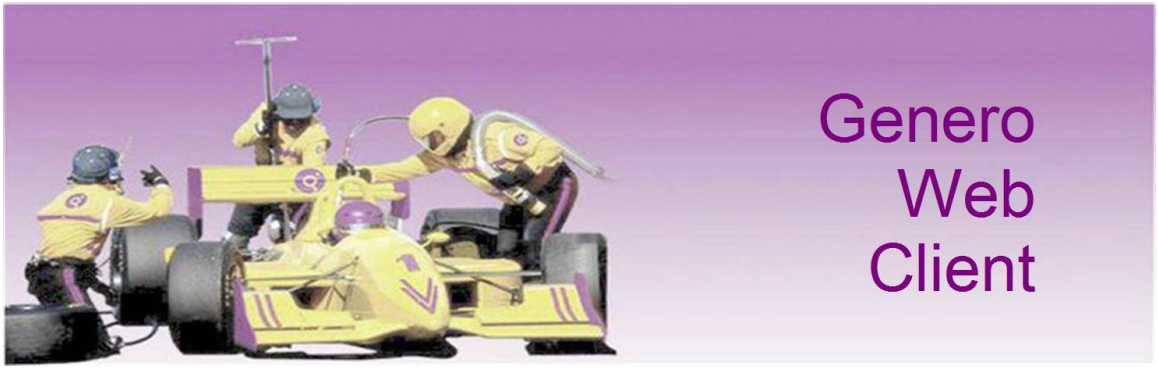


User Guide
Version 2.00

Note: This documentation is for Genero 2.00. 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.

Genero Web Client

Table Of Contents

GWC Overview & Architecture

INSTALLATION AND CONFIGURATION

Genero Web Client Overview

This section introduces you to the Genero Web Client (GWC).

Topics

What is the Genero Web Client?

The GWC is the Four J's product for creating Web applications with the BDL language.

Key Players

The key players involved in the development and deployment of a Web application using GWC include the BDL application development team, the Design team, the Advanced Production and Ergonomics team, and the Deployment and Infrastructure team.

Documentation Overview

This section provides you with a roadmap to the GWC documentation. Depending on your objectives, you can navigate to the section that interests you.

What is the Genero Web Client?

The Genero Web Client (GWC) is the Four J's product to create Web application with BDL language. It provides an application interface understandable by any browser using well-known web technologies like HTML, CSS and JavaScript.

It is flexible enough to let you build from a simple web application to a corporate web application. It brings to BDL applications the Internet world and the ability to be integrated in a Web site. GWC has its own application server to handle requests, uses a "template language" to create dynamic web pages and provides a default rendering that developers can customize.

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.

Key Players

The design, development, and roll-out of your GWC applications should involve persons with responsibilities in the following areas:

Area	Player Responsibility
BDL Application Development	Responsible for the development of the Genero application, concentrating on the business logic.
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.
Advanced Production and Ergonomics	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.
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.

It is rare that a single person fulfills the requirements demanded in each of these areas. Identifying a person or persons to be responsible for each area can alleviate issues when it comes time to test and deploy the application.

Documentation Overview

This document is organized into the following sections:

GWC Overview & Architecture

- The GWC Overview section provides an overview of the GWC product, a list of key players for the development and roll-out of GWC applications, and an overview of the GWC techincal documentation.
- The GWC Features section lists those Genero features supported by the GWC, as well as the Genero features NOT supported by the GWC.

- The Architecture section provides an overview of the architecture of the complete GWC solution, inclusive of required and optional third-party applications.
- The How the GWC Uses Web Technologies section discusses how GWC uses the Web technologies of HTML, CSS, XML, JavaScript, and the Four J's template language.
- The General Index section provides a quick-glance index for the documentation, sorted by keyword.

Installation and Configuration

- The Quick Start section provides a step-by-step overview for quick installation and surfacing of your Web application.
- The Installation section provides a detailed look at the installation of the GWC.
- The Configuration and Deployment section outlines the steps required for configuring the GWC and surfacing applications.
- The GWC Application Directory Structure section provides guidance on managing the files and directories required for customizing and surfacing applications in the GWC environment.
- The Configuring the Application Server for GWC section provides guidance for modifying the settings in the application configuration file (default **as.xcf**).
- The Troubleshooting Installation Issues section lists those issues that can cause problems during the installation process and offers solutions to work around those issues. It is assumed that you have already viewed the sections regarding Installation and Configuration and Deployment.
- The Internationalization section guides you in localizing applications to take advantage of non-ASCII character sets.

Customization of the Application Interface

- The Customization section identifies the methods available for you to customize your application.
- The Customization FAQ section answers those frequently-asked questions regarding customization of your GWC application.

Migration

This collection of topics provides information about adapting your Genero applications for the GWC and upgrading to the latest version of GWC.

- The Migrating Genero Applications to GWC section guides you in modifying your Genero applications to run in the GWC environment.
- The Migrating to GWC 1.32.1f, Migrating to GWC 1.30.1j, Migrating to GWC 1.30.1d, and Migrating to GWC 1.30.1c sections guide you in upgrading applications developed and tested with previous versions of GWC to the specified version.

Technical Reference

This collection of topics provides in-depth technical information regarding the components that work together to render an application using the GWC.

- The CSS Reference section list most of the available selectors for use in CSS when customizing the appearance of an application.
- The CSS Reference Sample section provides a graphical representation of the rendering of the selectors of input and display arrays, menus, and top menus.
- The Template Language Reference section provides detail about using the Four J's template language.
- The Template JavaScript API Reference section provides detail about using the GWC JavaScript API to put JavaScript functionality into a template.
- The Rendered HTML section details the relationship between 4GL code and the resulting rendered HTML.

Self-Paced Tutorials

- The Self-Paced Tutorials section is a series of self-paced tutorials walk you through the process of modifying an application through the use of templates, CSS, and JavaScript.

GWC Features

When working with the GWC, you are working within an html environment and are limited by html constraints. As a result, not all features supported by Genero are available with the GWC.

Topics

Genero Features Compatibility Status

This section identifies whether a Genero feature is supported by GWC. The features not currently supported that may possibly be supported in a future release are identified.

Unsupported Features

This section provides a concise list of those Genero features not supported by the GWC.

Genero Features Compatibility Status

The following table summarizes the status of Genero features for Genero Web Client. Refer to the Status Key for an explanation of the status.

Form Specifications			
Components	**Features**	**Status**	**Comments**
TOPMENU		OK	Accelerator keys are on hold. For more information on recognized accelerator keys, refer to the topic " Business Logic and GWC >> Accelerator Keys" (in the section Migrating Genero Applications to the GWC)
TOOLBAR		OK	Accelerator keys are on hold. For more information on recognized accelerator keys, refer to the topic " Business Logic and GWC >> Accelerator

			Keys" (in the section Migrating Genero Applications to the GWC)
LAYOUT		OK	Attribute on hold: SPACING
Containers			
Components	**Features**	**Status**	**Comments**
HBOX		OK	
VBOX		OK	
GROUP		OK	
FOLDER		OK	
PAGE		OK	
GRID		OK	
SCROLLGRID		OK	
TABLE		OK	Attributes on hold: UNHIDABLECOLUMNS, UNMOVABLECOLUMNS, UNSIZABLECOLUMNS, UNSORTABLECOLUMNS, WANTFIXEDPAGESIZE
	transposition	Not Planned	
	resize columns	On Hold	
Form Items			
Components	**Features**	**Status**	**Comments**
EDIT		OK	Attribute on hold: PROGRAM
BUTTON		OK	
BUTTONEDIT		OK	
CANVAS		On Hold	A solution is currently being examined based on svg.
COMBOBOX		OK	
CHECKBOX		OK	Three states
DATEEDIT		OK	
IMAGE		OK	Attributes on hold: AUTOSCALE, STRETCH
LABEL		OK	Attributes on hold: JUSTIFY
PROGRESSBAR		On Hold	
RADIOGROUP		OK	
TEXTEDIT		OK	Attributes on hold: STRETCH

RIP WIDGETS		On Hold	
Some attributes are on hold: ACCELERATOR, ACCELERATOR2, CENTURY, COLOR, COLOR WHERE, FORMAT, FONTPITCH, PICTURE, PROGRAM, SIZEPOLICY, SPACING			
Functional Features			
Components	**Features**	**Status**	**Comments**
Menu		OK	
Array		OK	
Construct		OK	All widgets are handled. New features (such as queryEditable) are available for COMBOBOX and RADIOGROUP.
MESSAGE		OK	
ERROR		OK	
Message files		On Hold	GWC does not support accelerator keys.
ON IDLE		On Hold	
MDI		Not Planned	
Start Menu		On Hold	
Statusbar		On Hold	
Styles		On Hold	At this time, presentation styles specified in a .4st file are not supported.

Status Key

Status	Definition
OK	Available and supported.
On Hold	Not yet implemented, but may be at a later time.
Not Planned	Will not be implemented.

Unsupported Features

Following is a concise list of unsupported features:

- Accelerator keys

- StartMenus
- MDI
- ProgressBar
- On idle
- Front Calls
- StatusBar
- Genero Styles (.4st)

For some unsupported features, alternate solutions have been identified. These alternate solutions are detailed in the topic "Business Logic and GWC" (in the section Migrating Genero Applications to GWC). You may also create your own interpretations.

Architecture

Users can connect to a Web application by connecting directly to the Genero Web Client, or they can access the application server via a Web server. The URI used to access the Web application determines which method is used.

Topics

Architecture overview

This section identifies the components required by the Genero Web Client solution, and describes how a request from a browser is served by the Genero Web Client engine. It identifies which components of the solution are provided by the installation of the Genero Web Client, and which components must be obtained and installed separately. It lists which files are needed by each component.

Connect directly to the Genero Application Server

This section discusses direct connections. With a direct connection, the browser connects directly to the Application Server without going through a Web Server. The connection is fully managed by the Application Server.

Connect to the Genero Application Server through a Web Server

This section discusses a Web server connection. With this type of connection, the browser connects to the Application Server through a Web Server. This type of connection is recommended for production environments.

Specifying the URI

This section lists the rules for specifying the URI are listed along with some examples.

