is any valid expression

Example

```
<gwc:snippet-root>
  <input type="text" readonly="readonly" class="gField gEdit"
   gwc:marks="
   field [CID];
   currentFieldStyle [CID,'gCurrentField'];
   modifiable isModifiable?[CID]:null;
   "
   gwc:attributes="
   class _tpl_+(isModifiable ? '':' gDisabled')+(color?'
gColor_'+color:'')+(hidden!=2?'':' gHidden');
   value value;
   size width;
   maxLength maxLength || null;
   title comment;
   "
  />
</gwc:snippet-root>
```

This defines the rendering of an Edit field.

Template Expressions

Topics

- expressions syntax

---

expressions syntax

Template expressions respect a defined syntax.

Syntax

expression is:

```
{ conditional_expression
  | unary_op expression
  | expression op expression
  | expression_as_string in expression_as_string
  | [ argument_list ]
  | ( expression )
  | fct ( argument_list )
  | template_path
  | numeric
  | boolean
  | string
  | null
  | undefined
}
```

where *conditional_expression* is:

```
boolean_expression ? if_true : if_false
```

where *unary_op* is:

```
{ ! | + | - }
```

where *op* is:

```
{ arithmetic_op | boolean_op | comparison_op }
```

where *arithmetic_op* is:

```
{ * | / | % | + | - }
```

where *comparison_op* is:

```
{ < | <= | > | >= | == | != }
```

where *boolean_op* is:

```
{ && | || }
```

where *boolean* is:

```
{ TRUE | FALSE }
```

where *string* is:

```
' string '
```

where *argument_list* is a list of expressions separated by comas:

```
{ [ expression [, ...] ] }
```

argument_list can be empty.

Notes

1. *conditional_expression* returns a value according to the result of a condition expression

   ```
   boolean_expression ? if_true : if_false
   ```

2. If the *boolean_expression* results in true, the conditional_expression returns the if_true expression, otherwise if_false is returned.

   ```
   orientation=='horizontal'?'':' gRadioGroupVertical'  # produces
   an empty string if orientation equals
                                                        #
   horizontal, otherwise returns gRadioGroupVertical
   ```

3. *expression_as_string* is a string
4. [*argument_list*] defines an array

   ```
   [ 1, 2, 'a', 'b' ]      # defines an array with 4 elements
   ```

5. (*expression*) defines a priority in the processing of the expressions.

   ```
   (a + b) * c             # means expression 'a + b' will be
   processed before the result is multiplied by c
   ```

6. *fct* is any valid function name
7. *string* is a string value
8. *numeric* is a numeric value
9. *expression_as_string* is any template expression resulting in a string value
10. *template_path* is any valid template path

11. null expression removes a corresponding attribute

```
title comment || null;  # if comment attribute is not an empty
string
                        # then set comment value to title
otherwise
                        # remove the html attribute title
```

12. *undefined* is the value returned by a template path when the path is not available. It is used to make the difference between a template path that would return an empty string and a path that does not exist in the current context.
13. The `+` operator can be used with string values to concatenate two strings. Other operators will not work unless string values evaluate to numeric expressions:

```
'This expression ' + 'works'
'12' - '34'                     # works
'12' - 'ab'                     # doesn't work
```

14. The boolean operators `&&` and `||` behave like their JavaScript equivalent. The following table gives the result of the expression:

expr_A op *expr_B*

Operator	expr_A is TRUE*	expr_B is FALSE*
&&	expr_B	false
||	expr_A	expr_B

16. **Evaluation is done like the test made by the* `gwc:condition` *instruction*

```
1 > 0 && 'expr_A is true'       # produces expr_A is true
1 < 0 && 'expr_A is false'      # produces 0
1 < 0 || 'expr_A is false'      # produces expr_A is false
```

17. As the `&&` operator's priority is greater than the `||` operator's priority, you can combine these operators to have an if … then … else … statement :

condition_expr && *expr_if_true* || *expr_if_false*

```
true  && 'bill' || 'bob'        # produces bill
false && 'bill' || 'bob'        # produces bob
```

19. The `in` operator is used to look for a string value in a string value list. Elements of the list are separated by a space character

```
'bill' in 'bob bill john'       # produces true
```

Template Functions

Template functions are defining an utility toolbox for snippet developers.

- Conversion Functions
- Logical Functions
- Number Functions
- String Functions
- Front End Protocol Functions
- Other Functions

Conversion Functions

- bool()
- number()
- string()
- object()

---

bool()

The `bool()` function converts an expression to a boolean.

Syntax

```
bool(expr)
```

Notes

1. `expr` is the expression to convert.

---

number()

The `number()` function converts an expression to a number.

Syntax

```
number(expr)
```

Notes

1. `expr` is the expression to convert.

---

string()

The `string()` function converts an expression to a string.

Syntax

```
string(expr)
```

Notes

1. `expr` is the expression to convert.

---

object()

The `object()` function inserts XML code.

Syntax

```
object(expr)
```

Notes

1. `expr` is a string interpreted as an XML snippet. `expr` can be the XML code between the gwc:snippet-root tags. All the template instructions in this string is interpreted against the current context. Any parsing error detected is written in the log files, assuming the TEMPLATE category is set.

Example

```
<p gwc:replace="object(value) />
```

This insert the value of a field as XHTML code in the html page.

Logical Functions

- for()
- switch()

---

for()

The `for()` function creates a numeric sequence.

Syntax

```
bool(expr1, expr2, [expr3])
```

Notes

1. Return a numeric sequence from `number(expr1)` to `number(expr2)` with a step of `number(expr3)` [1 by default].

Example

```
<div gwc:repeat="i for(0, 20, 2)" gwc:content="i" />
```

---

switch()

The `switch()` function compares an expression to different values, and return a result dependant on which value it equals to.

Syntax

```
switch(expr1, array1, array2, [expr2])
```

Notes

1. If `expr1` is in `array1` at position *x*, return value inside `array2` at position *x*, if any. In other cases return `expr2`, if any.

Example

```
<div gwc:content="switch( number(i), [1, 2, 3, 4], ['this is one',
'this is two', 'this is three', 'this is four' ], 'Don\'t know that
number...')" />
```

Number Functions

- min()
- max()
- round()
- fill()
- precision()
- abs()
- sin()
- cos()
- tan()

---

min()

The `min()` function returns the minimum between two numbers.

Syntax

`min(expr1, expr2)`

Notes

1. Return the minimum between `number(expr1)` and `number(expr2)`.

---

max()

The `max()` function returns the maximum between two numbers.

Syntax

`max(expr1, expr2)`

Notes

1. Return the maximum between `number(expr1)` and `number(expr2)`.

---

round()

The `round()` function returns the rounded value of a number.

Syntax

`round(expr)`

Notes

1. Return the rounded value of `number(expr)`.

---

fill()

The `fill()` function formats a number by filling its string representation.

Syntax

`fill(expr1, expr2, [expr3, [expr4]])`

Notes

1. Return `string(number(expr1))` filled with `string(expr3)` [" " by default] to reach `number(expr2)` width.
   Fill is done according on `bool(expr4)` [true by default]: true means filling to left, false filling to right.

---

precision()

The `precision()` function sets precision when displaying a number.

Syntax

`precision(expr1, expr2)`

Notes

1. Set precision of `number(expr1)` decimal part to `number(expr2)`.

---

abs()

The `abs()` function calculates the absolute value.

Syntax

`abs(expr)`

Notes

1. Calculates the absolute value of `number(expr)`.

---

sin()

The `sin()` function calculates the sine of an angle.

Syntax

`sin(expr)`

Notes

1. Calculates the sine of `number(expr)`.
2. `number(expr)` is the angle in degrees.

---

cos()

The `cos()` function calculates the cosine of an angle.

Syntax

`cos(expr)`

Notes

1. Calculates the cosine of `number(expr)`.
2. `number(expr)` is the angle in degrees.

---

tan()

The `tan()` function calculates the tangent of an angle.

Syntax

`tan(expr)`

Notes

1. Calculates the tangent of `number(expr)`.
2. `number(expr)` is the angle in degrees.

String Functions

- escapeJS()
- translate()
- split()
- length()
- substring()
- contains()
- indexOf()
- replace()
- filterAmpersand()

---

escapeJS()

The `excapeJS()` function escapes JavaScript-sensitive characters.

Syntax

```
escapeJS(expr)
```

Notes

1. Return `string(expr)` with ["", "\", "\n", "\r"] characters escaped with a "\".

---

translate()

The `translate()` function translates a string.

Syntax

```
translate(expr1, expr2, expr3)
```

Notes

1. Return `string(expr1)` with occurrences of string in `string(expr2)` replaced by the string at the corresponding position in `string(expr3)`. Use expression with square brackets [] to replace a list of strings. Use quotes to replace character by character.

Examples

```
<div gwc:attributes="class translate(style, ' -/\#.=+()[]',
'____________')"/>
```
To replace character by character.

```
<div gwc:attributes="class translate(style, ['\\r\\n','\\n','\\r'],
'<br/>')"/>
```
To replace each strings in the array expr2 by the string expr3.

---

split()

The `split()` function splits a string using a delimiter.

Syntax

```
split(expr1, expr2)
```

Notes

1. Return `string(expr1)` split into an array of substrings according to `string(expr2)` delimiter.

Example

```
<option gwc:attributes="selected valueChecked in split(value , '|') ?
'selected' : null"/>
```

---

length()

The `length()` function returns string length.

Syntax

`length(expr)`

Notes

1. Return `string(expr)` length.

---

substring()

The `substring()` function returns a substring of a string.

Syntax

`substring(expr1, expr2, [expr3])`

Notes

1. Return `string(expr1)` substring from `number(expr2)` position to `number(expr3)` position [end of `string(expr1)` by default].

---

contains()

The `contains()` function returns true if a string contains a substring.

Syntax

```
contains(expr1, expr2)
```

Notes

1. Return true if `string(expr2)` is a substring of `string(expr1)`.

---

indexOf()

The `indexOf()` function returns the position of a substring in a string.

Sytnax

```
indexOf(expr1, expr2)
```

Notes

1. Return position of subtring `string(expr2)` in `string(expr1)` [return -1 if `string(expr2)` is not found in `string(expr1)`].

---

replace()

The `replace()` function replaces a specified part of a string with another string.

Syntax

```
replace(expr1, expr2, expr3)
```

Notes

1. Replace any location of `string(expr2)` in `string(expr1)` by `string(expr3)`.

---

filterAmpersand()

The `filterAmpersand()` function filters the ampersand (&) on a given string. Double ampersand (&&) are replaced by a simple one.

Syntax

```
filterAmpersand(expr)
```

Notes

1. This function is useful to filter labels containing GDC accelerators as defined in in the Genero Business Language documentation.