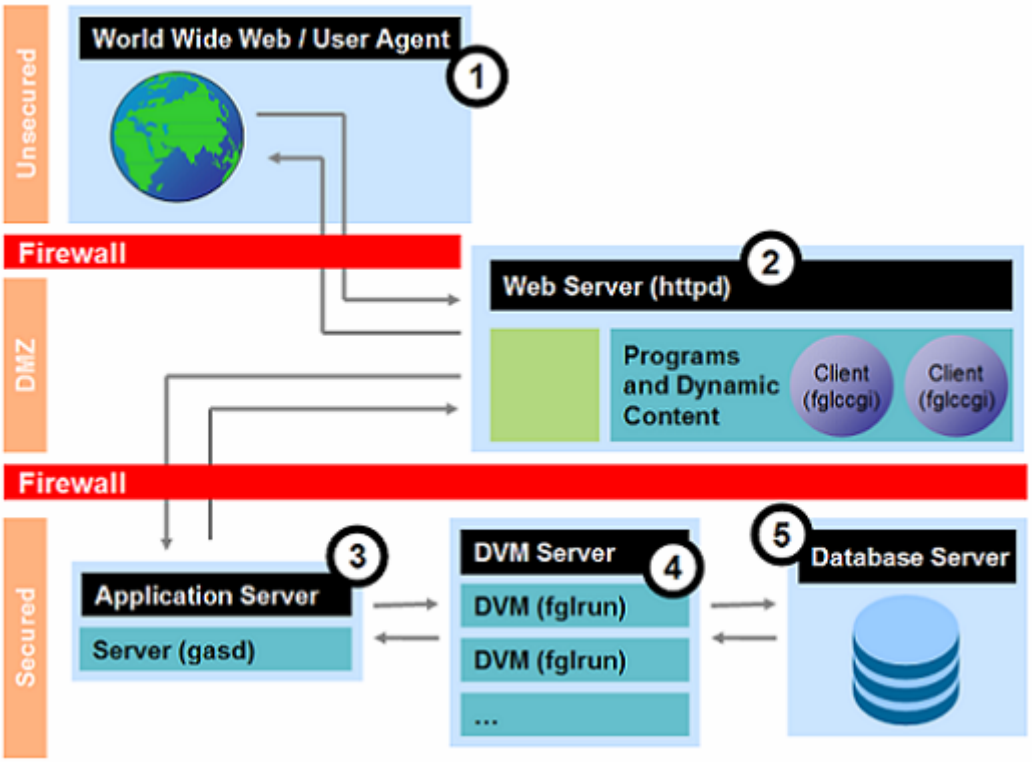
Architecture Overview

This section looks at the following topics:

- Components involved in the Genero Web Client solution, detailing how a request gets from the browser to an application.
- Third-party software requirements.
- Component relationships, or the relationship between the user agent, the Genero Web Client, the Dynamic Virtual Machine (DVM), and the files required by each.

Components involved in the Genero Web Client solution

The Genero Web Client works with a user agent, 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 in the Genero Web Client solution are shown in the diagram above.

- A *user agent* (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 Connector configuration is specified in the file connector.xcf.
- The Connector handles communication with the *Application Server*, also called the Genero Web Client daemon (3). The Genero Web Client 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 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.

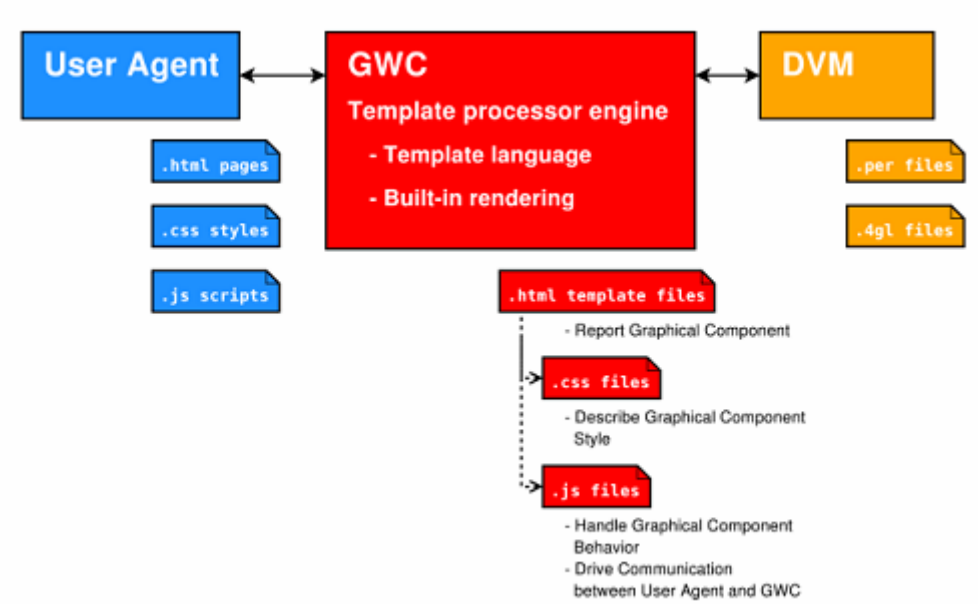
In development, it is typical to have the user agent (browser) connect directly to the application server, bypassing the Web server and Connector. For production, it is recommended that you include the Web server in your Genero Web Client solution.

Third-party software requirements

The Genero Web Client provides its own version of the Genero Application Server and includes the Genero Web Client Engine.

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 Genero Web Client, and a DVM. It also identifies which files are needed by each engine.

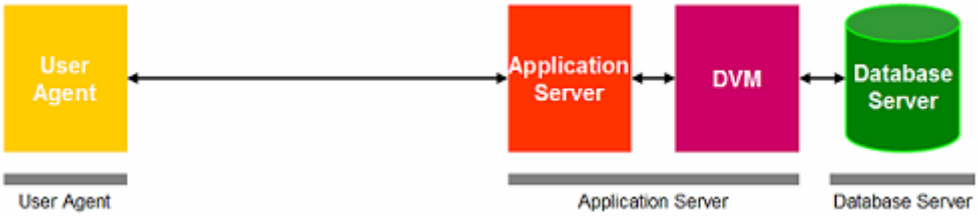
Connection Types

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

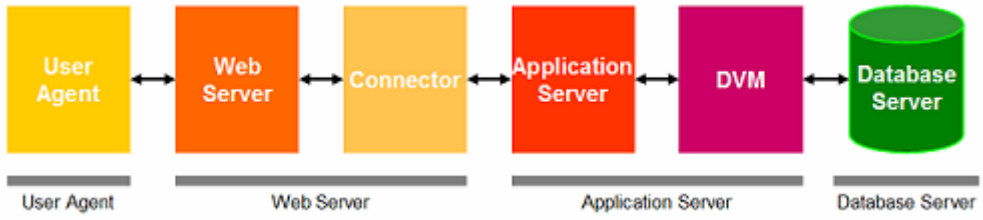
Direct connection allows User Agents (different web browsers) to connect directly to the Application Server, without using a Web Server. Direct connection is provided to simplify the architecture of development environments; it is not recommended for production environments.



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 Connector.

Connect to the application server through a Web server



When you connect through a Web Server, a 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).

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
        /directory [...]
        /connector-name
      ]
    |
    app-server[:app-server-port]
  }
  /scope
  /action
  /
    {
      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
action	STRING	Action requested of the Application Server.	r
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

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

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

Example 2

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

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

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

Example 4

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

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

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
```

How the Genero Web Client Uses Web Technologies

Genero Web Client allows developers to create HTML-based applications. Genero Web Client transforms (renders) a BDL application into a Web application.

BDL and HTML do not have the same widgets. For example, there are no native folder pages or calendars in HTML. Constraints in terms of communication are also not the same. The browser native mode is to submit information page by page, whereas Genero checks information field by field.

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

Template

A template is a HTML file that displays your application in a browser, using a Genero front end. The template defines how and where your application is displayed inside a HTML page.

Generated HTML

Genero Web Client converts BDL into pure HTML pages, following the current standards. The generated pages create working applications that do not need any CSS or JavaScript, however their appearance may be rough.

Cascading Style Sheets (CSS)

CSS are used to format HTML pages. Genero Web Client uses CSS to place and shape widgets as described in the Generated HTML description.

Javascript

The JavaScript code provided with Genero Web Client has two main goals: (1) to wrap widgets and handle

communication with the Genero Web Client and (2) to apply advanced styles to the rendered HTML

Template Language

The template language has been introduced to integrate web-designed pages and extend generated HTML capabilities. This language is used inside an HTML page and interpreted by the Genero Web Client engine, which generates new HTML code. You can perform instructions ranging from simple condition tests to loops on table lines.

Layout mechanism

The GWC renders the content of forms using a layout mechanism relying on JavaScript and CSS styles.

Template

A template is a HTML 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.

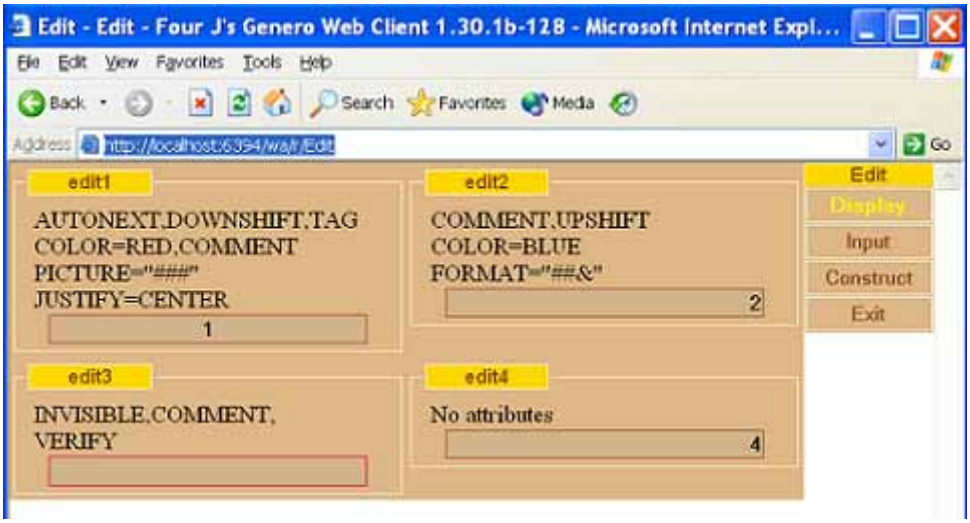
Excerpt from generodefault.html in $FGLASDIR/tpl:

```
01 <html>
02   <head>
03     $(res.meta-tags)
04     <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
05     <title gwc:content="string:${window/text} - ${application/text}
- Four J's Genero Web Client ${server/version}">Title of the
page</title>
06     <script language=javascript
src="$(connector.uri)/fjs/uaapi/webBrowser.js"></script>
07     ...
08   </head>
09
10   <body>
11     <form id="gDialogForm" method=post gwc:attributes="action
document/URL">
12       ...
13       <table width="100%" height="100%" border="0" cellpadding="0"
cellspacing="0">
14         ...
15         <tr height="100%">
```

```
16            <td width="100%" style="vertical-align:top;">
17              <table gwc:condition="form" id="gFormTable"><tr><td>
18              <div id="gForm-div" gwc:content="form" />
19              </td></tr></table>
20            </td>
21            <td valign="top" id="gPanel">
22              <div gwc:condition="menu" gwc:replace="menu" />
23              <div gwc:condition="dialog" gwc:replace="dialog" />
24            </td>
25          </tr>
26          ...
27        </table>
28      </form>
29
30      <script defer language=javascript
gwc:contentprotocol="string"><!--
31        var gLayoutData = ${document/layoutData};
32        gInitFieldMode( gIdToElement('gDialogForm'),
gSMART_FIELD_MODE, gINCREMENTAL_MODE, ${configuration/timeout/useragent
- 3} ); /// [ gFIELD_MODE | gSMART_FIELD_MODE ], [ gINCREMENTAL_MODE |
gFULL_MODE ], [keep-alive interval]
33      //--></script>
34    </body>
35 </html>
```

The application elements are rendered within cells of a HTML table in the page. For example, `<div id="gForm-div" gwc:content="form" />` displays the current form of the application. As a result of this code, Genero Web Client generates HTML code corresponding to the active form of the application. Similarly, `<div gwc:condition="menu" gwc:replace="menu" />` and `<div gwc:condition="dialog" gwc:replace="dialog" />` are instructions telling the Genero Web Client where to render the action panel.

Example (Edit application)

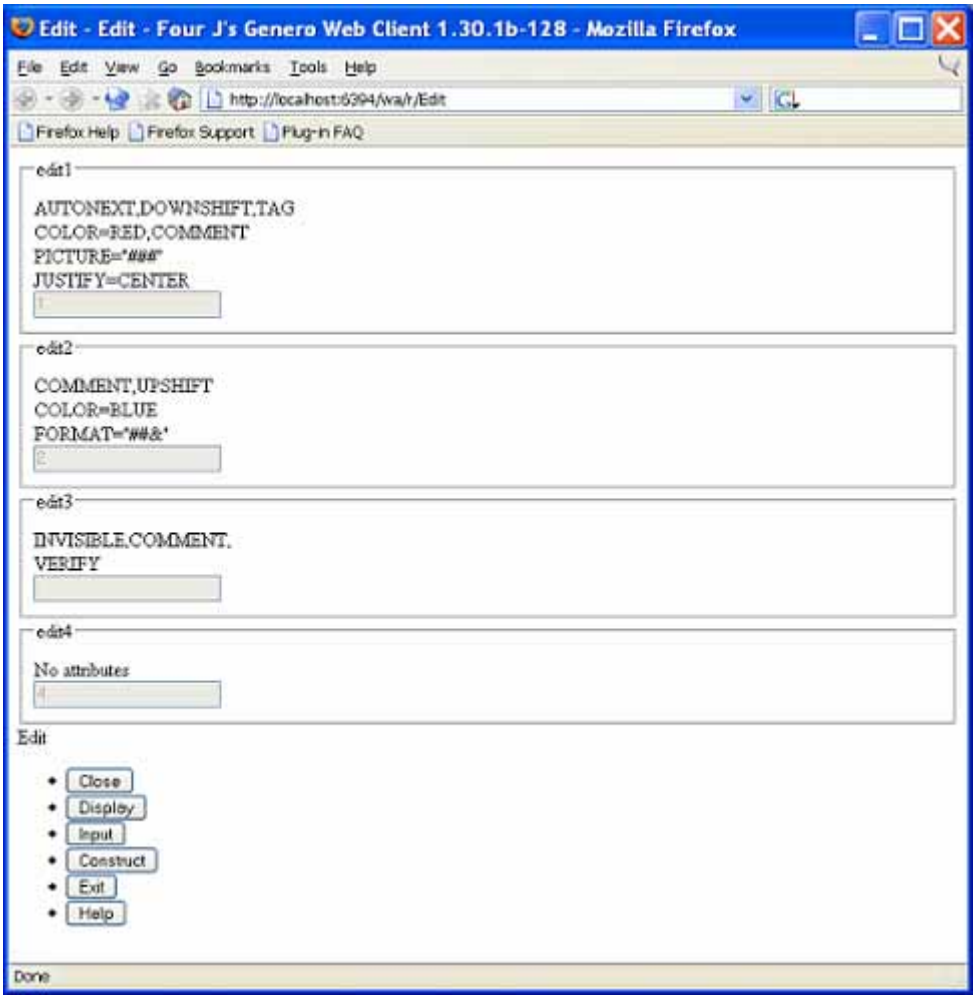


The screenshot shown above assumes you are using the default rendering of the Genero Web Client. Elsewhere in this manual, you can examine creating your own templates and using the template language to integrate or develop Web-designed pages.

Now that you see the container, the inside of this window is pure HTML.

Generated HTML

Genero Web Client converts BDL into pure HTML pages, following the current standards specified by the W3C. Generated pages provide working applications that do not require any CSS or JavaScript, however their appearance may be rough, as shown in the following screenshot:



Example of a generated menu (classical input HTML widgets):

```
01 <DIV class=gMenu>
```

```
02   <SPAN class=gMenuTitle>Edit</SPAN>
03   <UL>
04     <LI class="gMenuAction gHidden">
05       <INPUT class=gAction type=submit value=Close name=close>
06     <LI class=gMenuAction>
07       <INPUT class="gAction gCurrentAction" type=submit
value=Display name=display>
08     <LI class=gMenuAction>
09       <INPUT class=gAction type=submit value=Input name=input>
10     <LI class=gMenuAction>
11       <INPUT class=gAction type=submit value=Construct
name=construct>
12    <LI class=gMenuAction>
13       <INPUT class=gAction type=submit value=Exit name=exit>
14    <LI class="gMenuAction gHidden">
15       <INPUT class=gAction type=submit value=Help name=help></LI>
16   </UL>
17 </DIV>
```

The generated HTML structure is **flexible enough** to be easily customized.

Most HTML representations of a Genero widget have a container that can move to any place on the HTML page. In the default template, the menu is generated using the GWC instruction `gwc:replace="menu"` and placed in a table cell (see Edit screenshot in the Template paragraph). The menu can be moved anywhere simply by changing the location of the GWC instruction in the template. We can say that Genero Web Client provides ***placement capability***.

Most generated HTML objects have an identifier (name or id) and classes. Genero Web Client provides default CSS and JavaScript that handle and reshape pages. With our Edit example, the default rendering turns the menu buttons into flat ones and highlights the current action. Genero Web Client offers ***shape and behavior design***.

Developers can use this default rendering, or they can create appropriate CSS and JavaScript to control the look and behavior of widgets.

Cascading Style Sheet

CSS are used to format HTML 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 as described in the Generated HTML paragraph. The menu buttons are displayed flat, thanks to a CSS style:

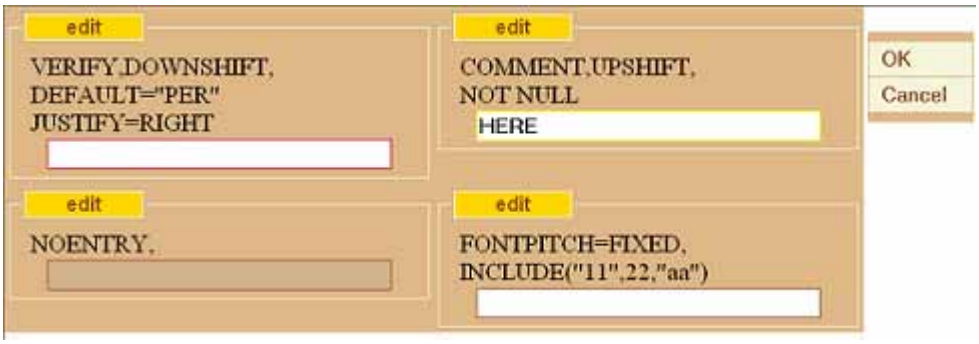
Excerpt from genero.css:

```
01 .gMenu INPUT {
```

```
02    ...
03   border: 0 none;
04    ...
05 }
```

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:



Generated HTML examples

```
<INPUT class="gEdit gNoEntry gDisabled" readOnly id=r3>
```

In this example, the `NOENTRY` state is represented by a class value gNoEntry

```
<INPUT class="gEdit gNotNull gUpshift" id=r2 value=here>
```

In this example, the `NOT NULL` and `UPSHIFT` states are represented by class values set to gNotNull and gUpshift

Developers can define their own styles for each widget state. Some complex renderings are set with JavaScript help.

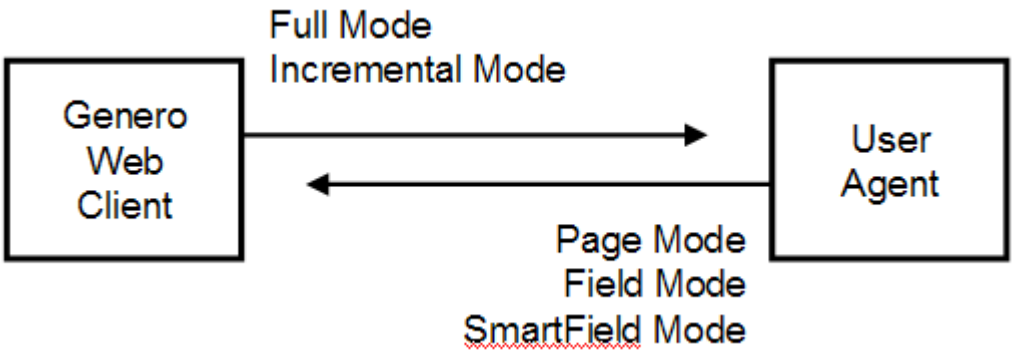
JavaScript

The JavaScript code provided with Genero Web Client has two main goals:

JavaScript, as provided with the Genero Web Client, can accomplish two goals.

- Provide wrappers for Genero widgets and handle communication between a field and the Genero Web Client.
- Apply advanced styles to the rendered HTML.

A wrapper initializes with a call to the **gInitFieldMode** function. This function takes two parameters: the first parameter specifies the communication mode from the Genero Web Client to the User Agent, while the second parameter specifies the communication mode from the User Agent to the Genero Web Client:



- **Full Mode**: The full page is sent each time
- **Incremental Mode**: Only changes are sent
- **Page Mode**: All the page is sent at once. (Thus interactive triggers are forbidden)
- **Field Mode**: Current field information is sent on all changes
- **Smart Field Mode**: Information concerning the changes of the fields values or states will be sent only if the client is not able to handle the current action. For example, this happens when a trigger is encountered or when the whole page has to be reloaded.

The following step listing describes how the JavaScript wrappers operate in Field Mode.

1. The wrapper detects a change in the field (receiving focus, mouse click, and so on), and tells the framework.
2. The framework asks the wrapper for information (value of the field, and so on) and sends the data to the Genero Web Client using the XMLHttpRequest object.
3. The Genero Web Client returns the response from the DVM to the framework using the XMLHttpRequest object.
4. The framework tells the wrappers about the changes they must perform.
5. The wrappers update the fields based on the new information.

A JavaScript API is provided to make the scripting of the Genero Web Client JavaScript functionalities easier. This API provides easy access to application events like changing a table offset, changing the value of a form field, or emulating a key press. In a future version of Genero Web Client, this API is expected to be enhanced to help the user create custom widgets. For additional information on the Genero Web Client and JavaScript API, see the *Template JavaScript API Reference* section of this manual.

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 inside an HTML page, 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 protocol.

Example

```
01 <title gwc:content="string:${window/text} -
${application/text}">Title of the page</title>
```

The conversion protocol is a string. Genero Web Client engine returns a string containing the window name and the application name.

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="string:${window/text} -
${application/text}">Title of the page</title>
```

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

For more information

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

Layout mechanism

With version 1.32, GWC introduces a new layout mechanism that is significantly more compatible with the GDC layout.

The principles are as follow:

- At a minimum, a widget has the size needed for its content.
- Containers are resized according to their content.
- Widgets keep the left and right alignment as defined in the form specification file (.per file).
- The layout is no longer "proportional" to the window's width.
- The space between widgets is minimized.

To achieve this new layout policy, the generated HTML has changed with regards to previous GWC versions. High-level containers (such as HBOX, VBOX, and other containers) become HTML tables, and the layout of the grids is finalized with some JavaScript managing the width of the widgets. The GWC uses a template path, `document/layoutData`, to generate the layout information needed by the JavaScript function. A GWC application can live without this layout information, but the layout will result in having all the form fields packed on the left side of the window and misaligned.