Example

```
<span gwc:content="filterAmpersand('&Save && exit'/>
```

will produce ...

```
<span>Save & exit</span>
```

Front End Protocol Functions

- makeScrollOffsetIDID()
- makePageSizeIDID()
- makeRowSelectionIDID()
- makeTableNoSortValue()
- makeColumnSortIDID()
- makeValueIDID()
- makeFocusXDID()
- makeFocusIDID()
- makeKeyXDID()
- makeKeyIDID()
- makeProcessingXDID()
- combEvents()
- makeSessionVarIDID()

---

makeScrollOffsetIDID()

The `makeScrollOffsetIDID()` function builds offset action.

Syntax

```
makeScrollOffsetIDID(expr1, expr2)
```

Notes

1. `expr1` is the table id.
2. `expr2` is a value for the offset.

Example

```
<input type="submit" gwc:attributes="name makeScrollOffsetIDID(ID,
repeat/p/index);... />
```
produces
```
<input type="submit" name="o88/50"... />
```

---

makePageSizeIDID()

The `makePageSizeIDID()` function builds page size action.

Syntax

```
makePageSizeIDID(expr1, expr2)
```

Notes

1. `expr1` is the table id.
2. `expr2` is a size value.

Example

```
<input type="submit" gwc:attributes="name makePageSizeIDID(ID,
'50');... />
```
produces
```
<input type="submit" name="s88/50"... />
```

---

makeRowSelectionIDID()

The `makeRowSelectionIDID()` function builds inclusive row selection action.

Syntax

```
makeRowSelectionIDID(expr1, expr2)
```

Notes

1. *expr1* is the table id or the selectorID for a matrix.
2. *expr2* is the row index.

Example

```
<input type="submit" gwc:attributes="name makeRowSelectionIDID(r/ID,
repeat/r/index);... />
or
<input type="submit" gwc:attributes="name
makeRowSelectionIDID(selectorID, selectorIndex);... />
```
produces
```
<input type="submit" name="r89/50"... />
```

---

makeTableNoSortValue()

The `makeTableNoSortValue()` function builds a reset value for table sort.

Syntax

```
makeTableNoSortValue()
```

---

makeColumnSortIDID()

The `makeColumnSortIDID()` function builds a column sort action in inclusive format

Syntax

```
makeColumnSortIDID(expr1,expr2)
```

Notes

1. expr1 is the table id, for example the template path *table/ID*.
2. expr2 is the sort value, for example the template path *table/ID* or the template function makeTableNoSortValue.

makeValueIDID()

The `makeValueIDID()` function builds a value in inclusive format

Syntax

```
makeValueIDID(expr1,expr2)
```

Notes

1. expr1 is the object id.
2. expr2 is the value of the field.

Example

```
<input type="submit" gwc:attributes="name makeValueIDID(ID, value)"
.../>
```
produces
```
<input type="submit" name="v89/tata"... />
```

makeFocusXDID()

The `makeFocusXDID()` function gives the focus to an item.

Syntax

```
makeFocusXDID()
```

Notes

1. the syntax is `control-name=control-value` where control-name is `makeFocusXDID()` and control-value is the field id.

Example

```
<input type="radio" gwc:attributes="name makeFocusXDID(); value ID;...
/>
```
produces
```
<input type="radio" name="f" value="89"... />
```

makeFocusIDID()

The `makeFocusIDID()` function gives the focus to an item.

Syntax

```
makeFocusIDID(expr)
```

Notes

1. `expr` is the field id.

Example

```
<input type="submit" gwc:attributes="name makeFocusIDID(ID);"... />
```
produces
```
<input type="submit" name="f/89"... />
```

makeKeyXDID()

The `makeKeyXDID()` function sends a key.

Syntax

```
makeKeyXDID(expr)
```

Notes

1. the syntax is `control-name=control-value` where control-name is `makeKeyXDID()` and control-value is the key.

Example

```
<input type="radio" gwc:attributes="name makeKeyXDID(); value 'tab';...
/>
```
produces
```
<input type="radio" name="f" value="tab".. />
```

makeKeyIDID()

The `makeKeyIDID()` function sends a key.

Syntax

```
makeKeyIDID(expr)
```

Notes

1. `expr` is the the key.

Example

```
<input type="submit" gwc:attributes="name makeKeyIDID('tab');"... />
```
produces
```
<input type="submit" name="f/tab"... />
```

---

makeProcessingXDID()

The `makeProcessingXDID()` function sends an action to get the page in process.

Syntax

```
makeKeyIDID()
```

Example

```
<input type="submit" gwc:attributes="name makeProcessingXDID();"... />
```
produces
```
<input type="radio" name="t" value=""... />
```

---

combEvents()

The `combEvents()` function combines multiple actions.

Syntax

```
combEvents(expr1, [expr2, ...])
```

Notes

1. Combine multiple action `string(expr1)`, `string(expr2)`,... into a single action.

---

makeSessionVarIDID()

The `makeSessionVarIDID()` function builds a session variable.

Syntax

```
makeSessionVarIDID(expr1,expr2)
```

Notes

1. `expr1` is the session variable name
2. `expr2` is the session variable value.
3. setting a variable to a empty string is equivalent to deleting the variable.

Example

```
<input type="hidden" gwc:attributes="name
makeSessionVarIDID('var1','value1')" />
```

Other Functions

- XPath()
- XPathConfig()
- ImageURI()
- includeSnippet()
- check()
- noOp()
- colorToRGB()

XPath()

The `XPath()` function returns xpath evaluation on current AUI tree context.
The result can only be a **string** not a node.

Syntax

```
xpath(expr)
```

Notes

1. Return evaluation of xpath string(expr) result on current AUI tree context.

Example

```
<span gwc:content="xpath('//Window[1]/@name')" />
```

This gives the name of the first window.

```
<div gwc:repeat="i for(1,application/ui/windows/length)" gwc:omit-
tag="true">
  <span gwc:condition="xpath('//Window[' + i + ']/Form/@name')"
gwc:content="' :: ' + ( xpath('//Window[' + i + ']/Form/@text') ||
xpath('//Window[' + i + ']/@text') || xpath('//Window[' + i +
']/Form/@name') ) " />
  <span gwc:condition="!xpath('//Window[' + i + ']/Form/@name')"
gwc:content="':: ' + xpath('//Window[' + i + ']/@name')" />
</div>
```

This gives the list of windows.

XPathConfig()

The `XPathConfig()` function returns xpath evaluation on current application configuration.

Syntax

```
xpathConfig(expr)
```

Notes

1. Return evaluation of xpath `string(expr)` result on current application configuration.

Example

```
<input id="gKeepaliveValue" gwc:attributes="value
XPathConfig('/APPLICATION/TIMEOUT/USER_AGENT/text()')" />
```

ImageURI()

The `ImageURI()` function builds the URI to the default image location.

Syntax

`imageURI(expr)`

Notes

1. Return URI to `string(expr)` default image location.

Example

```
<img gwc:attributes="src imageURI('accept')" alt="accept" />
```

includeSnippet()

The `IncludeSnippet()` function includes a snippet.

Syntax

`includeSnippet(expr)`

Notes

1. Include snippet with id `string(expr).`

Example

```
<div gwc:condition="application/state/ended"
gwc:content="includeSnippet('EndingPage')" />
```

check()

The `check()` function checks the validity of an expression.

Syntax

```
check(expr)
```

Notes

1. `expr` is the expression to be checked.

---

noOp()

The noOp() function evaluates an expression and returns true.

Syntax

```
noOp(expr)
```

Notes

1. `expr` is the expression to be validated.

---

colorToRGB()

The colorToRGB() function gets the RGB value of a Genero predefined color name.

Syntax

```
colorToRGB(expr)
```

Notes

1. `expr` is the name of a Genero color name.
2. If `expr` is not a predefined Genero color name, `expr` is returned.

Template Paths Overview

With template paths, you can access most elements in your application, as well as information about the Application Server.

- What is a Template Path?
- Template Paths detail

What is a Template Path?

A template path is an element that returns a typed value. Types can be one of the following:

Object

An object is an element that will handle some fields. Fields are sub-paths handling typed values as described here. Fields can be accessed using a relative path **fieldName,** or using the object relative or absolute path with the / operator : **objectPath/fieldName**.

Selectable Object

Some objects are selectable; an identifier is added to the path, making it possible to select one of the object instances according to a selector. The path will then look like :**objectPath[***expression***]**; *expression* is a selector identifying an object instance. The selector meaning will be unique for each **Selectable Object**.
When the object is referenced without using the selector, the default object instance is used. The default object instance's meaning will be unique for each **Selectable Object**.

Component

A component is an object that will be also associated with a snippet. This type of value could be used in an instruction asking for rendering (gwc:replace or gwc:content) as a call to the snippet corresponding to this component. This type of value could also be used in gwc:condition to test the existence or non-existence for this component.
A component is an object and will have some sub paths.
The snippet identifier used to render the component will be unique for each **Component**

Collection

A collection is a list of items. The type of the items will be unique, and will be one of the types listed.
A collection will be scanned using the gwc:repeat instruction.
Each collection also has an additional field **length**, to retrieve the number of items in the collection : **collectionPath/length**.

Attribute

An attribute is a literal value typed as a STRING, BOOLEAN, or NUMBER.

Template Paths Details

The template paths provided for your use are documented according to the associated object. See the following topics for details about the various template paths:

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy

Template Paths - Server hierarchy

The Server object

Path: `server`

The **server** is a unique object that represents the GWC rendering engine instance.

Fields	Type	Description
`server/version`	Attribute (string)	Returns the version of the server.
`server/development`	Attribute (boolean)	Indicate if the server is in development mode (not in production). Templates and snippets are reloaded automatically in development mode.
`server/production`	Attribute (boolean)	Indicate if the server is in production mode (not in development). Templates and snippets are **not** reloaded automatically in production mode.

Example

```
<title gwc:content="(application/ui ? application/ui/text : '') + ' - '
+ server/version">Application name</title>
```

This displays the GWC engine version in the browser title bar.

Template Paths - Document hierarchy

The Document object

Path: `document`

The **document** is a unique object that represents the HTML document that is currently rendered.

Fields	Type	Description
`document/url`	Attribute (string)	Specifies the URL of the next document to get. Used to set the action attribute of a form HTML tag.
`document/dialog`	Attribute (string)	Describes the JavaScript structure handling 4GL dialog structure used by smart JavaScript framework.
`document/components`	Attribute (string)	A string describing a JavaScript structure used by smart JavaScript framework.
`document/marks`	Attribute (string)	A string describing a JavaScript structure used by smart JavaScript framework.
`document/errors`	Attribute (string)	A string containing all errors handled by the Rendering Engine during the current page computation (snippet parsing, expression parsing, expression evaluation, and so on).
`document/uploadUrl`	Attribute (string)	A URL to send the files to upload. Example: *Usage in a FORM tag in snippet FileUpload.xhtml for Set1.* `<form method="post" enctype="multipart/form-data" ... gwc:attributes="action document/UploadUrl ..." >`
`document/blobUrl`	Attribute (string)	A URL to retrieve documents from the DVM. Example: *Display a link to download a report.* `<a gwc:attributes="href document/blobUrl + '/report.pdf'">Click to download report</a>`

`document/auiUrl`	Attribute (string)	A URL to retrieve the Abstract User Interface tree. Example: *Display the AUI tree in a new window* `<a gwc:attributes="href document/auiUrl" target="_blank">gtree</a>`
`document/request`	Object	A request object handling the request sent by the Web browser.