The benefits of this new layout policy are numerous:

- Alignment problems disappear.
- Data and/or labels are no longer clipped.
- The consequences of the "fixed" minimum sized layout are removed:

- Small fields are no longer too small.
- Large fields are not longer too wide.
- Space usage is globally optimized.

FAQ

Summary:

Startup questions

Do I need to be superuser to install GWC ?

No ...

But, some parts of GWC need to be installed with special rights.
For example, the web server part assumes that you have the rights to install the product in the web server directories.
You may also install GWC in the DVM directory. This is not recommended. If you intend to install GWC in $FGLDIR, then also check the rights for this directory.

What does "address already in use" mean ?

A gasd has already been started on the same port. Check in your AS configuration file ($FGLASDIR/as.xcf) the port where gasd started. Search for the section:
<INTERFACE_TO_CONNECTOR>
<TCP_BASE_PORT>6300</TCP_BASE_PORT>
<TCP_PORT_OFFSET>94</TCP_PORT_OFFSET>
By default gasd is started on port 6394. Set the values to a port which is not used by another application.

Why do I receive a 404/400 error when I use fglccgi.exe or fglcisapi.dll ?

By default, from IIS 6.x, running cgi or isapi is disabled. To use fglccgi.exe or fglcgisapi you need to enable their execution.
In the IIS manager console, go to the "Web Service Extension", select "CGI Extensions" and click on "Allow". Do the same thing with "ISAPI Extensions".

Why are some widgets partially rendered ?

The default theme (default rendering) is done by CSS and javascript. If you lack one of the two features, the rendering may be incorrect.

- Check that your browser meets the 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 **$(connector.uri)** in front of each reference to a file on the application server. For example, genero.css is on the application server, the reference to this file in the default template is:
  <link rel="stylesheet" href="**$(connector.uri)**/fjs/defaultTheme/genero.css" type="text/css"/>

Customization questions

How do I use a custom template ?

You need to configure your application to use the custom template. This template is a type of style that you apply to an application window.

See Declare a template and Apply a template in the Customize chapter for more details.

How do I use a custom cascading style sheet ?

This CSS has to be referenced in the application html template.

For more information see Preparing Customization and Customize with CSS in the Customize chapter.

Common errors

Bypassed triggers

In **page mode** if you have triggers in your input, you may encounter this error:

An error has occurred...
Error:

```
GWC: Edit - There is at least one bypassed trigger.
    Bypassed trigger owner field : r1(115)
    Trigger type: On Change
```

In **page mode**, triggers in input are not allowed, as there is no communication to the DVM when going from field to field.

Runtime error

This error appears when the application cannot start or cannot continue. This may due to a configuration error or an application error. Check in the AS log file.

An error has occurred...
Error:

```
Runtime error
```

Unsafe object

The object you tried to access does not belong to the AUI tree.

An error has occurred...
Error:
Access to an unsafe object (operator *)

Quick Start

When working with new software, one of the first goals of many administrators is to quickly install the application and verify the installation was successful. The Quick Start documentation identifies the process to follow to perform a rapid installation and the validation of that installation. After completing and validating the installation of Genero Web Client, additional configuration instructions are provided to assist you in delivering your applications via the Genero Web Client.

Topics

Quick Install

This section gives instructions for quickly installing the Genero Web Client application.

Launch Demos

This section describes the demo applications installed with the Genero Web Client and how to launch a specific demo applications from within a browser.

Run Application

This section describes how to configure and run a Genero Web Client application.

Quick Install

The Genero Web Client package installs two parts: the Application Server and the Web Server. The Application Server part installs, among other things, the Genero Web Client engine (gasd) and the demo programs. The Web Server installs the connectors (cgi or isapi).

Tip: you do not need to download and install the Genero Application Server package (GAS), as the Genero Web Client already include its own specific GAS.

For development, install the Application Server part. For production/deployment, you will need a Web Server.

Genero Web Client needs a Genero DVM. Verify that a DVM has already been installed.

To install the Genero Web Client:

- On Unix, issue the command: **/bin/sh fjs-gwc-1.32.xx-xxx.sh -i**
- On Windows, run the installer.

Next, follow the installation instructions. The installation directory is set to **$FGLASDIR**.

When the installation is done:

- Set Genero Web Client environment with "envas" script (located in **$FGLASDIR** directory)
- Launch the Genero Web Client engine with the command: **gasd**.

If you encounter any problems, check the sections on Installation or Troubleshooting Installation Issues.

Launch Demos

Genero Web Client is delivered with demo programs. The page **demos.html** lists the predefined applications. Use the following URL to display the list:

```
http://<app_server>:6394/demos.html
```

(Connecting directly to the application server.)

`http://<web_server>/cgi-bin/fglccgi/demos.html` (Connecting via a Web server.)

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 lunch application is a program to order your meals. It uses a database and some templates. You can refer to the readme in **$FGLASDIR/demo/lunch**.

Within demos.html:

The "Genero Web Client Widgets Demonstration" section lists some of the regular demo programs of the DVM. For example:

- The "DateEdit" demo shows you how a calendar is rendered.
- The "TopMenu" demo displays a drop-down menu that is handled entirely by CSS and JavaScript.
- The "Lists" demo displays a table that you can sort.

The "Genero Web Client template Demonstration" section lists applications illustrating the template language usage. For example:

- The "Repeat" demo displays a table with odd and even rows in different colors.

Run Application

This section describes how to configure a Genero Web Client application and run it.

An application is added either inside the as.xcf (located in $FGLASDIR/etc) or configured in an external file. If the application is defined within as.xcf, the application server dameon (gasd) needs to be restarted after any changes to the application definition. By default, application configuration files are located in $FGLASDIR/app.

Based on inheritance concepts, an application is configured with a few lines.

Example

```
<APPLICATION Id="myapp" Parent="defaultgwc">
  <EXECUTION>
    <PATH>/home/myapp/bin</PATH>
    <MODULE>app.42r</MODULE>
  </EXECUTION>
</APPLICATION>
```

If you use the predefined Genero Web Client environment, the parent of the application is set to "defaultgwc". Then you only need to specify the path to your application.

The application name is set with "Id". In this example, the application is named "myapp". You need to specify where to find the compiled modules; set the path in <EXECUTION><PATH> section. If the "Id" of the application does not have the same name as your main module then add a <MODULE> tag. Here, the application name is "myapp" and the main module is app.42r.

To run the application, use, for example, the URL http://<app_server>:6394/wa/r/myapp.

Application using database

For applications using a database, you need to set the environment variables to access the database. Environment variables are defined in the <EXECUTION> section of an application.

Example (using Informix)

```
<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>
  ...
</EXECUTION>
```

Applications with images

The default images are in $FGLASDIR/pic. To define your own directory of pictures, you need to declare an alias in the as.xcf and a reference to this alias from application configuration.

Alias example

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

An alias is like any Web Server alias. This maps a URL path to the server directory. In this example, the alias /images is mapped to the directory /home/app/images. Then, for example, you can access an image with the URL http://<app_server>:6394/images/img.png, assuming img.png is in directory /home/app/images.
After that, you need to reference this alias in the application configuration.

Picture example

```
<APPLICATION ...>
  <EXECUTION> ... <EXECUTION>
  <PICTURE>
    <PATH>$(connector.uri)/images</PATH>
  </PICTURE>
</APPLICATION>
```

With the <PICTURE> tag, you can specify a path, which is the alias. $(connector.uri) is added to make the pictures stored on the application server available from the Web server.

For more details on configuration you can refer to the Configuration and Deployment section.

Application with arguments

To send arguments to your Main function the keyword Arg is used in the URL.

Example

```
http://server/cgi-bin/fglccgi/wa/r/myapp?Arg=value1&Arg=value2
```

value1 is the value of the first argument and value2 is the value of the second argument. Each argument is separated by an ampersand.

Installation

The installation process involves validating that your system meets the base system requirements for the Genero Web Client, selecting the type of installation you wish to perform.

Topics

System requirements

This section lists the software supported by the Genero Web Client solution.

Determining the installation type

This section provides information regarding the three installation types offered by the Genero Web Client during installation, allowing you to select the correct type of installation for your environment.

Installation procedures

This section documents the installation procedure.

Directories and files

This section lists those directories and files created by the installation process.

Validate your installation

This section provides instructions for testing your installation, allowing you to verify that your installation was successful.

System Requirements

This section lists the software and software versions for operating systems, DVMs, Web servers and user agents supported by the Genero Web Client solution.

Operating Systems

The following table identifies the supported operating systems.

Identifier	OS Name	Platform	OS Version
aix0510	IBM AIX 5.1 64 bits	RS/6000	5.1
aix0510	IBM AIX 5.2 64 bits	RS/6000	5.2
aix0510	IBM AIX 5.3 64 bits	RS/6000	5.3
h64i112	Hewlett Packard Itanium HP-UX 11.23 64 bits	Itanium 2	11.23
hpx1100	Hewlett Packard HP-UX 11 32 bits	PA RISC	11
l64il23	Ligne RedHat Enterprise Edition 3 Update 4 (64bits for Itanium 2 (glibc2.3)	Itanium 2	Linux distribution with glibc 2.3
l64pl23	Ligne RedHat Enterprise Edition 4 Update 1 for processor IBM power5 64bits (glibc2.3)	ppc64	Linux distribution with glibc 2.3
l64xl23	Ligne RedHat Enterprise Edition 3.0 Update 5 for processor x86 64bits (glibc2.3)	x86_64	Linux distribution with glibc 2.3
lnxlc23	Linux Red Hat Entreprise Edition 3.0 Update 1 (glibc 2.3)	Intel	Linux distribution with glibc 2.3
osf0510	Compaq Tru64 UNIX 5.1B	Alpha	5.1B
s640800	SUN Microsystems Solaris 8 64 bits Sparc	SPARC	
s640800	SUN Microsystems Solaris 9 64 bits Sparc	SPARC	
sco0505	SCO Open Server 5.0.5 and 5.0.7	Intel	Linux distribution with glibc 2.3
windows	Microsoft Windows	Intel	NT4/SP6, 2000/SP4

Dynamic Virtual Machine (DVM)

At a minimum, DVM 2.00+ is required. While the Genero Web Client may appear to work with previous versions of the DVM, it is unsupported.

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

The following table identifies the supported user agents.

User Agent	Version(s) Supported
Microsoft	• Internet Explorer (IE) 5.5 (**Note:** IE 5.5 is no longer supported by Microsoft.) • Internet Explorer (IE) 6.x
Mozilla-based Browsers	The browsers that derive from the Mozilla 1.7 trunk: • Mozilla 1.7+ (Mozilla 1.6 is ok but has a few layout problems on some Genero Web Client pages) • FireFox 0.9+ • Netscape 7.2+ • Camino 0.8+ **Note:** Mozilla 1.7 is available on the latest Mandrake and Red Hat Enterprise Linux; other Linux still come with Mozilla 1.6 (which is OK apart from little layout problems). New versions are announced with Mozilla 1.7+ and FireFox 0.9+.

Note: You will likely notice a difference in performance between applications delivered through Internet Explorer and those delivered through FireFox, as FireFox has faster JavaScript and CSS engines. Future releases of Genero Web Client will include optimizations for Internet Explorer, especially for folder containers.

Determining the Installation Type

Since Genero Web Client includes its own specific Genero Application Server (GAS), you are presented with a list of three types of installations. These types are:

- **Installation Type 1**: Install the Genero Web Client Components

  Choose this type to install the Genero Application Server required by the Genero Web Client. Installation Type 1 is all that is needed for development purposes.

- **Installation Type 2**: Install the CGI Connector

  Choose this type to install the CGI Connector on the machine that hosts your Web server.

- **Installation Type 3**: Install both the Genero Web Client Components and the Connector

  Choose this type if your application server and web server sit on the same host (machine).

Running the installation program

You may have experienced Genero Application Server (GAS) with other products like Genero Desktop Client ActiveX, Genero Java Client, or Genero Web Services Extension. However, Genero Web Client is delivered with its specific Genero Application Server, and includes its own rendering engine. As a result, the use of Genero Desktop Client ActiveX, Genero Java Client, or Genero Web Services Extension is not possible when using the Genero Application Server installed with the Genero Web Client. If you want to use any of these products and the Genero Web Client simultaneously, you must install the Genero Application Server package as well as the Genero Web Client package, and you must ensure the two installation do not overlap (by using different directories, different port configurations, and so on).

The Genero Web Client uses the same executable name as the Genero Application Server for its daemon (**gasd** on Unix systems, **gasd.exe** on Windows systems) and the same file name for its default configuration file (**as.xcf**). In addition, the installation directory for the Genero Web Client is also stored in the FGLASDIR environment variable, the same environment variable that stores the installation directory for the Genero Application Server (FGLASDIR).

The installation procedure differs between UNIX and Windows platforms:

- Installing on UNIX
- Installing on Windows

Installing on UNIX platforms

Warning: Four J's prohibits the installation of the Genero Web Client by user "root". You should create a specific user to own the files installed by the Genero Web Client and to start the Genero Application Server. Once you have created the user, log in as that user and run the installation program.

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-gwc-1.32.xx-lnxlc22.sh -h
```

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

```
$ /bin/sh fjs-gwc-1.32.xx-lnxlc22.sh -i
```

The installation program identifies the operating system and checks that all the system requirements are met. If all system requirements are met, the installation program prompts you to select an installation type, as described in Determining the Installation Type:

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

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.

Installing on Microsoft Windows platforms

For the Microsoft Windows platform, Genero Web Client is provided as a standard Windows setup program. Distribution files and the installation program are provided in the same file.

```
fjs-gwc-1.32.xx-windows.exe
```

As in Genero Web Client UNIX packages, you must select your installation type as described in the section Determining the Installation Type.

Note: With Microsoft Internet Information Services (IIS), the installed files may not have the right permissions. You need to update these file permissions to match IIS permissions.

Directories and Files

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	Connector to Genero Web Client.

	fglcgienv	Tool to check Web Server environment.
	connector.xcf	Connector configuration file.
$FGLASDIR		Genero Web Client installation directory.
$FGLASDIR/bin	gasd	Genero Web Client daemon.
$FGLASDIR/etc	as.xcf	Genero Web Client configuration file.
$FGLASDIR/tpl/	generodefault.html	Genero Web Client default template.
$FGLASDIR/web/fjs/	demos.html	Demonstrations listing.
$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.
	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).

For more information on the directories and files required to support the deployment of a Genero Web application, refer to the section Genero Web Client Application Directory Structure.

Validate your installation

Application Server

1. Genero Web Client requires its daemon to be started. Set the Genero Web Client environment using the script `$FGLASDIR/envas` and start the Genero Web Client daemon with the `gasd` command.
2. 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.

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
   ```

Configuration and Deployment

Topics

Configuring Genero Web Client

This section discusses how a Genero Web Client administrator configures the Genero Web Client application server and the Genero Web Client connector.

Deploying Applications

This section explains how to deploy a Genero Web application as either an internal application or an external application.

License Usage

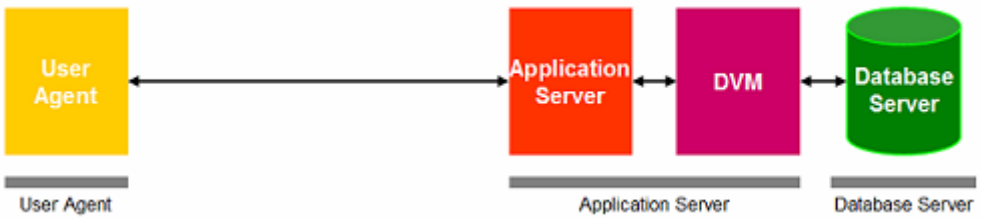
This section explains how licenses are consumed when applications are started by the Genero Web Client.

Configuring Genero Web Client

To access a Web application, a user specifies either a direct connection or a connection through a Web server.

Direct Connection

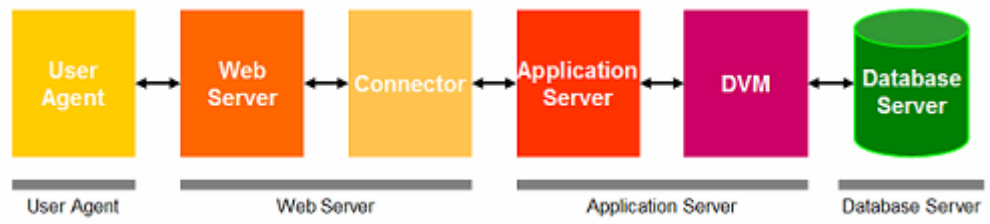
A *direct connection* is when an application is accessed by connecting directly to the application server installed with the Genero Web Client. A direct connection, when permitted, can be requested from any user agent (Web browser); it is not browser dependent. A direct connection is fully managed by the application server.



A direct connection is typically used in most development environments. Although this method of connecting to an application is possible in a production environment, it is not recommended.

Connection through a Web Server

The common connection method in a production environment is to connect to the application server through a Web server. Under this configuration, the Web server manages the connection between the User Agent and the application server.



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`

Configuring the Genero Web Client

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

For each Genero Web Client application server 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. A full explanation of the application server configuration file is available in the Reference >> AS and Connector configuration section of this manual.

The Genero Web Client 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:

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

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

To create an 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, 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 a Connector (for a Web server)

For each Web server you introduce into your solution, you must install and configure a Genero Web Client Connector. An explanation of the Connector configuration file can be found in the Reference section of this manual, under the heading Configuring the Application Server for GWC.

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 **connector.xcf** file match a base port and port offset set in an application server configuration file.

Deploying Applications

To deploy an application, it must be defined. A Genero Web Client application can be defined as an internal application or as an external application.

- Creating an application deployment strategy
- Defining an internal application
- Defining an abstract application
- Define an external application or application group

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 Genero Web Client 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 (connector.xcf), refer to the section Configuring the Applicaiton Server for GWC.

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 application server 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 Genero Web Client.

Defining an internal application

An internal application is defined within the application server configuration file. By default, the application configuration is defined in the **as.xcf** file, although an alternate application server configuration file can be specified when starting the application server.

Example in as.xcf with '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>
228
228 <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, refer to the section Configuring the Application Server for GWC.
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.

Defining an abstract application

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 exactly the same way an internal or external application is, except it has an extra attribute set in its `APPLICATION` tag: `Abstract="TRUE"`

Defining an external application or application group

Putting all the application in the **as.xcf** file is not scalable. Therefore, Genero Web Client supports external applications. An external applications has its configuration defined in an application-specific XML file with the extension **.xcf**.

Example

The following XML defines the Edit application in an external application configuration file *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 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 Genero Web Client 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 Genero Web Client after modifying an external configuration file.
3. The directory where Genero Web Client searches for the external application configuration file is defined in **as.xcf** by the tag `<GROUP Id="_default">/some/where/</GROUP>`. The default after installation is `$FGLASDIR/app`.

Limitations

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

Application Group

Putting all the external application configuration files in just one directory is not scalable. Therefore, Genero Web Client allows you to define groups.

Example

The following XML, from the as.xcf file, specifies the *tpl-demo* group.

```
01 <?xml version="1.0" encoding="UTf-8"?>
02 <?fjsApplicationServerConfiguration Version="1.30"?>
03 <CONFIGURATION>
...
181 <APPLICATION_LIST>
...
206  <!-- template demos group -->
207  <GROUP Id="tpl-demo">$(res.path.as.demo)/template/app</GROUP>
...
228 <APPLICATION_LIST>
...
235 <CONFIGURATION>
```

In this above XML, a group named `tpl-demo` is defined. A directory is specified for the group. When a request is made on the URL:

```
http://host:6394/wa/r/tpl-demo/tpl-replace
```

a file named `tpl-replace.xcf` is sought in the `$(res.path.as.demo)/template/app` directory. If this file is found, then Genero Web Client loads it, and launches the corresponding application.

Notes

1. Group applications `Id` are `name-of-group/name-of-application`. The `Id` attribute of `<APPLICATION>` tag is not used. In the example above, the `Id` of the application is `tpl-demo/tpl-replace`.
2. When there is no defining `<MODULE>` tag in the application configuration, the module taken by default is just `name-of-application`.

Related Topics

- Configuring the Application Server for GWC
- Step 0: Using the Built-In Rendering

The Genero Web Client and License Usage

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 browser. If within this application, a RUN or a RUN WITHOUT WAITING is executed, the same license is used, even if the first browser opens new browsers. If, however, an application is started in another browser (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 the user does not exit the application but instead closes the browser, the DVM continues to run until the application times out (the number of seconds set for the USER_AGENT timeout passes). At that time, 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`".

Genero Web Client Application Directory Structure

When developing applications for the Genero Web Client, it is typical to do all development on the application server and to move specific files to the Web server during deployment for your production environment.

Recommended Directory Structure for Development

This section discusses the recommended directory structure for the application development phase.

Where to Place Files for Production

This section discusses the available options for the location of key files in a production environment.

For a listing of those files created by the installation of Genero Web Client, refer to the section "Directories and Files" (located within the Installation topic).

Recommended Directory Structure for Development

For any Genero application, you have to manage .4gl, .per, and other files associated with the DVM (.4ad, .4st, and so on). In addition to these files, Genero Web Client introduces html templates, CSS and javascript. For these Genero Web Client-related files, it is suggested that you organize the directory to represent a small local web site.

Example

- myApp: root directory of your application
  - src: 4GL programs, per files, and so on
  - web: root directory of your web site
    - img: web site pictures
    - inc: javascript and CSS files
    - html: html pages (templates)

This is a simple directory structure for organizing files relating to 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.

For development purposes, the local web site helps you visualize the templates in the almost final environment. To reference your local site in the configuration file as.xcf, set an alias.

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

Development and testing of your Web applications will likely occur using a direct connection to the application server. When it is time to deploy your application and connect via a Web server, you may want to move some of the files sitting on your application server to the Web server. In general, Four J's recommends you keep all files on the application server and move them to the Web server only if performance issues arise and you have no choice but to move files to the Web server.

Topics

- Which files can exist on the application server, Web server, or both?
- Keeping files on the application server.
- Moving files to the Web server.
- Maintaining files on both the application server and Web server.

Files involved

For production, you must decide where to place the following files:

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

Note: The template files (html) must be located on the application server.

Leaving the 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, `$(connector.uri)` must be referenced. Remember, `$(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. Therefore, when going through a Web server, `$(connector.uri)` is required to retrieve documents sitting on the application server.