The Request object

Path: `document/request`

The **request** is a unique sub-object of the document that represents the HTTP request that provokes this document rendering.

Fields	Type	Description
`document/request/header[`*`header-name`*`]`	Selectable attribute (string)	A string containing the selected header value. Example : *An element to render only on Firefox* `<div gwc:condition="contains(document/request/header['user-agent'], 'gecko')>You use Firefox</div>`
`document/request/headers`	Collection of header objects	A collection of all headers handled in the request.

The Header object

The **header** objects are scanned through the `document/request/headers` collection.

Fields	Type	Description
`name`	Attribute (string)	A string containing the header name.
`value`	Attribute (string)	A string containing the header value.

Example

```
<div gwc:repeat="h document/request/headers">
  <b gwc:content="h/name"/>=<i gwc:content="h/value" />
</div>
```

Displays debug information.

Template Paths - Application hierarchy

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
    - **/application**
        - **/application/state**
            - **/application/state/processing**
            - **/application/state/ended**
                - **/application/state/ended/error**
        - **/application/meta**
            - **/application/meta/launch**
            - **/application/meta/filetransfer**
            - **/application/meta/variable[...]**
        - **/application/ui**
            - /application/ui/startmenu
            - /application/ui/topmenu
            - /application/ui/toolbar
            - **/application/ui/stylelist**
            - /application/ui/window[..]
            - /application/ui/topMostNormalWindow
        - **/application/interrupt**

The Application object

Path: `application`

Snippet ID: `Application`

Corresponding AUI Tree element: *AUI Root Element*

The **Application** is a unique object that represents the 4GL application currently running and used to rendered the current document.

This object value is rendered thru the *Application* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields	Type	Description
`application/CID`	Attribute (string)	The corresponding component identifier.
`application/type`	Attribute (string)	Type of the component.
`application/connectorUri`	Attribute (string)	Connector part of the URI

`application/state`	Object	An ApplicationState object handling the state of the application.
`application/meta`	Object	An ApplicationMetaInformation object handling miscellaneous information from the application depending to its state.
`application/ui`	Object	A UserInterface object handling the application abstract user interface description.
`application/interrupt`	Object	An Interrupt object handling the interruption of the application.

The ApplicationState object

Path: `application/state`

The **ApplicationState** is a unique sub-object from the Application object that represents the application state.

With the introduction of the snippet-based rendering engine, the main template can be used to surface the application regardless of its state, eliminating the need for individual template files for each application state. A change in the application state (interactive, processing, and so on) will be handled in the main template using the application state paths listed below.

Fields	Type	Description
`application/state/interactive`	Attribute (boolean)	DVM state interactive or not.
`application/state/processing`	Object	A Processing object describing the processing state if any.
`application/state/ended`	Object	An Ended object describing the ended state if any.

The Processing object

Path: `application/state/processing`

The **Processing** is a unique sub-object from ApplicationState object that represents the application processing state.

Fields	Type	Description
`application/state/processing/transactionPending`	Attribute (boolean)	True if the application server is still processing the request.

The Ended object

Path: `application/state/ended`

The **Ended** is a unique sub-object from ApplicationState object that represents the application ended state.

Fields	Type	Description
`application/state/ended/normal`	Attribute (boolean)	True if the application has properly ended.
`application/state/ended/timeout`	Attribute (boolean)	True if the application has not been launch within DVM_AVAILABLE timeout.
`application/state/ended/error`	Object	An EndedError object describing the application error that provokes the application end.

The EndedError object

Path: `application/state/ended/error`

The **EndedError** is a unique sub-object from Ended object that represents the application error that provokes the application end.

Fields	Type	Description
`application/state/ended/error/message`	Attribute (string)	Message displayed when the application stopped with an error.

The ApplicationMetaInformation object

Path: `application/meta`

The **ApplicationMetaInformation** is a unique sub-object from Application object that represents the application meta information.

Information related to the application and independent of the application state [file transfer, launch] are called meta-information. This type of information can be used through the meta-information paths listed below.

Fields	Type	Description
`application/meta/launch`	Object	An ApplicationLaunch object describing the sub-application currently launched.
`application/meta/filetransfer`	Object	A FileTransfer object describing the files transfers sessions currently active.
`application/meta/variables`	Collection of Variable objects	A list of session variables.

The ApplicationLaunch object

Path: `application/meta/launch`

The **ApplicationLaunch** is a unique sub-object from ApplicationMetaInformation object that represents the sub application currently launched.

When a new application is launched, Information from the new application are retrieved with the paths below.

Fields	Type	Description
`application/meta/launch/url`	Attribute (string)	URL of the new application launched.

The FileTransfer object

Path: `application/meta/filetransfer`

The **FileTransfer** is a unique sub-object from ApplicationMetaInformation object that handles files available through the File Transfer.

Fields	Type	Description
`application/meta/filetransfer/files`	Collection of File objects	The list of all files in the application. Used in a repeat instruction to loop on each file in the list.
`application/meta/filetransfer/currentfiles`	Collection of File objects	The list of the current files in the application [i.e. files transferred since previous action]. Used in a repeat instruction to loop on each file in the list.

The File object

The **File** object is a handled by the *files* and *currentfiles* collections from FileTransfers object. It describes one transfered file.

Fields	Type	Description
`url`	Attribute (string)	URL to retrieve the file from the GAS file transfer directory.
`name`	Attribute (string)	Name of the file.
`size`	Attribute (number)	Size of the file in byte.
`isCurrent`	Attribute (boolean)	True if the file is part of the current file transfer process.

The Variable object

Path: `application/meta/variable[name]`

Session variables are held by a collection. As a result, there is a collection path `application/meta/variables` to iterate through the full collection of variables, as well as a selector path to access variables by name.

Fields	Type	Description
`application/meta/variable/XDID`	Attribute (string)	Name of the session variable for JavaScript usage.
`application/meta/variable/name`	Attribute (string)	Name of the session variable.
`application/meta/variable/value`	Attribute (string)	Value of the session variable.
`application/meta/variable/readOnly`	Attribute (boolean)	True if the session variable is read-only.

Example

```
<h3>List of current session variables</h3>
  <div gwc:omit-tag="true" gwc:repeat="var application/meta/variables">
  <p>
    <span gwc:content="var/name" style="color:red;"/><br/>
    <span gwc:content="'XDID='+ var/XDID" /><br/>
    <span gwc:content="'value='+ var/value" /><br/>
    <span gwc:content="'readOnly='+ var/readOnly" /><br/>
  </p>
</div>
```

For more details about session variables usage, see the chapter Session Variables and Cookies.

The UserInterface object

Path: `application/ui`

Snippet ID: `UserInterface`

Corresponding AUI Tree element: *UserInterface*

The **UserInterface** is a unique sub-object from Application object that represents the application abstract User Interface description.

This object value is rendered thru the *UserInterface* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields	Type	Description
`application/ui/CID`	Attribute (string)	The corresponding component identifier
`application/ui/text`	Attribute (string)	UserInterface text node value.
`application/ui/dbDate`	Attribute (string)	Application date format.
`application/ui/intermediateTrigger`	Attribute (string)	Indicates if the intermediate trigger should be executed. This corresponds to the Dialog.fieldOrder entry in the fglprofile.
`application/ui/error`	Attribute (string)	Error message
`application/ui/startmenu`	Object	A StartMenu object describing the StartMenu component attached to the application.
`application/ui/topmenu`	Object	A TopMenu object describing the TopMenu component attached to the application.
`application/ui/toolbar`	Object	A Toolbar object describing the Toolbar component attached to the application.
`application/ui/styleList`	Object	A StyleList object describing the application styles.
`application/ui/windows`	Collection of Window objects	The list of windows owned by the application. Used in a repeat instruction to loop on each window in the list.
`application/ui/window[window-name]`	Object	This selectable path will return the Window object of the application with this given name. If no selection is explicitely done. This path will return the Window object describing the current active window for this application.
`application/ui/modalWindows`	Collection of Window objects	The list of modal windows owned by the application. *N.B.* this is a sub-list of the `application/ui/windows` one.

		Used in a repeat instruction to loop on each window in the list.
`application/ui/topMostNormaWindow`	Object	A Window object which describes the most recent window with a non-modal (aka normal) style.

The StyleList object

Path: `application/ui/styleList`

Snippet ID: `StyleList`

Corresponding AUI Tree element: *StyleList*

The **StyleList** is a unique sub-object from UserInterface object that represents the application styles (see 4ST files in BDL manual).

This object value is rendered thru the *StyleList* snippet when invoked in a gwc:replace or gwc:content instruction.
To access sub-element of the StyleList object there is currently no paths. You have either to use xpath() function or XSLT snippets.

Fields	Type	Description
`application/ui/styleList/CID`	Attribute (string)	The corresponding component identifier.
`application/ui/styleList/type`	Attribute (string)	Type of the component.
`application/ui/styleList/name`	Attribute (string)	Style name.

The Interrupt object

Path: `application/interrupt`

Snippet ID: `LocalAction`

Corresponding AUI Tree element: *UIDS2/LocalAction*

Fields	Type	Description
`application/interrupt/CID`	Attribute (string)	The corresponding component identifier
`application/interrupt/type`	Attribute (string)	Type of the component.
`application/interrupt/DID` **Deprecated !**	Attribute (string)	The corresponding dialog identifier. This path is **deprecated** use XDID instead.
`application/interrupt/XDID`	Attribute (string)	Dialog identifier in exclusive format. **Example:** `<input type="radio" gwc:attributes="name application/interrupt/XDID; value '';... />` produces `<input type="radio" name="x" value=""... />` In inclusive format, no value is needed. **Example:** `<input type="submit" gwc:attributes="name application/interrupt/XDID;"... />` produces `<input type="submit" name="x"... />`
`application/interrupt/name`	Attribute (string)	The interrupt identifier.
`application/interrupt/text`	Attribute (string)	The displayed text.
`application/interrupt/image`	Attribute (string)	The associated image.
`application/interrupt/comment`	Attribute (string)	The associated comment.
`application/interrupt/isActive`	Attribute (boolean)	True if the Interrupt is enabled.

Template Paths - StartMenu hierarchy

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
    - /application
        - /application/ui
            - **/application/ui/startmenu**

The StartMenu object

Path: `application/ui/startmenu`

Snippet ID: `StartMenu`

Corresponding AUI Tree element: *StartMenu*

The **StartMenu** is a unique sub-object from the UserInterface object that represents the application Start Menu if any.