Moving Files onto the Web Server

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

- Ensure you have defined the same aliases on the Web server as you have declared in the application server configuration file (**as.xcf**). Refer to the `<ALIAS>` section in your application configuration file.
- Move the CSS, JavaScript, and documents to the right place on the Web server (in the directories specified by the aliases) .
- Move template images. Template images are those used by the html page, 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, gasd 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 `$(connector.uri)`, the Web server will not be able to access files that sit on the application server files.
- You may decide to leave a subset of the files on the application server and not place them on the Web server. For example, you may decide to leave **genero.css** on the application server, and only copy your custom CSS files to the Web server. In this situation, you would leave `$(connector.uri)` in the path to **genero.css** in your template.

Configuring the Application Server for GWC

GWC configuration is handled by two files: the Genero Application Server configuration file (default **as.xcf**) to configure the application server and the GAS Connector configuration file (**connector.xcf**) to configure the GAS Connector.

In this section, concepts for completing the configuration of the Genero Application Server configuration file for use with the GWC is presented. This section is not intended to be a complete reference for the configuration of this file; it is intended to discuss those features you must understand to configure this file for the GWC.

Topics

This section covers the configuration of the application server for those areas specific to the Genero Web Client. For other configuration guidance, refer to the "Genero Application Server Manual", Section "Configuration Reference", Chapter "Application Server".

- Applications
- Application Groups
- Application Definitions

Complete configuration reference documentation

- The reference documentation for the Genero Application Server configuration file (**as.xcf**) can be found in "Genero Application Server Manual", Section "Configuration Reference", Chapter "Application Server".
- The reference documentation for the GWC Connector configuration file (**connector.xcf)** can be found in "Genero Application Server Manual", Section "Configuration Reference", Chapter "Connector".

Applications

The last section of the as.xcf file is the applications list. It can contain application groups, abstract applications, and application definition.

- Application group
- Application definition

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

Application group

Application groups are used to give a hierarchy to your applications. It actually defines an alias used in the application URL which points to the physical directory holding the external applications configuration files.

```
<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>
```

The id **_default** is a key word that defines the default group.
For example to access the Edit application the URL is:
http://server/cgi-bin/fglccgi/wa/r/Edit

```
02 <GROUP Id="tut-demo">$(res.path.as.demo)/tutorial/app</GROUP>
```

This group gather applications related to the tutorial.
For example, to access to the tutorial first step use the URL:
http://server/cgi-bin/fglccgi/wa/r/**tut-demo**/tutorialStep1

Application definition

The `APPLICATION` tag defines an application environment. In this tag, 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`

An abstract application is used to share common configuration between multiple child applications. An abstract application can't be instantiated.

```
<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. *resource* is a local `RESOURCE` definition
4. *exCompId*, *timeCompId*, *picCompId* and *themeCompId* are components identifiers
5. the content of *execution*, *timeout*, *picture* and *theme* is the same as the content of their respective components

Example (excerpt from tutorialStep5.xcf)

```
01 <APPLICATION Parent="demo-tut-abstract">
02   <RESOURCE Id="res.template.tutorial.addcard"
Source="FILE">$(res.path.demo.dem-
tut)/web/tutorial/tutorialStep3.html</RESOURCE>
03   <EXECUTION>
04     <PATH>$(res.path.demo.dem-tut)/src</PATH>
05     <MODULE>card.42r</MODULE>
06   </EXECUTION>
07   <OUTPUT>
08     <MAP Id="DUA_GWC">
09       <THEME>
10         <TEMPLATE
Id="addcard">$(res.template.tutorial.addcard)</TEMPLATE>
11       </THEME>
12     </MAP>
13   </OUTPUT>
14 </APPLICATION>
```

`Parent="demo-tut-abstract"` defines that the application inherit demo-tut-abstract application definition.
The execution section is redefined to use a new path to the program and a new main module.
According to the output section, if an application window has "addcard" style, the template for the window rendering is $(res.template.tutorial.addcard) .

Troubleshooting Installation Issues

In this section, troubleshooting issues are presented as Frequently Asked Questions.

- Startup questions
  - Do I need to be superuser to install Genero Web Client?
  - What does "address already in use" mean?
  - Why do I receive a 404/400 error when I use fglccgi.exe or fglcisapi.dll?
  - Why are some widgets partially rendered?
- Common errors
  - Bypassed triggers
  - Runtime error
  - Unsafe object
- How do I use the debugger with a Genero Web Client application?

Startup questions

Do I need to be superuser to install Genero Web Client ?

You do not need to be superuser to install Genero Web Client, however some parts of Genero Web Client 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.

Also, although this is NOT recommended, you may want to install Genero Web Client in the DVM directory. Again, this is not recommended. If, however, you intend to install Genero Web Client in **$FGLDIR**, then check the rights for this directory.

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.

Why do I receive a 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".

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 `$(connector.uri)` in front of each reference to a file on the application server. For example, genero.css is on the application server, the reference to this file in the default template is:

```
<link rel="stylesheet"
href="$(connector.uri)/fjs/defaultTheme/genero.css"
type="text/css"/>
```

Common errors

Bypassed triggers

In **page mode** if you have triggers in your input, you may encounter this error:

An error has occurred...
Error:

```
GWC: Edit - There is at least one bypassed trigger.
    Bypassed trigger owner field : r1(115)
    Trigger type: On Change
```

In **page mode**, triggers in input are not allowed, as there is no communication to the DVM when going from field to field.

Runtime error

This error appears when the application cannot start or cannot continue. This may due to a configuration error or an application error. Check in the AS log file.

An error has occurred...
Error:

```
Runtime error
```

Unsafe object

The object you tried to access does not belong to the AUI tree.

An error has occurred...
Error:
Access to an unsafe object (operator *)

Using the Debugger

This section provides instructions for using the debugger with Genero Web Client.

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

How to set up the debugger for Genero Web Client 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)

Caution: 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.

How to Set Up the Debugger for Genero Web Client 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.

Internationalization

This section explains how the Genero Web Client handles international applications.

Topics

Encoding Architecture

This section describes the GWC encoding architecture.

Charsets Configuration

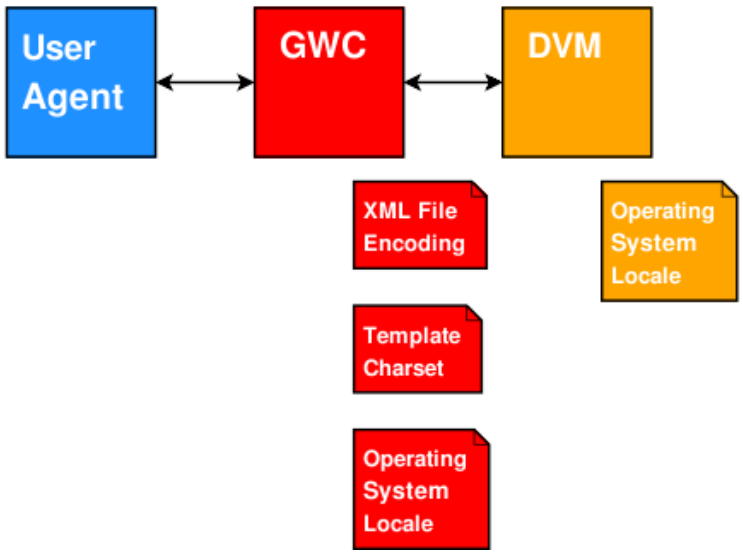
This section identifies the four places where character sets (charsets) can be defined.

Supported Charsets

This section lists all character sets (charsets) known by the GWC.

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 GWC 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 GWC.
4. With environment locales when launching GWC.

DVM Locale

If application files (*i.e.* .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 "BDL Reference Manual", section "Programming Applications", chapter "Localization". Locales can be set in the GWC 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 GWC 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

GWC 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>
...
```

GWC System Encoding

GWC interacts with Operating Systems in many ways:

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

In these cases GWC 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 Gwc 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 Gwc starts with 'UTF-8' system encoding
```

Note: Encodings have different names across Operating Systems. To unify them, GWC 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 GWC uses `UTF-8` encoding for handling all Unicode characters.

Supported Charsets

The following list contains all character sets known by the GWC. One coded character set can be listed with several different names. Depending on your Operating System, DVM may support these character sets. Read "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
```

Customizing Web Applications

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.

The default GWC rendering is used for all applications, however individual applications can be customized. There are three main ways to customize this rendering: CSS, JavaScript and Template Language. Each application can configure its own templates and reference specific CSS and JavaScript.

Topics

Preparing for Customization

This section covers organizing your directory structure for application customization, creating a template, declaring one or more templates for your application, and applying a template.

Customize with CSS

This section covers customizing an application's look-and-feel using CSS, declaring a style, overriding existing styles, and adding user styles to specific widgets.

Customize with Templates Language

This section discusses the use of template language to merge the business logic seamlessly into the HTML pages that display the application.

Customize with JavaScript

This section discusses the use of JavaScript to ease the interaction with users and to refine the application's design.

Preparing for Customization

To customize your application, create your own CSS, JavaScript, and template files. We advise you to leave the CSS, JavaScript and templates delivered by the installation of GWC unchanged.

Steps

- Organize your directory structure for the files that enable customization of the application.
- Create a template for the application to use.
- Declare the template as available for use by the application.
- Apply the template to a window or form.

Organize Your Files

Files required for the deployment of a GWC application should be organized as outlined in the section GWC Application Directory Structure.

Create a Template

To create a custom template, either create an HTML template from scratch or create a copy of $FGLSADIR/tpl/generodefault.html and rename the copy. The custom template should be created in the appropriate directory (as specified in the section GWC Application Directory Structure).

In the custom template, add or replace the links to your own CSS and JavaScript files. The links are typically found in the `<HEAD>` section of the HTML document, although the `<SCRIPT>` tag for JavaScript files may also be found in the `<BODY>` section.

Example for CSS:

```
01 <link rel="stylesheet" href="$(connector.uri)/mysite/inc/myapp.css"
type="text/css"/>
```

(assuming your CSS file is named myapp.css and stored in /myApp/web/inc directory, and assuming that an alias "/mysite" has been created to reference "/myApp/web", as shown in the example in the section GWC Application Directory Structure.)

$(connector.uri) is added for the deployment with a Web server.

Example for JavaScript:

```
01 <script language=javascript
src="$(connector.uri)/mysite/inc/myapp.js"></script>
```

(assuming your JavaScript file is named myapp.js and stored in /myapp/web/inc directory, and assuming that an alias "/mysite" has been created to reference "/myApp/web", as shown in the example in the section GWC Application Directory Structure.)

Declare a Template

Once you have created a template, you must declare the template for your application. You can have several templates for one application. Each template defines a style for the application window, so a template is related to a window style.

Example:

```
01 <APPLICATION Id="demotpl" Parent="defaultgwc">
02   <RESOURCE Id="res.template.mytpl"
Source="FILE">/myapp/web/html/tpl.html</RESOURCE>
03   <RESOURCE Id="res.template.mytpl2"
Source="FILE">/myapp/web/html/tpl2.html</RESOURCE>
04   <OUTPUT>
05     <MAP Id="DUA_GWC">
06       <THEME>
07         <TEMPLATE Id="style1">$(res.template.mytpl)</TEMPLATE>
08         <TEMPLATE Id="style2">$(res.template.mytpl2)</TEMPLATE>
09       </THEME>
10    </MAP>
11  </OUTPUT>
12 </APPLICATION>
```

where **res.template.mytpl** is a file resource, as in line 02:

```
<RESOURCE Id="res.template.mytpl"
Source="FILE">/myapp/web/html/tpl.html</RESOURCE>
```

Apply a Template

A template can be applied to a window or a form. There are two ways to use a template.