This object value is rendered thru the *StartMenu* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`application/ui/startmenu/CID`	Attribute (string)	The corresponding component identifier.
`application/ui/startmenu/type`	Attribute (string)	Type of the component.
`application/ui/startmenu/name`	Attribute (string)	StartMenu identifier.
`application/ui/startmenu/text`	Attribute (string)	StartMenu title.
`application/ui/startmenu/hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`application/ui/startmenu/items`	Collection of StartMenuItem objects	The list of all items in the StartMenu. Used in a repeat instruction to loop on each file in the list.

The StartMenuItem objects

StartMenuItem objects are in fact one of the following type :

- a StartMenuCommand object ;
- a StartMenuGroup object ;
- a StartMenuSeparator object ;

The StartMenuCommand object

Path: no absolute path available

Snippet ID: `StartMenuCommand`

Corresponding AUI Tree element: *StartMenuCommand*

The **StartMenuCommand** is an object implementing a StartMenuItem object that represents a StartMenu command.

This object value is rendered thru the *StartMenuCommand* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`id`	Attribute (string)	Command identifier.
`DID` **Deprecated !**	Attribute (string)	Dialog identifier. This path is **deprecated** use IDID or XDID instead.

`IDID`	Attribute (string)	Dialog identifier in inclusive format..
		For example:
		`<input type="submit" gwc:attributes="name IDID;".. />`
		Produces:
		`<input type="submit" name="m/89"... />`

`XDID`	Attribute (string)	Dialog identifier in exclusive format. For example: `<input type="radio" gwc:attributes="name XDID; value ID;... />`

		Produces: `<input type="radio" name="m" value="89"... />`
`name`	Attribute (string)	StartMenu command identifier.
`text`	Attribute (string)	Text to be displayed for this command.
`image`	Attribute (string)	The associated image
`disabled`	Attribute (boolean)	True if the command is disabled
`isActive`	Attribute (boolean)	True if the command is active
`exec`	Attribute (string)	Command line to be executed.
`waiting`	Attribute (boolean)	True if the command is started without waiting.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.

The StartMenuGroup object

Path: no absolute path available

Snippet ID: `StartMenuGroup`

Corresponding AUI Tree element: *StartMenuGroup*

The **StartMenuGroup** is an object implementing a StartMenuItem object that represents a group of StartMenu items.

This object value is rendered thru the *StartMenuGroup* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.

`name`	Attribute (string)	StartMenu identifier.
`text`	Attribute (string)	Text displayed for the StartMenu group.
`image`	Attribute (string)	Image display for this StartMenu group.
`disabled`	Attribute (boolean)	True if the group is not active.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`items`	Collection of StartMenuItem objects	The list of all items in the StartMenu group. Used in a repeat instruction to loop on each file in the list.

The StartMenuSeparator object

Path: no absolute path available

Snippet ID: `StartMenuSeparator`

Corresponding AUI Tree element: *StartMenuSeparator*

The **StartMenuSeparator** is an object implementing a StartMenuItem object that represents a separator.

This object value is rendered thru the *StartMenuSeparator* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.

Template Paths - TopMenu hierarchy

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
    - /application
        - /application/ui
            - **/application/ui/topmenu**

The TopMenu object

Path: `application/ui/topmenu` or `application/ui/window/form/topmenu`

Snippet ID: `TopMenu`

Corresponding AUI Tree element: *TopMenu*

The **TopMenu** is a unique sub-object from the UserInterface or Form object that represents the application or window top menu if any.

This object value is rendered thru the *TopMenu* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.
`name`	Attribute (string)	TopMenu identifier.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the attribute TAG value.
`items`	Collection of TopMenuItem objects	The list of all items in the top menu. Used in a repeat instruction to loop on each file in the list.

The TopMenuItem objects

TopMenuItem objects are in fact one of the following type :

- a TopMenuCommand object ;
- a TopMenuGroup object ;
- a TopMenuSeparator object ;

The TopMenuCommand object

Path: no absolute path available

Snippet ID: `TopMenuCommand`

Corresponding AUI Tree element: *TopMenuCommand*

The **TopMenuCommand** is an object implementing a TopMenuItem object that represents a TopMenu command.

This object value is rendered thru the *TopMenuCommand* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.
`id`	Attribute (string)	Object identifier.
`DID` **Deprecated !**	Attribute (string)	Dialog identifier. This path is **deprecated** use IDID or XDID instead.
`IDID`	Attribute (string)	Dialog identifier with inclusive format.
`XDID`	Attribute (string)	Dialog identifier with exclusive format.
`name`	Attribute (string)	TopMenu command identifier.
`text`	Attribute (string)	Text displayed for this command.

`comment`	Attribute (string)	Comment displayed for this command.
`image`	Attribute (string)	The associated image.
`isActive`	Attribute (boolean)	True if the command is active.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the attribute TAG value.

The TopMenuGroup object

Path: no absolute path available

Snippet ID: `TopMenuGroup`

Corresponding AUI Tree element: *TopMenuGroup*

The **TopMenuGroup** is an object implementing a TopMenuItem object that represents a group of TopMenu items.

This object value is rendered thru the *TopMenuGroup* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.
`name`	Attribute (string)	TopMenu group identifier.
`text`	Attribute (string)	Text displayed for the TopMenu group.
`comment`	Attribute (string)	Comment displayed fot the TopMenu group.
`image`	Attribute (string)	Image associated to the TopMenu group

`isActive`	Attribute (boolean)	True of the TopMenu group is active.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the attribute TAG value.
`items`	Collection of TopMenuItem objects	The list of all items in the TopMenu group. Used in a repeat instruction to loop on each file in the list.

The TopMenuSeparator object

Path: no absolute path available

Snippet ID: `TopMenuSeparator`

Corresponding AUI Tree element: *TopMenuSeparator*

The **TopMenuSeparator** is an object implementing a TopMenuItem object that represents a separator.

This object value is rendered thru the *TopMenuSeparator* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.

Template Paths - Toolbar hierarchy

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
    - /application
        - /application/ui
            - **/application/ui/toolbar**

The Toolbar object

Path: `application/ui/toolbar` or `application/ui/window/form/toolbar`

Snippet ID: `Toolbar`

Corresponding AUI Tree element: *Toolbar*

The **Toolbar** is a unique sub-object from the UserInterface or Form object that represents the application or window tools bar if any.

This object value is rendered thru the *Toolbar* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.
`name`	Attribute (string)	ToolBar identifier.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the attribute TAG value.
`buttonTextHidden`	Attribute (number)	Set to 1 if the texts associated to the ToolBar items are not displayed.
`items`	Collection of ToolbarItem objects	The list of all items in the top menu. Used in a repeat instruction to loop on each file in the list.

The ToolbarItem objects

ToolbarItem objects are in fact one of the following type :

- a ToolbarCommand object ;
- a ToolbarSeparator object ;

The ToolbarCommand object

Path: no absolute path available

Snippet ID: `ToolbarItem`

Corresponding AUI Tree element: *ToolbarItem*

The **ToolbarCommand** is an object implementing a ToolbarItem object that represents a Toolbar command.

This object value is rendered thru the *ToolbarItem* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.
`id`	Attribute (string)	Object identifier.
`DID` **`Deprecated !`**	Attribute (string)	Dialog identifier. This path is **deprecated** use IDID or XDID instead.
`IDID`	Attribute (string)	Dialog identifier with inclusive format.
`XDID`	Attribute (string)	Dialog identifier with exclusive format.
`name`	Attribute (string)	ToolBar command identifier.
`text`	Attribute (string)	Text displayed for this command.
`comment`	Attribute	Comment displayed for this command.

	(string)	
`image`	Attribute (string)	Image associated to this command.
`isActive`	Attribute (boolean)	True if the command is active.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL..
`tag`	Attribute (string)	Corresponds to the attribute TAG value.
`buttonTextHidden`	Attribute (number)	Set to 1 if text associated to the command should not be displayed.

The ToolbarSeparator object

Path: no absolute path available

Snippet ID: `ToolbarSeparator`

Corresponding AUI Tree element: *ToolbarSeparator*

The **ToolbarSeparator** is an object implementing a ToolbarItem object that represents a separator.

This object value is rendered thru the *ToolbarSeparator* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.

Template Paths - Window hierarchy

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
- /application
    - /application/ui
        - **/application/ui/window**
            - **/application/ui/window/menu**
            - **/application/ui/window/dialog**
            - **/application/ui/window/form**
                - /application/ui/window/form/toolbar
                - /application/ui/window/form/topmenu
            - **/application/ui/window/localaction**
            - **/application/ui/window/idle**

The Window object

Path: `application/ui/window` or `application/ui/topMostNormalWindow` or also relative paths according to sequences.

Snippet ID: `Window`

Corresponding AUI Tree element: *Window*

The **Window** is a sub-object of UserInterface object that represents one of the application window/frame.

This object value is rendered thru the *Window* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type (see ... for more details)
`name`	Attribute (string)	Window identifier.
`message`	Attribute (string)	Window message.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.

`localActions`	Collection of LocalAction Objects	
`localActions/length`	Attribute (number)	Number of items in the collection.
`class`	Attribute (string)	Corresponds to attribute STYLE value.
`menu`	Object	A Menu object handling the menu of the application if any. *N.B.* `dialog` and `menu` are mutually exclusive.
`dialog`	Object	A Dialog object handling the dialog description of the application if any. *N.B.* `dialog` and `menu` are mutually exclusive.
`form`	Object	A Form object handling the layout description of the application.

The Menu object

Path: `window/menu`

Snippet ID: `Menu`

Corresponding AUI Tree element: *Menu*

The **Menu** is a unique sub-object of Window object that represent the main menu of the application. This object will only exist if the current window is handling a `MENU` statement. In all other cases including `DIALOG` statements, the window is handling a Dialog object

This object value is rendered thru the *Menu* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type (see ... for more details)
`idle`	Object	A Idle object when the ON IDLE instruction is used.
`text`	Attribute	Menu title.

	(string)	
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.
`class`	Attribute (string)	Corresponds to the attribute STYLE value.
`image`	Attribute (string)	Image associated to the Menu.
`comment`	Attribute (string)	Comment associated to the Menu.
`actions`	Collection of Action objects	The list of all actions available Used in a repeat instruction to loop on each window in the list.
`visibleActions`	Collection of Action objects	The list of actions available that do not have an explicit Action View (as TopMenu commands or Toolbar commands for instance). *N.B.* this is a sub-list of the `actions` one. Used in a repeat instruction to loop on each actions in the list.

The Dialog object

Path: `window/dialog`

Snippet ID: `Dialog`

Corresponding AUI Tree element: *Dialog*

The **Dialog** is a unique sub-object of Window object that represent this window interaction. This object will only exist if the current window is handling another statements than `MENU`.

This object value is rendered thru the *Dialog* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute	Object type.

	(string)	
`idle`	Object	A Idle object when the ON IDLE instruction is used.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.
`standAloneMatrixDetected`	Attribute (boolean)	True if the dialog contains a matrix and is in interactive mode (display array or input array).
`pageSize`	Attribute (number)	Number of rows displayed by the matrix.
`size`	Attribute (number)	Number of rows in the matrix.
`currentRow`	Attribute (number)	The matrix current row.
`offset`	Attribute (string)	The matrix offset.
`actions`	Collection of Action objects	The list of all actions available Used in a repeat instruction to loop on each window in the list.
`visibleActions`	Collection of Action objects	The list of actions available that do not have an explicit Action View (as TopMenu commands or Buttons for instance. *N.B.* this is a sub-list of the `actions` one. Used in a repeat instruction to loop on each window in the list.

The Form object

Path: `window/form`

Snippet ID: `Form`

Corresponding AUI Tree element: *Form*

The **Form** is a unique sub-object of Window object that represent the form embed in this window if any.

This object value is rendered thru the *Form* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`name`	Attribute (string)	Form identifier.
`text`	Attribute (string)	Form title.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.
`topmenu`	Object	TopMenu of the Form.
`toolbar`	Object	Toolbar of the Form.
`item`	Object	A LayoutContainer object handling the layout of the form.
`minHeight`	Attribute (number)	Corresponds to the 4GL attribute MINHEIGHT.
`minWidth`	Attribute (number)	Corresponds to the 4GL attribute MIWIDTH.

The Action object

Path: actions are accessed relatively thru Collection iteration

Snippet ID: `Action` or `MenuAction`

Corresponding AUI Tree element: *Action*

The **Action** is a sub-object of Menu or Dialog object that represent the form embed in this window if any.

This object value is rendered thru the *Action* (if issue from or Dialog object) or *MenuAction* (if issue from or Menu object) snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
CID	Attribute (string)	The corresponding component identifier.
type	Attribute (string)	Type of the component.
id	Attribute (string)	Object identifier.
DID **Deprecated !**	Attribute (string)	Dialog identifier. This path is **deprecated** use IDID or XDID instead.
IDID	Attribute (string)	Dialog identifier with inclusive format.
XDID	Attribute (string)	Dialog identifier with exclusive format.
name	Attribute (string)	Action identifier
text	Attribute (string)	Text displayed for this action.
image	Attribute (string)	Image associated to the action.
comment	Attribute (string)	Comment displayed for this action.
isActive	Attribute (boolean)	True if the action is active.
hidden	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
defaultView	Attribute (string)	The action defaulView value.
hasCustomView	Attribute (boolean)	True if the action has another view (for example in ToolBar or TopMenu)
hasFocus	Attribute (boolean)	True if the action has the focus.

The LocalAction object

Path: application/ui/window/localAction

Snippet ID: LocalAction

Corresponding AUI Tree element:
UIDS2/UserInterface/Window/LocalActionList/LocalAction

Local Actions are actions handled by the **Front End only**. For more details on the list of Local Actions see the Genero BDL manual, chapter "Interaction Model". There are some Local Actions not handled by GWC: editcopy, editcut, editpaste.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`id`	Attribute (string)	Object identifier.
`DID` **`Deprecated !`**	Attribute (string)	Dialog identifier. This path is **deprecated** use IDID or XDID instead.
`IDID`	Attribute (string)	Dialog identifier with inclusive format.
`XDID`	Attribute (string)	Dialog identifier with exclusive format.
`name`	Attribute (string)	Action identifier
`text`	Attribute (string)	Text displayed for this action.
`image`	Attribute (string)	Image associated to the action.
`comment`	Attribute (string)	Comment displayed for this action.
`isActive`	Attribute (boolean)	True if the action is active.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`defaultView`	Attribute (string)	The action defaulView value.
`hasCustomView`	Attribute (boolean)	True if the action has another view (for example in ToolBar or TopMenu)

The Idle object

Path: `dialog/idle or menu/idle`

Snippet ID:

Corresponding AUI Tree element:

Fields	Type	Description
`DID` **`Deprecated !`**	Attribute (string)	Dialog identifier. This path is **deprecated** use XDID instead.

`XDID`	Attribute (string)	Dialog identifier in exclusive format.
		Example:
		`<input type="radio" gwc:attributes="name makeIdleXDID(); value '';... />`
		produces
		`<input type="radio" name="i" value=""... />`
		In inclusive format, no value is needed.
		Example:
		`<input type="submit" gwc:attributes="name makeIdleXDID();"... />`
		produces
		`<input type="submit" name="i"... />`

Template Paths - Layout hierarchies

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
- /application
    - /application/ui
        - /application/ui/window
            - /application/ui/window/menu
            - /application/ui/window/dialog
            - /application/ui/window/form
                - **/application/ui/window/form/grid**
                    - **/application/ui/window/form/grid/gridlayout**
                - **/application/ui/window/form/scrollgrid**
                    - **/application/ui/window/form/grid/gridlayout**
                - **/application/ui/window/form/group[...]**
                - **/application/ui/window/form/folder[...]**
                - **/application/ui/window/form/vbox[...]**
                - **/application/ui/window/form/hbox[...]**
                - **/application/ui/window/form/table[...]**
                - **/application/ui/window/form/table[...]/action**

The LayoutContainer objects

LayoutContainer objects are in fact one of the following type :

- a Grid object ;
- a ScrollGrid object ;
- a Group object ;
- a Folder object ;
- a VBox or HBox object ;
- a Table object ;

These objects are selectable objects. They can be accessed using paths like **objectPath[*id*]**, where *id* is the object name.

Example

```
application/ui/window/form/grid['grid1']
application/ui/window/form/scrollgrid['sg1']
application/ui/window/form/group['grp1']
application/ui/window/form/folder['fd1']
application/ui/window/form/vbox['vb1']
application/ui/window/form/hbox['hb1']
application/ui/window/form/table['table1']
```

The Grid object

Path: `application/ui/window/form/grid[name]` or relative paths according to sequences.

Snippet ID: `Grid`

Corresponding AUI Tree element: *Grid*

The **Grid** is a LayoutContainer.

This object value is rendered thru the *Grid* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`name`	Attribute (string)	Grid identifier.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the attribute TAG value.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.
`class`	Attribute (string)	Corresponds to the attribute STYLE value.
`layout`	Object	A GridLayout object describing the layout for this given grid.

The ScrollGrid object

Path: `application/ui/window/form/scrollgrid[name]` or relative paths according to sequences.

Snippet ID: `ScrollGrid`

Corresponding AUI Tree element: *ScrollGrid*

The **ScrollGrid** is a LayoutContainer.

This object value is rendered thru the *ScrollGrid* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`name`	Attribute (string)	ScrollGrid identifier.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the attribute TAG value.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.
`class`	Attribute (string)	Corresponds to the attribute STYLE value.
`offset`	Attribute (number)	ScrollGrid offset
`size`	Attribute (number)	Size of the associated array.
`pageSize`	Attribute (number)	Number of rows displayed by the scrollGrid.
`isScrollable`	Attribute (boolean)	True if you can scroll in the array.
`pages`	Collection of untyped object	A collection of untyped object to be able to iterate on the ScrollGrid pages.

`layout`	Object	A GridLayout object describing the layout for this given grid.

The GridLayout object

Path: `grid/layout` or `scrollgrid/layout`

Snippet ID: `GridLayout`

The **GridLayout** is a sub-oject of Grid or ScrollGrid describing the grid layout.

This object value is rendered thru the *GridLayout* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`GID`	Attribute (string)	GridLayout identifier
`data`	Attribute (string)	Data to handle the layout design for JavaScript.
`width`	Attribute (number)	GridLayout width.
`height`	Attribute (number)	GridLayout height.
`marginLeft`	Attribute (number)	Width of the smallest empty cell on the left hand side of the grid. If one line of the grid begins with a occupied cell, the margin is 0.
`marginRight`	Attribute (number)	Width of the smallest empty cell on the right hand side of the grid. If one line of the grid ends with a occupied cell, the margin is 0.
`lines`	Collection of GridLayoutLine objects	A collection of grid lines.
`tableLines`	Collection of GridLayoutTableLine objects	A collection of grid table lines. Enable the layout of a grid content to be fully based on a HTML table. Compared to the `lines` collection, the rowspan attribute of a TD element can be directly mapped to the cell's height attribute.

The GridLayoutLine object

Path: relative paths according to sequences.

The **GridLayoutLine** is a sub-object of GridLayout describing one grid line.

Fields (Attributes)	Type	Description
`cells`	Collection of GridLayoutCell objects	List of cells containg a widget.
`allcells`	Collection of GridLayoutCell objects	List of all cells having a widget or not.

The GridLayoutTableLine object

Path: relative paths according to sequences.

The **GridLayoutTableLine** is a sub-object of GridLayout describing one grid line.

Fields (Attributes)	Type	Description
`cells`	Collection of GridLayoutCell objects	List of cells.

The GridLayoutCell object

Path: relative paths according to sequences.

The **GridLayoutCell** is a sub-object of GridLayoutLine describing one grid cell.

Fields (Attributes)	Type	Description
`GID`	Attribute (string)	
`isEmpty`	Attribute (boolean)	True if the cell does not contain a widget.
`x`	Attribute	The cell x coordinate in the Grid.

	(number)	
`y`	Attribute (number)	The cell y coordinate in the Grid.
`width`	Attribute (number)	The cell width.
`height`	Attribute (number)	The cell height.
`item`	Object	A LayoutContainer object or a GridElem object filling this grid cell.

The Group object

Path: `application/ui/window/form/group[name]` or relative paths according to sequences.

Snippet ID: `Group`

Corresponding AUI Tree element: *Group*

The **Group** is a LayoutContainer.

This object value is rendered thru the *Group* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`name`	Attribute (string)	Group identifier.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute value.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.

class	Attribute (string)	Corresponds to the STYLE attribute value.
item	Object	A LayoutContainer object describing the content of the group.

The Folder object

Path: application/ui/window/form/folder[name] or relative paths according to sequences.

Snippet ID: Folder

Corresponding AUI Tree element: *Folder*

The **Folder** is a LayoutContainer.

This object value is rendered thru the *Folder* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
CID	Attribute (string)	The corresponding component identifier.
type	Attribute (string)	Type of the component.
name	Attribute (string)	Folder identifier
text	Attribute (string)	Folder title.
hidden	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
tag	Attribute (string)	Corresponds to the TAG attribute value.
style[attribute-name]	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.
class	Attribute (string)	Corresponds to the STYLE attribute value.
pages	A collection	The list of all pages in this folder.

	of FolderPage object	
`page[page-name]`	Object	This selectable path will return the FolderPage object of the folder with this given name. If no selection is explicitely done. This path will return the current FolderPage object.

The FolderPage object

Path: relative paths according to sequences.

The **FolderPage** is a sub-object of Folder describing one folder page.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type.
`id`	Attribute (string)	Object identifier.