Template applied in .per file

```
01 LAYOUT(TEXT="Example", STYLE="style1")
```

The `STYLE` attribute of the `LAYOUT` section specifies the template to use for this window. Remember that "style1" references the tpl.html template.

Template applied in .4gl file

```
01 OPEN WINDOW win WITH FORM "aform" ATTRIBUTES(STYLE="style1")
```

The `STYLE` attribute for `WINDOW` specifies the template to use.

Customize with CSS

CSS is the fastest and the simplest way to change your application's look. We use CSS 2 standards described on the W3C site. Most generated HTML elements have classes and containers (`<DIV>`) so you can change the widget's look and place the elements where you want.

Topics

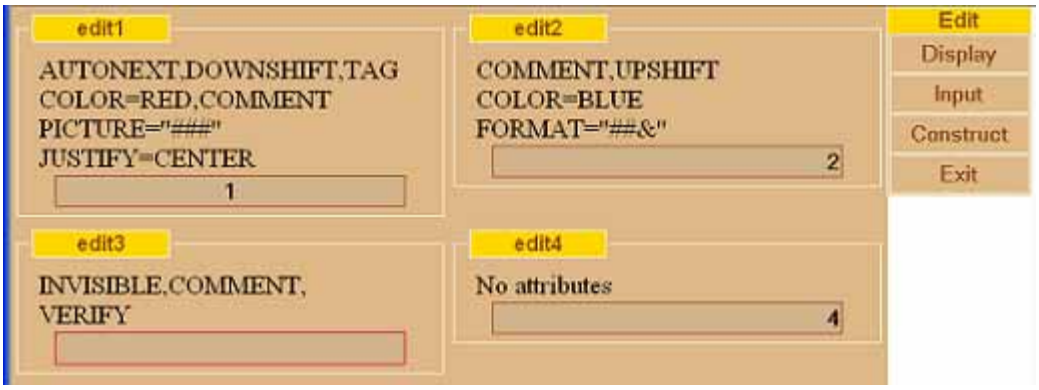
- The default Genero CSS.
- Declaring styles.
- Overriding existing styles.
- Adding user-defined styles.

The Default Genero CSS

The file **genero.css** lists the default rendering styles. This file is located in **$FGLASDIR/web/fjs/defaultTheme** directory. 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.

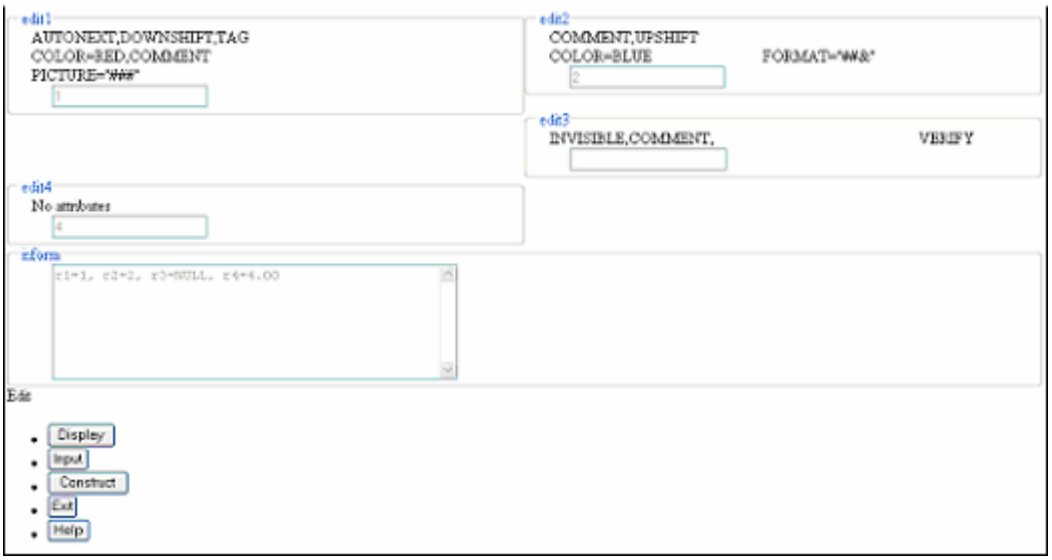
Example using CSS

The following screenshot is the Edit demonstration program using the default template **generodefault.html**, which specifies the **genero.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.



Declaring 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 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.

Adding a New 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.

```
01  <script language=javascript
src="/fjs/uaapi/webBrowser.js"></script>
02  <script language=javascript
src="/fjs/asapi/application.js"></script>
03  <script language=javascript
src="/fjs/defaultTheme/genero.js"></script>
04  <link rel="stylesheet" href="/fjs/defaultTheme/genero.css"
type="text/css"/>
    ...
05  <link rel="stylesheet" href="/css/mystyle.css" type="text/css"/>
```

Using the STYLE tag

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.

```
01  <script language=javascript
src="/fjs/uaapi/webBrowser.js"></script>
02  <script language=javascript
src="/fjs/asapi/application.js"></script>
03  <script language=javascript
src="/fjs/defaultTheme/genero.js"></script>
04  <link rel="stylesheet" href="/fjs/defaultTheme/genero.css"
type="text/css"/>
    ...
05  <style type="text/css">
06  .mystyle {
07     width: 100%;
08     background-color: beige;
09  }
10  </style>
```

Overriding existing 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 (**genero.css**) for the selector `.gDialog LI`. This selector is for a list item of the action panel (i.e., an action) as described in the rendered HTML section of the manual.

```
01  .gDialog LI {
02     cursor: pointer;
03     white-space: nowrap;
04     text-align: left;
05     border-bottom: 1px solid;
06     border-color: burlywood;
07  }
```

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:

```
01  .gDialog LI {
02     background-color: red;
03  }
```

As this is a new style, it will be added to the style information read from the **genero.css** file for this selector.

If a style already has been defined for the selector, you redefine the style. For example, to change the border appearance for the actions in the action panel, simply redefine the `border-bottom` style. To redefine the style, proceed as above and add the style information in your style declaration:

```
01  .gDialog LI {
02     background-color: red;
03     border-bottom: 3px dashed;
04  }
```

As the style `border-bottom` is defined twice (initially in the **genero.css** file and again in your style declaration), the one with the higher weight will be used.

Adding user styles

You can add your own styles for any widget by adding the `STYLE` attribute to your widget in the .per file. The **STYLE** attribute value is transmitted to the HTML **class** attribute.

```
01  EDIT f001=FORMONLY.field1, STYLE="style1 style2";
```

Generated HTML

```
01  <input class="edit currentField typeString style1 style2" id=field1
type=text value="">
```

Corresponding CSS

```
.style1 {
   width: 100%;
}

.style2 {
   border: 1px solid gold;
   background-color: beige;
}
```

Customize with Templates

While applications can be run with the default rendering, you may want to customize your application further. The provision of a template language makes this possible. An HTML page becomes a template when you introduce template language into the page.

In the typical web application development process, web designers take care of the HTML pages and developers work on business logic. When it is time to merge design and business logic, it can prove difficult and one often feels as if the work is being done twice.

The templates approach allows smoother passage of this step. Developers can add GWC template instructions to HTML pages; these pages can be modified later by designers without syntax errors.

In the Concepts chapter, the default template displays the entire application window. Using template language, you can choose the granularity of the default rendering, ranging from the entire application window to the value of a table cell.

Here is an example of how you can proceed:

1. Create your HTML page.
2. Locate the values or elements you want GWC to handle.
3. Add the corresponding template instructions.

For more information

- Refer to the Tutorial for a step-by-step example of a application being customized by changes to the template.

Customize with JavaScript

JavaScript has been added to ease the interaction with users and to refine the application's design. An application without JavaScript is fully functional but will not integrate new features like incremental mode, or the look and feel that makes GWC more user-friendly. Modifying default JavaScript files is not supported.

We recommend leaving the default JavaScript files unchanged. To add your own JavaScript, make a template and reference it as described at the beginning of this chapter.

Customization FAQ

This section presents Frequently Asked Questions regarding customization.

- How do I use a custom template ?
- How do I use a custom cascading style sheet ?

How do I use a custom template ?

You need to configure your application to use the custom template. This template is a type of style that you apply to an application window.

See Declare a template and Apply a template in the Customize chapter for more details.

How do I use a custom cascading style sheet ?

This CSS has to be referenced in the application HTML template.

For more information see Preparing Customization and Customize with CSS in the Customize chapter.

Migrating Genero Applications to GWC

An application is a program designed to complete a specific task. A Web application is an application designed for the Web. When discussing Web applications, a natural tendency is to think in terms of Web sites. There are two types of web sites: sites that provide static content and sites that provide functionality and data storage. The latter represents Web applications, and GWC belongs to this second category.

Web applications are gaining in popularity due to the ease of deployment to the end-user: a Web application is surfaced in a browser on the client machine; no additional software needs to be installed on the client. This does not mean, however, that all applications will work as Web applications. While Web applications can take advantage of an HTML environment and browser technology, they are also limited by the restrictions that are inherent with these technologies. This chapter explains precautions to take before writing a GWC application.

Topics

Business Logic and GWC

Certain Genero features are not supported by the GWC. In this section, these unsupported features are explained and, where possible, alternate solutions are specified.

- Accelerator Keys
- StartMenus
- MDI
- ProgressBars
- On Idle
- Front Calls
- Triggers
- RUN and RUN WITHOUT WAITING

Presentation Logic and GWC

The presentation of a GWC application is dependant on a variety of factors, including how the key elements of a Web application are presented, the size of the user agent, the fonts used for presentation, and so on.

Business Logic and GWC

A Genero application separates the business logic from the presentation logic, and these principles remain true for GWC applications. We advise you to become familiar with Genero guidelines first.

Accelerator keys

GWC does not support all accelerator keys. Browsers have their own accelerator keys (also known as *shortcuts*); application-defined accelerator keys may interfere with them. For example, if your application uses accelerator keys [F1] or [F5], they conflict with the Internet Explorer shortcuts for "Help" and "Reload".

The following table identifies the default accelerator keys supported by GWC, along with the action they are mapped to:

Supported Accelerator Key	Mapped Action
Return / Enter	accept
Tab	nextfield
Shift + Tab	prevfield
Home	firstrow
End	lastrow
Next / Page Up	nextpage
Previous / Prior / Page Down	prevpage

For example, if you press the [Enter] or [Return] key, an accept action is sent to the DVM, regardless of the accelerator defined by the .4ad file.

If your application uses accelerator keys, you must modify your application to use explicit actions, buttons, or commands.

StartMenus

StartMenus are not supported.

One possible implementation could be to create an application launcher. This program would execute `RUN ... IN FORM MODE WITHOUT WAITING` to run the StartMenuCommand.

MDI

Every application is in SDI.

ProgressBars

In a Genero application, a ProgressBar is updated without user interaction using a server-push technique. This is not implemented in the current version of GWC.

On Idle

On idle is not handled by the GWC.