`IDID`	Attribute (string)	Dialog identifier in inlcusive format.
		For example:
		`<input type="submit" gwc:attributes="name p/IDID;"... />`
		produces:
		`<input type="submit" name="p88/89"... />`

`XDID`	Attribute (string)	Dialog identifier in exclusive format.
		For example:
		`<input type="radio" gwc:attributes="name p/XDID; value p/ID;... />`
		produces:
		`<input type="radio" name="p88" value="89".. />`

`name`	Attribute (string)	FolderPage identifier
`text`	Attribute (string)	Folder page title.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute	Corresponds to the TAG attribute value.

	(string)	
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.
`class`	Attribute (string)	Corresponds to the STYLE attribute value.
`image`	Attribute (string)	Corresponds to the IMAGE attribute value.
`comment`	Attribute (string)	Corresponds to the COMMENT attribute value.
`isCurrent`	Attribute (boolean)	True if the FolderPage is selected.
`hasAction`	Attribute (boolean)	True if the FolderPage has an associated Action.
`body`	Object	A LayoutContainer object describing this page content.

The Box object

Path: `application/ui/window/form/vbox[name]` or `application/ui/window/form/hbox[name]` or relative paths according to sequences.

Snippet ID: `VBox` or `HBox`

Corresponding AUI Tree element: *VBox* or *HBox*

The **Box** is a LayoutContainer.

This object value is rendered thru the *VBox* or *HBox* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`name`	Attribute (string)	LayoutContainer identifier.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute value.

`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.
`class`	Attribute (string)	Corresponds to the STYLE attribute value.
`items`	Collection of LayoutContainer objects	A list of the layout element to render vertically or horizontally.

The Table object

Path: `application/ui/window/form/table[name]` or relative paths according to sequences.

Snippet ID: `Table`

Corresponding AUI Tree element: *Table*

The **Table** is a LayoutContainer.

This object value is rendered thru the *Table* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Type of the component.
`id`	Attribute (string)	Table identifier.
`name`	Attribute (string)	Table name.
`offset`	Attribute (number)	Table offset.
`scrollOffsetXDID`	Attribute (string)	Offset of the Table in exclusive format. See the function makeScrollOffsetIDID to use the inclusive format. **Syntax:** `control-name=control-value` where `control-name` is scrollOffsetXDID and `control-value` is an offset value.

		Example:
		`<input type="radio" gwc:attributes="name pageSizeXDID; value '50';... />`
		produces
		`<input type="radio" name="s88" value="50"... />`

`pageSize`	Attribute (number)	Number of rows displayed by the Table.

`pageSizeXDID`	Attribute (string)	Number of rows displayed by the Table in exclusive format. See the function makePageSizeIDID to use the inclusive format.
		Syntax:
		`control-name=control-value`
		where `control-name` is pageSizeXDID and `control-value` is a size value.
		Example:
		`<input type="radio" gwc:attributes="name pageSizeXDID; value '50';... />`
		produces
		`<input type="radio" name="s88" value="50"... />`

`size`	Attribute (number)	Number of rows in the underlying array.
`isSelectable`	Attribute (boolean)	True if you can select a row of the Table (for example in a display)
`isScrollable`	Attribute (boolean)	True if you can scroll the Table rows (for example in a input).
`isSortable`	Attribute (boolean)	True if the attribute UNSORTABLECOLUMNS si set to false.
`isModifiable`	Attribute (boolean)	True if you can type in the table (for example in an input).
`sortType`	Attribute (string)	Value is "asc" if the sorting is ascendant. Value is "desc" if the sorting is descendant.
`currentRow`	Attribute (number)	Index of the selected line.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Customizing with Genero Presentation Styles for more details.
`class`	Attribute (string)	Corresponds to the STYLE attribute value.
`columns`	Collection of Table Column objects	A list of the columns in this table.

rows	Collection of Table Row objects	A list of the rows in this table.
action	Object	A TableAction object describing the action associated with the double-click user action for this table.

The Table Column object

Path: relative paths according to sequences.

Corresponding AUI Tree element: *TableColumn*

The **TableColumn** is a sub-object from the Table describing one column header.

Fields (Attributes)	Type	Description
DID **Deprecated !**	Attribute (String)	Dialog identifier. This path is **deprecated** use IDID or XDID instead.

IDID	Attribute (string)	Dialog identifier in inclusive format.
		For example:
		`<input type="submit" gwc:attributes="name c/IDID;"... />`
		where *c* is the column object
		produces:
		`<input type="submit" name="c88/89"... />`

XDID	Attribute (string)	Dialog identifier in exclusive format.
		For example:
		`<input type="radio" gwc:attributes="name c/XDID; value c/ID;... />`
		where *c* is the column object
		produces:
		`<input type="radio" name="c88" value="89"... />`

columnSortXDID	Attribute (string)	Dialog identifier in exclusive format. See also function makeTableNoSortValue.
text	Attribute (string)	Column title.
name	Attribute (string)	Column identifier.
isModifiable	Attribute (boolean)	True if the columns is in input mode.
isSortable	Attribute (boolean)	True is the attribute UNSORTABLE is set to false.
isSorted	Attribute	True if the Column is already sorted.

	(boolean)	
hidden	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
tag	Attribute (string)	Corresponds to the TAG attribute value.

The Table Row object

Path: relative paths according to sequences.

Corresponding AUI Tree element: *TableColumn/ValueList*

The **TableRow** is a sub-object from the Table describing one row content.

Fields (Attributes)	Type	Description
DID **Deprecated!**	Attribute (string)	Dialog identifier. This path is **deprecated** use XDID path or makeRowSelectionDID function instead.

XDID	Attribute (string)	Dialog identifier in exclusive format. To use the inclusive format see the function makeRowSelectionIDID.
		Syntax:
		`control-name=control-value`
		where the `control-name` is the row id and `control-value` is the row index.
		Example:
		`<input type="radio" gwc:attributes="name r/XDID; value repeat/r/index;... />`
		produces
		`<input type="radio" name="r89" value="50"... />`

id	Attribute (string)	Row identifier.
isCurrent	Attribute (boolean)	True if the row is selected.
cells	Collection of Table Cell objects	A list of the cells in this table row.

The Table Cell object

Path: relative paths according to sequences.

Corresponding AUI Tree element: *TableColumn/ValueList/Value*

The **TableCell** is a sub-object from the TableRow describing one cell content.

Fields (Attributes)	Type	Description
`hidden`	Attribute (number)	True if the Cell is not visible.
`item`	Object	A Widget object describing the widget filling this cell.
`column`	Object	A Table Column object describing the column of the current cell.

The TableAction object

Path: `table/action`

Snippet ID: `TableAction`

Corresponding AUI Tree element:

The **TableAction** is a sub-object of Table object that represent the action to activate when the user double-click on a table row.

This object value is rendered thru the *TableAction* snippet when invoked in a gwc:replace or gwc:content instruction.

Fields (Attributes)	Type	Description
`CID`	Attribute (string)	The corresponding component identifier.
`type`	Attribute (string)	Object type (see ... for more details)
`DID` **`Deprecated!`**	Attribute (string)	Dialog identifier. This path is **deprecated** use XDID path or makeValueIDID function instead.
`name`	Attribute (string)	Action name.
`text`	Attribute (string)	Text displayed for the Action.
`image`	Attribute (string)	Image assoiated to the Action.
`comment`	Attribute (string)	Comment for the Action.

`isActive`	Attribute (boolean)	True if the Action is active.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Value of the TAG attribute.

Template Paths - Widgets hierarchies

- The Server hierarchy
- The Document hierarchy
- The Application hierarchy
- /application
    - /application/ui
        - /application/ui/window
            - /application/ui/window/menu
            - /application/ui/window/dialog
            - /application/ui/window/form
                - **/application/ui/window/form/staticlabel[...]**
                - **/application/ui/window/form/staticimage[...]**
                - **/application/ui/window/form/hline**
                - **/application/ui/window/form/button[...]**
                - **/application/ui/window/form/hboxtag[...]**
                - **/application/ui/window/form/formfield[...]**
                    - **/application/ui/window/form/formfield[...]/label[...]**
                    - **/application/ui/window/form/formfield[...]/image[...]**
                    - **/application/ui/window/form/formfield[...]/edit[...]**
                    - **/application/ui/window/form/formfield[...]/buttonedit[...]**
                    - **/application/ui/window/form/formfield[...]/textedit[...]**
                    - **/application/ui/window/form/formfield[...]/slider[...]**
                    - **/application/ui/window/form/formfield[...]/combobox[...]**
                    - **/application/ui/window/form/formfield[...]/radiogroup[...]**
                    - **/application/ui/window/form/formfield[...]/checkbox[...]**
                    - **/application/ui/window/form/formfield[...]/timeedit[...]**
                    - **/application/ui/window/form/formfield[...]/spinedit[...]**
                    - **/application/ui/window/form/formfield[...]/progessbar[...]**
                    - **/application/ui/window/form/formfield[...]/canvas[...]**

The Grid Elements objects

Grid Elements objects are in fact one of the following type :

- a StaticLabel object ;
- a StaticImage object ;
- a FormField object ;
- a HLine object ;
- a Button object ;
- a HBoxTag object ;
- a HBoxTagCell object ;

Formfield is a selectable object. It can be accessed using a path like **objectPath[*id*]**, where *id* is the Formfield name.

Example

```
application/ui/window/form/formfield['formonly.r1']
```

The Widgets objects

Widgets objects are in fact one of the following type :

- a Label object ;
- a Image object ;
- a Edit object ;
- a ButtonEdit object ;
- a TextEdit object ;
- a DateEdit object ;
- a Slider object ;
- a ComboBox object ;
- a RadioGroup object ;
- a CheckBox object ;
- a TimeEdit object ;
- a SpinEdit object ;
- a ProgressBar object ;
- a Canvas object;

The StaticLabel object

Path: `application/ui/window/form/staticLabel[name]` or relative paths according to sequences.

Snippet ID: `StaticLabel`

Corresponding AUI Tree element: *Label*

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Object type (see ... for more details)
`name`	Attribute (string)	StaticLabel name.
`text`	Attribute (string)	Correspond to the TEXT attribute in 4GL.
`comment`	Attribute (string)	Corresponds to the COMMENT attribute in 4GL.
`justify`	Attribute (string)	Corresponds to the JUSTIFY attribute in 4GL.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute in 4GL.
`class`	Attribute (string)	Corresponds to the STYLE attribute in 4GL.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.

The StaticImage object

Path: `application/ui/window/form/staticLabel[name]` or relative paths according to sequences.

Snippet ID: `StaticImage`

Corresponding AUI Tree element: *Image*

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Object type (see ... for more details)
`name`	Attribute (string)	StaticLabel identifier
`image`	Attribute (string)	Correspond to the IMAGE attribute in 4GL.
`width`	Attribute (number)	Corresponds to the WIDTH attribute in 4GL.
`height`	Attribute (number)	Corresponds to the HEIGHT attribute in 4GL.
`stretch`	Attribute (string)	Corresponds to the STRETCH attribute in 4GL.
`autoscale`	Attribute (string)	Corresponds to the AUTOSCALE attribute in 4GL.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute in 4GL.
`class`	Attribute (string)	Corresponds to the STYLE attribute in 4GL.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.

The FormField object

Path: `application/ui/window/form/formfield[name]` or relative paths according to sequences.

Snippet ID: `FormField`

Corresponding AUI Tree element: *FormField*

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Type of the component.
`selectorID`	Attribute (string)	Id of the FormField in matrix.
`selectorDID` **Deprecated !**	Attribute (string)	Dialog identifier. This function is **deprecated** use selectorXDID instead.

`selectorXDID`	Attribute (string)	Dialog identifier in exclusive format. See the function makeRowSelectionIDID to use the inclusive format.
		Syntax:
		`control-name=control-value`
		where `control-name` is selectorXDID and `control-value` is the row index.
		Example:
		`<input type="radio" gwc:attributes="name selectorXDID; value selectorIndex;... />`
		produces
		`<input type="radio" name="r89" value="50"... />`

`isSelected`	Attribute (boolean)	True if the FormField is selected.
`selectorIndex`	Attribute (number)	Index of FormField in a matrix.
`isFirstColumn`	Attribute (boolean)	True if the FormField is in the first column.
`item`	Object	The FormField Widget.

The HLine object

Path: grid/hline

Snippet ID: `HLine`

Corresponding AUI Tree element: *HLine*

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Object type (see ... for more details)

The Button object

Path: relative paths according to sequences.

Snippet ID: `Button`

Corresponding AUI Tree element: *Button*

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Object type (see ... for more details)
`name`	Attribute (string)	StaticLabel identifier
`text`	Attribute (string)	Text displayed for the Button.
`comment`	Attribute (string)	Comment displayed for the Button.
`image`	Attribute (string)	Correspond to the IMAGE attribute in 4GL.
`isActive`	Attribute (boolean)	True if the Button is active.
`hasFocus`	Attribute (boolean)	True if the Button has focus.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute in 4GL.
`class`	Attribute (string)	Corresponds to the STYLE attribute in 4GL.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.

The HBoxTag object

Path: relative paths according to sequences.

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Object type (see ... for more details).
`layoutData`	Attribute (string)	
`width`	Attribute (number)	
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`spacerWidth`	Attribute (number)	Width of the spacers.
`cells`	Collection of HBoxTagCell objects	List of HBoxTagCell that contains an object.
`allCells`	Collection of HBoxTagCell objects	List of all HBoxTagCell objects even empty cells.

The HBoxTagCell object

Path: grid/hline

Snippet ID: `HLine`

Corresponding AUI Tree element: *HLine*

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Object type (see ... for more details)

item	Object	Can be any GridElement object.
isEmpty	Attribute (boolean)	True if the HBoxTagCell is empty.

The Common fields

Path: formfield/objet/field

Fields (Attributes)	Type	Description
CID	Attribute (number)	Component identifier.
DID **Deprecated !**	Attribute (integer)	Dialog identifier. This path is **deprecated** use XDID path or makeValueIDID function instead.

XDID	Attribute (string)	Dialog identifier in exclusive format. To use the inclusive format see the function makeValueIDID.
		Example:
		`<input type="radio" gwc:attributes="name XDID; value value"... />`
		produces:
		`<input type="radio" name="v89" value="tata"... />`

id	Attribute (integer)	Object identifier.
type	Attribute (string)	Object type.
name	Attribute (string)	The Widget name.
comment	Attribute (string)	Corresponds to the COMMENT attribute of the Widget.
hidden	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
tag	Attribute (string)	Corresponds to the TAG attribute of the Widget.
class	Attribute (string)	Corresponds to the STYLE attribute of the Widget.
style[attribute-name]	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.

`value`	Attribute (string)	Corresponds to the VALUE attribute in the AUI tree. See also makeValueIDID to build a value in inclusive format.
`isModifiable`	Attribute (boolean)	True if you can enter a value for the widget (for example in input mode).
`hasFocus`	Attribute (boolean)	True id the Widget has the focus.
`noEntry`	Attribute (number)	Corresponds to the NOENTRY attribute in 4GL.
`notNull`	Attribute (number)	Corresponds to the NOT NULL attribute in 4GL.
`isRequired`	Attribute (boolean)	True if the REQUIRED attribute is set in 4GL.
`isQuery`	Attribute (boolean)	Corresponds to the QUERYEDITABLE in 4GL.
`isSelectable`	Attribute (boolean)	True if the Widget can be selected.
`width`	Attribute (number)	Corresponds to the WIDTH attribute in 4GL.
`queryCleanerValue`	Attribute (string)	
`tabIndex`	Attribute (number)	Corresponds to the TABINDEX attribute in 4GL.

The Label object

Path: formfield/label

Snippet ID: `Label`

Corresponding AUI Tree element: *FormField/Label*

Fields (Attributes)	Type	Description
`isNumeric`	Attribute (boolean)	True if the label value represents a number.
`justify`	Attribute (string)	Corresponds to the JUSTIFY attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The Image object

Path: relative paths according to sequences.

Snippet ID: `Image`

Corresponding AUI Tree element: *formfield/image*

Fields (Attributes)	Type	Description
`image`	Attribute (string)	Correspond to the IMAGE attribute in 4GL.
`height`	Attribute (number)	Corresponds to the HEIGHT attribute in 4GL.
`stretch`	Attribute (string)	Corresponds to the STRETCH attribute in 4GL.
`autoscale`	Attribute (string)	Corresponds to the AUTOSCALE attribute in 4GL.
`sizePolicy`	Attribute(string)	Corresponds to the SIZEPOLICY attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The Edit object

Path: relative paths according to sequences.

Snippet ID: `Edit`

Corresponding AUI Tree element: *formfield/edit*

Fields (Attributes)	Type	Description
`maxLength`	Attribute (number)	Size of the field.
`isPassword`	Attribute (boolean)	True if the INVISIBLE attribut is present.
`isNumeric`	Attribute (boolean)	True if the field value is a number.

`justify`	Attribute (string)	Corresponds to the JUSTIFY attribute in 4GL.
`picture`	Attribute (string)	Corresponds to the PICTURE attribute in 4GL.
`shift`	Attribute (string)	Value is "up" if UPSHIFT attribute is present. Value is "down" if DOWNSHIFT attribute is present.
`century`	Attribute (string)	Corresponds to the CENTURY attribute in 4GL.
`include`	Attribute (string)	Corresponds to the INCLUDE attribute in 4GL.
`verify`	Attribute (string)	Corresponds to the VERIFY attribute in 4GL.
`autonext`	Attribute (string)	Corresponds to the AUTONEXT attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The ButtonEdit object

Path: relative paths according to sequences.

Snippet ID: `ButtonEdit`

Corresponding AUI Tree element: *formfield/buttonedit*

The ButtonEdit has the same fields than the Edit widget.

Fields (Attributes)	Type	Description
`edit`	Object	Use the Edit.xhtml snippet to render the edit part of the ButtonEdit.
`button`	Object	

List of fields for the Button object of ButtonEdit object.

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.

`DID` **`Deprecated!`**	Attribute (number)	Dialog identifier. This path is **deprecated** use XDID path or makeValueIDID function instead.
`type`	Attribute (string)	Object type (see ... for more details)
`name`	Attribute (string)	Button identifier.
`comment`	Attribute (string)	Corresponds to the COMMENT attribute in 4GL.
`hidden`	Attribute (number)	Corresponds to the HIDDEN attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute in 4GL.
`class`	Attribute (string)	Corresponds to the STYLE attribute in 4GL.
`style[ attribute-name ]`	Attribute (string)	This selectable path will return the given attribute value inherited for this window. See Genero Presentation Styles for more details.
`image`	Attribute (string)	Corresponds to the IMAGE attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The TextEdit object

Path: relative paths according to sequences.

Snippet ID: `TextEdit`

Corresponding AUI Tree element: *formfield/textedit*

Fields (Attributes)	Type	Description
`maxLength`	Attribute (number)	Size of the field.
`height`	Attribute (number)	Corresponds to the number of lines displayed.
`isNumeric`	Attribute (boolean)	True if the field value is a number.

`stretch`	Attribute (string)	Corresponds to the STRETCH attribute in 4GL.
`scrollbars`	Attribute (string)	Corresponds to the SCROLLBARS attribute in 4GL.
`shift`	Attribute (string)	Value is "up" if UPSHIFT attribute is present. Value is "down" if DOWNSHIFT attribute is present.
`wantReturn`	Attribute (boolean)	True if the WANTNORETURNS attribute in 4GL is not set.
`wantTabs`	Attribute (string)	Corresponds to the WANTTABS attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The DateEdit object

Path: relative paths according to sequences.

Snippet ID: `DateEdit`

Corresponding AUI Tree element: *formfield/dateedit*

Fields (Attributes)	Type	Description
`maxLength`	Attribute (number)	Size of the field.
`justify`	Attribute (string)	Corresponds to the JUSTIFY attribute in 4GL.
`picture`	Attribute (string)	Corresponds to the PICTURE attribute in 4GL.
`format`	Attribute (string)	Corresponds to the FORMAT attribute in 4GL.
`century`	Attribute (string)	Corresponds to the CENTURY attribute in 4GL.
`include`	Attribute (string)	Corresponds to the INCLUDE attribute in 4GL.
`verify`	Attribute (string)	Corresponds to the VERIFY attribute in 4GL.
`autonext`	Attribute (string)	Corresponds to the AUTONEXT attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The Slider object

Path: relative paths according to sequences.

Snippet ID: `Slider`

Corresponding AUI Tree element: *FormField/Slider*

Fields (Attributes)	Type	Description
`orientation`	Attribute (string)	Corresponds to the ORIENTATION attribute in 4GL.
`step`	Attribute (number)	Corresponds to the STEP attribute in 4GL.
`valueMin`	Attribute (number)	Corresponds to the VALUEMIN attribute in 4GL.
`valueMax`	Attribute (number)	Corresponds to the VALUEMAX attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The ComboBox object

Path: relative paths according to sequences.

Snippet ID: `ComboBox`

Corresponding AUI Tree element: *FormField/ComboBox*

Fields (Attributes)	Type	Description
`valueText`	Attribute (string)	Value of the ComboBox object.
`queryEditable`	Attribute (boolean)	Corresponds to the QUERYEDITABLE attribute in 4GL.
`hasEmptyItem`	Attribute (boolean)	True if the NOT NULL in 4GL is not set.
`items`	Collection of ChoiceItem objects	A list of choices.

`shift`	Attribute (string)	Corresponds to the SHIFT attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The RadioGroup object

Path: relative paths according to sequences.

Snippet ID: `RadioGroup`

Corresponding AUI Tree element: *FormField/RadioGroup*

Fields (Attributes)	Type	Description
`items`	Collection of ChoiceItem objects	A list of choices.
`orientation`	Attribute (string)	Corresponds to the ORIENTATION attribute in 4GL.
`hasEmptyItem`	Attribute (boolean)	True if the NOT NULL in 4GL is not set.

Common attributes for this object are available in the Common Fields section.

The ChoiceItem object

Path: relative paths according to sequences.

Fields (Attributes)	Type	Description
`text`	Attribute (string)	Displayed text for the item.
`name`	Attribute (string)	Item identifier.

Common attributes for this object are available in the Common Fields section.

The CheckBox object

Path: relative paths according to sequences.

Snippet ID: `CheckBox`

Corresponding AUI Tree element: *FormField/CheckBox*

Fields (Attributes)	Type	Description
`text`	Attribute (string)	Corresponds to the TEXT attribute in 4GL.
`valueChecked`	Attribute (string)	Corresponds to the VALUECHECKED attribute in 4GL.
`valueUnchecked`	Attribute (string)	Corresponds to the VALUEUNCHECKED attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The TimeEdit object

Path: relative paths according to sequences.

Snippet ID: `TimeEdit`

Corresponding AUI Tree element: *FormField/TimeEdit*

The TimeEdit object has all the attributes listed in the Common Fields.

Common attributes for this object are available in the Common Fields section.

The SpinEdit object

Path: relative paths according to sequences.

Snippet ID: `SpinEdit`

Corresponding AUI Tree element: *FormField/SpinEdit*

Fields (Attributes)	Type	Description
`step`	Attribute (number)	Corresponds to the STEP attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The ProgressBar object

Path: relative paths according to sequences.

Snippet ID: `ProgressBar`

Corresponding AUI Tree element: *FormField/ProgressBar*

Fields (Attributes)	Type	Description
`valueMin`	Attribute (number)	Corresponds to the VALUEMIN attribute in 4GL.
`valueMax`	Attribute (number)	Corresponds to the VALUEMAX attribute in 4GL.

Common attributes for this object are available in the Common Fields section.

The Canvas object

Path: relative paths according to sequences.

Snippet ID: `Canvas`

Corresponding AUI Tree element: *FormField/Canvas*

This object also have common items, see Common fields for more details

Fields (Attributes)	Type	Description
`height`	Attribute (number)	Corresponds to the height of the canvas.
`items`	Collection of CanvasItem objects	A list of CanvasItem which type could be: *CanvasArc, CanvasCircle, CanvasLine, CanvasOval, CanvasPolygon, CanvasRectangle, CanvasText*

`items` has a length attibute that indicates the number of CanvasItems in the list.

The CanvasItem object

Path: relative paths according to sequences.

A CanvasItem is of type: *CanvasArc, CanvasCircle, CanvasLine, CanvasOval, CanvasPolygon, CanvasRectangle, CanvasText.* The type also defines the Snipet ID.

Fields (Attributes)	Type	Description
`CID`	Attribute (number)	Component identifier.
`type`	Attribute (string)	Type of the component.
`name`	Attribute (string)	Name of the object.
`comment`	Attribute (string)	Corresponds to the COMMENT attribute in 4GL.
`tag`	Attribute (string)	Corresponds to the TAG attribute in 4GL.
`startX`	Attribute (integer)	Corresponds to the startX attribute of a Canvas element in 4GL.
`startY`	Attribute (integer)	Corresponds to the startY attribute of a Canvas element in 4GL.
`endX`	Attribute (integer)	Corresponds to the endX attribute of a Canvas element in 4GL.
`endY`	Attribute (integer)	Corresponds to the endY attribute of a Canvas element in 4GL.
`xyList`	Attribute (string)	Corresponds to the xyList attribute of a Canvas element in 4GL.
`startDegrees` **`CanvasArc and CanvasCircle Only !`**	Attribute (integer)	Corresponds to the startDegrees attribute of a Canvas element in 4GL.
`extentDegrees` **`CanvasArc and CanvasCircle Only !`**	Attribute (integer)	Corresponds to the extenDegrees attribute of a Canvas element in 4GL.
`diameter` **`CanvasArc and CanvasCircle Only !`**	Attribute (integer)	Corresponds to the diameter attribute of a Canvas element in 4GL.
`text` **`CanvasText Only !`**	Attribute (string)	Corresponds to the text attribute of a Canvas element in 4GL.
`anchor`	Attribute	Corresponds to the anchor attribute of a Canvas

CanvasText Only !	(string)	element in 4GL.
acceleratorKey1	Attribute (string)	Corresponds to the acceleratorKey1 attribute of a Canvas element in 4GL.
acceleratorKey3	Attribute (string)	Corresponds to the acceleratorKey3 attribute of a Canvas element in 4GL.
fillColor	Attribute (string)	Corresponds to the fillColor attribute of a Canvas element in 4GL.

For more details about elements attributes, please refer to the Genero Business Development Language documentation on Canvas.

Migrating from GAS 2.10.x or GWC 2.10.x

This section discusses tasks that you must complete when migrating from Genero 2.10.x to a later version.

Topics

- Application configuration
- Template and snippets
- Deprecated functions and paths

Application configuration

Add `noNamespaceSchemaLocation` attribute in external application configuration file

All external application configuration files must be updated by adding the `noNamespaceSchemaLocation` attribute, as described below:

The original Edit.xcf:

```
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
  </EXECUTION>
</APPLICATION>
```

The new Edit.xcf:

```
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION Parent="defaultgwc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/2.11/cfextwa.x
sd">
  <EXECUTION>
    <PATH>$(res.path.fgldir.demo)/Widgets</PATH>
  </EXECUTION>
</APPLICATION>
```

If this attribute is missing, the corresponding application will fail to start, and the following message will be written to the log file:

```
Can't find 'noNamespaceSchemaLocation' attribute in external
application file '/home/f4gl/gwc/app/Edit.xcf'.
```

Output drivers for Internet Explorer

Specific output drivers DUA_AJAX_HTML and DUA_PAGE_HTML have been added to support certain features (such as the Canvas widget) on Internet Explorer. As a result, all customized snippets specified for DUA_AJAX will also need to be specified for DUA_AJAX_HTML.

The original CardStep1.xcf:

```
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.x
sd">
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.x
sd">
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 behaviours see chapter 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 tag followed by the URL parameters.

Template and snippets

Main template

$FGLASDIR/tpl/set1/main.xhtml has changed, mainly concerning the JavaScript part. If you have customized this template file, it is recommended that you use the new main.xhtml and add your modifications to it.

Style management

Styles management has changed for all the widgets. Template paths style['allInlines4ST'] and style['allClasses4ST'] have been added. They are shortcuts to the styles management. For more information, see the chapter on Genero Presentation Styles

---

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

Migrating to GWC 2.10

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. While that rendering engine will continue to render your applications, all new development and advances will be focused on the snippet-based rendering engine introduced with GWC 2.10. It is recommended that you utilize this new rendering engine.

Topics

- How to use the legacy built-in rendering engine
- What you should do before migrating to the snippet-based rendering engine

---

How to use the legacy built-in rendering engine

To deploy your application using the legacy built-in rendering engine, you need to ensure the OUTPUT_DRIVER is set to `GWC`.

To run a specific application with the legacy built-in rendering engine when the application is mapped to run with AJAX mode, you would modify the application's configuration file to use the GWC output driver instead of the GWC2 output driver.

In the following discussion, an application's configuration is updated, causing the application to be rendered using the legacy built-in rendering engine.

The original 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>
```

Notes:

1. The rendering and theme components are not explicitly defined in this application configuration. This application therefore inherits the rendering and theme components of the 'defaultgwc' parent application. (Line 02)

Examine the configuration for the parent application 'defaultgwc', and you see that both the rendering and theme components are set to use GWC2 output driver and snippet-based themes.

From the as.xcf (the GAS configuration file provided in GWC installation package):

```
01  ...
02  <!--This is the default application for GWC-->
03  <APPLICATION Id="defaultgwc" Parent="defaultwa" Abstract="TRUE">
04    <TIMEOUT Using="cpn.gwc.timeout.set1"/>
05    <PICTURE Using="cpn.gwc.picture"/>
06    <OUTPUT Rule="UseGWC">
07      <MAP Id="DUA_Symbol-WC" Allowed="TRUE">
08        <RENDERING Using="cpn.rendering.xslt"/>
09        <THEME Using="cpn.theme.default.gwc">
10          <TEMPLATE Id="_default">$(res.theme.symbol-
wc.stylesheet)</TEMPLATE>
11        </THEME>
12      </MAP>
13      <MAP Id="DUA_GWC" Allowed="TRUE">
14        <RENDERING Using="cpn.rendering.gwc"/>
15        <THEME Using="cpn.theme.default.gwc"/>
16      </MAP>
17      <MAP Id="DUA_AJAX" Allowed="TRUE">
18        <RENDERING Using="cpn.rendering.gwc2" />
19        <THEME Using="cpn.theme.ajax.gwc" />
20      </MAP>
21      <MAP Id="DUA_PAGE" Allowed="TRUE">
22        <RENDERING Using="cpn.rendering.gwc2" />
23        <THEME Using="cpn.theme.page.gwc" />
24      </MAP>
25      <MAP Id="DUA_PDA" Allowed="TRUE">
26        <RENDERING Using="cpn.rendering.gwc2" />
27        <THEME Using="cpn.theme.pda.gwc" />
28      </MAP>
29    </OUTPUT>
30  </APPLICATION>
31  ...
```

Notes:

1. This application is defined as an abstract application. This means its purpose or role is to provide a baseline of configuration settings to be inherited by other applications. (Line 03)
2. In the `MAP` element, the rendering and theme for the AJAX mode is explicitly specified for this abstract application. (Lines 17 - 20)
3. For the AJAX mode, the rendering component references the GWC2 rendering component. (Line 18)
4. For the AJAX mode, the theme specifies the snippet-based theme for AJAX mode. (Line 19)

Modify the application's configuration to explicitly specify the legacy built-in rendering engine, overriding the GWC2 and snippet-based theme settings inherited from the 'defaultgwc' application.

Modified 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    <OUTPUT Rule="UseGWC">
07      <MAP Id="DUA_AJAX" Allowed="TRUE">
08        <RENDERING Using="cpn.rendering.gwc"/>
09        <THEME Using="cpn.theme.default.gwc"/>
10      </MAP>
11   </OUTPUT>
12 </APPLICATION>
```

Notes:

1. An OUTPUT element is added. You need an `OUTPUT` element to contain a `MAP` element. (Lines 06 - 11)
2. In the `MAP` element, the rendering and theme for the AJAX mode is explicitly specified for this application. (Lines 07 - 10)
3. For the AJAX mode, the rendering component references the GWC rendering component (Line 08)
4. For the AJAX mode, the theme specifies the default theme, designed to work with the built-in rendering engine. (Line 09)

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.

Migrating to Genero Application Server 2.00

Topics

- fglxslp migration tool
- fglxmlp XML preprocessor

---

fglxslp

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
```

Notes:

- *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

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
```

Notes:

- *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
```