Executing Client Programs using Front Calls

Front Calls executing client programs are not yet available. In an HTML environment, access to the client machine is limited and secured. As a result, a browser needs customer authorization to access a client resource. This would require GWC to implement a secured module using activeX or an applet.

Note that browsers have built-in features to open PDF, Microsoft Word, and Microsoft Excel files. For such operations, you simply need to set the URL where the files can be downloaded.

Triggers

Dialogs using many triggers require frequent communication between the browser and the DVM. Time lags experienced by this back-and-forth communication are a factor of the size and speed of your network. To minimize the impact of triggers, the GWC default communication modes reduce data exchange and update relevant parts of the page only (instead of refreshing the entire HTML page). This is transparent to the end-user.

From an application development perspective, realize that some complex instructions cannot be done by JavaScript unless the entire page is updated.

RUN and RUN WITHOUT WAITING

To use the statements `RUN` and `RUN WITHOUT WAITING`, you must also specify `IN FORM MODE`. This allows the newly-forked application to run in interactive mode.

A new browser is opened where the new application is executed.

- Using `RUN ... IN FORM MODE`, the first application is frozen and displays a pending page with this type of message: "Your request is currently being processed by the Application Server. Please wait... ". If you close the first application before the second one, you may encounter problems, as the first application is waiting for the second application to terminate.
- Using `RUN ... IN FORM MODE WITHOUT WAITING`, each application runs independently.

Example

```
RUN "fglrun run.42r" IN FORM MODE WITHOUT WAITING
```

Presentation Logic and GWC

The representation of an application can vary between different front ends. While GWC provides default rendering of the application, some representations are more suitable for a desktop client and others for HTML clients. Also, as JavaScript can handle a widget's look and behavior, the page processing time can vary. The more complex a page is, the more time it takes to be completely rendered.

Topics

Tips

This section provides tips that can assist you in successfully launching a Web application.

Styles

This section discusses how key elements are presented by the GWC.

Tips

- To prepare for customization, assign names to widgets you plan to customize.

  In generated HTML, if a form component name cannot be converted to HTML encoding, its entity equivalent is used. Genero transforms the character in HTML identity (&#xxx); for example, "佚-span". This can result in having an id attribute set to a value that corrupts the style definition for the tag:

  ```
  #&#20314;-span {
         width:25%;
  }
  ```

Therefore, applications designed for the GWC should follow the standards set by the W3C; the value for `id` and `name` attributes should begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), colons (":"), and periods (".").

- Resizable tables do not exist; adjust the table lines in the .per file.
- Use light pages (avoid tables in folder pages due to JavaScript process).

Styles

Forms are rendered as defined in the .per files. Genero styles (.4st) are not supported. To define the look of an application, you must use CSS to replace Genero styles. In other words, you must translate the styles defined in .4st into CSS styles.

Locate the key elements of your interface:

- Window
- Toolbar
- Menu
- Actions panel
- StatusBar
- StartMenu

The default rendering is an example; you can build your own renderer using CSS and JavaScript. The following explanations are made in the default rendering context.

Window

For GWC, the window is a part of the HTML page; it may also be the entire HTML page. Some window styles cannot be applied or do not make sense in the HTML environment.

GWC default behavior is to display **one window at a time**, except `RUN`, which forks a new browser. There is no default mechanism for multiple windows (such as creating a pop-up window).

ToolBar

GWC default behavior displays a toolbar at the top of the page, under the topmenu (if it exists). In Genero style we can say the default rendering sets startMenuPosition to "top".

Menu

GWC default behavior displays a ring menu on the right (ringMenuPosition set to right). Other styles are not handled, but rendering can be customized using CSS and Javascript.

Styles like dialog or popup do not open a new window. The default rendering does not handle these styles. However, the style attribute is transmitted, and you can make your own renderer to open a pop-up window.

Action panel

GWC default behavior displays an action panel on the right (actionPanelPosition set to right). This could be customized by CSS and JavaScript.

StatusBar

The StatusBar contains simple messages, error messages, and overwrite status. In GWC, the StatusBar is not represented as a single line. With CSS and JavaScript an error message can be an alert (pop-up dialog in modal mode), and a simple message sent to the browser status bar.

StartMenu

StartMenus are not yet supported. See Business Logic for GWC - StartMenus for an alternate solution.

For More Information

- For more information on customizing your applications read the chapter on Customization.
- For more information on supported and unsupported features, refer to GWC Features.
- For more information on the suggested directory structure for files supporting a GWC application, refer to GWC Application Directory Structure.

Migrating to GWC 1.33.1h

This section identifies changes that have occurred with the release of GWC 1.33.1h. In order to have the application rendered correctly by GWC 1.33.1h, you must make modifications to all applications developed with and tested against earlier versions of GWC.

Specifically, changes must be made in the following areas:

- RUN behavior
- Log configuration

RUN behavior

By default, the RUN command opens a new window. For technical reasons, to prevent undesired delay at the start up of a new application, the RUN command now refreshes the parent window.

The difference between the old and new behavior is the method in which the parent window is updated. A page can evolve with JavaScript, however the state the page has reached is lost the next time the page is refreshed. The new behavior reloads the entire page, resulting in a loss of possible modifications made with JavaScript on the page. The old behavior updated the page incrementally with JavaScript, and as a result maintained any JavaScript modifications to that page. To have your application keep the old behavior, you must modify the template. Immediately before the call to the JavaScript `gInitFieldMode` function, add the following instruction:

```
    gConfiguration.refreshPageMethodAfterNewApplication =
gINCREMENTAL_MODE;
```

Log configuration

The log can be configured to display additional information, such as tasks or processes handled by the Genero Application Server daemon (gasd). The output can be specified as either plain text or XML. XML output allows further analysis using XLST. For more information on configuring logging for the GWC, see the section on Logging in the *Genero Application Server Manual*.

Migrating to GWC 1.32.1f

This section identifies changes that have occurred with the release of GWC 1.32.1f. In order to have the application rendered correctly by GWC 1.32.1f, you must make modifications to all applications developed with and tested against earlier versions of GWC. If you migrating applications based on GWC 1.30.1j or earlier, you must also implement the changes identified in the section Migrating to GWC 1.30.1j.

Specifically, changes must be made in the following areas:

- Deprecated template path and new template path
- New CSS class

Deprecated template path and new template path

Because of the new layout engine, the template path `document/styles` is no longer used and is deprecated. It has been replaced by the template path `document/layoutData` that contains layout information used by a Javascript method to have the widgets in the form properly aligned.

The former path was used to fill in the content of a HTML *style* tag:

```
<style type="text/css" gwc:content="document/styles"></style>
```

The new template path must be used as initialization data for the Javascript variable *gLayoutData*. This initialization must be done prior to the call to the *gInitFieldMode* function:

```
<script defer language=javascript gwc:contentprotocol="string"><!--
  var gLayoutData = ${document/layoutData};
  gInitFieldMode( gIdToElement('gDialogForm'), gSMART_FIELD_MODE,
gINCREMENTAL_MODE, ${configuration/timeout/useragent - 3} );
//--></script>
```

The `gLayoutData` variable is not required in order to have a GWC application to work. However, omitting it would result in having all the form fields misaligned and packed on the left side of the window.

New CSS class

With previous version of the GWC, the width style of form elements like the Edit widget, Labels, etc was set to 100% so the widgets could fit in their containing box and the size

of the containing box was set using the *document/styles* template path. With the new layout engine the form content is no longer proportional to the browser's window size and form elements tend to keep their natural size.

The Javascript function used to have the elements in the form properly aligned uses the elements size to calculate what their containing box's width should be. The width style of 100% have to be avoided to ensure correct calculations are done by the function. However, after the calculation process is done, a widget can be too small with regards to its containing box's size. That's why the new CSS class `gFill` has been added. This class is set on the *Grid* elements of the form after the layout function has finished is job and sets the width of the widgets to 100%. For example:

```
.gFill .gEdit,
.gFill .gTextEdit,
.gFill .gDateEdit,
.gFill .gButtonEdit {

  width: 100%;
}
```

So if you use custom styles with your GWC appllications, be sure to update the styles for the widgets and add the styles associated to the `gFill` selector.

Migrating to GWC 1.30.1j

This section identifies changes that have occurred with the release of GWC 1.30.1j. In order to have the application rendered correctly by GWC 1.30.1j, you must make modifications to all applications developed with and tested against earlier versions of GWC. If you migrating applications based on GWC 1.30.1d or earlier, you must also implement the changes identified in the section Migrating to GWC 1.30.1d.

Specifically, changes must be made in the following areas:

- Top-level template
- Customized templates

New top-level template

The default template page no longer uses the *window* template language path. Instead, it uses a table to lay out the application elements like the toolbar, the form, the action panel, and so on. Using a table allows for easier placement of the application elements.

Changes to customized templates

If you have customized templates, you need to make the following changes:

- If you have a form in your template, you must add a line after your `<form>` start:

```
<div gwc:replace="application/intermediatetrigger" />
```

- Change the gInitFieldMode JavaScript line of initialization to match the following, where *ratio* is an arbitrary value between 0 and 1 (such as "0.8"):

```
<script language="javascript" gwc:contentprotocol="string">
  gInitFieldMode( gIdToElement('gDialogForm'), gSMART_FIELD_MODE,
gINCREMENTAL_MOD, ${configuration/timeout/useragent * ratio} );  /// [
gFIELD_MODE | gSMART_FIELD_MODE ], [ gINCREMENTAL_MODE | gFULL_MODE ],
[keep-alive interval]
</script>
```

  Making this modification sets up the keep-alive feature. *configuration/timeout/useragent* is the timeout value GWC waits before considering the connection with the user agent lost. So you must give the gInitFieldMode function a lower value than this value in order to have the keep-alive feature work.

Note that while your application will work without making these changes, not applying the changes can lead to future problems.

Migrating to GWC 1.30.1d

This section identifies changes that have occurred with the release of GWC 1.30.1d. In order to have the application rendered correctly by GWC 1.30.1d, you must make modifications to those applications developed with and tested against earlier versions of GWC. If you migrating applications based on GWC 1.30.1b or earlier, you must also implement the changes identified in the section Migrating to GWC 1.30.1c.

Specifically, changes must be made in the following areas:

- Application configuration file extension

Changing the application configuration file extension

For external applications, the extension of application configuration files has been changed from `.xml` to `.xcf`. For example, the sample application configuration file located in `$FGLASDIR/app/` and named `Edit.xml` has been renamed `Edit.xcf`.

Migrating to GWC 1.30.1c

This section identifies changes that have occurred with the release of GWC 1.30.1c. In order to have the application rendered correctly by GWC 1.30.1c, you must modify the templates for all applications developed with and tested against earlier versions of GWC.

Specifically, changes must be made in the following areas:

- CSS
- Javascript
- HTML
- Referencing
- Template expressions
- as.xcf changes

Changes in CSS

CSS classes are now prefixed with "g". For example, the "menu" class is now named "**gMenu**". This change was implemented to prevent confusion with non-GWC classes.

As a result, if you have customized the CSS, please prefix the GWC classes with a "g". If this change is not implemented, the layout will not render correctly.

Changes in JavaScript

Public JavaScript API are now prefixed with "g". For example, in the default template **generodefault.html**:

```
    InitFieldMode( IdToElement('dialogForm'), SMART_FIELD_MODE,
INCREMENTAL_MODE );
```

is replaced by

```
    gInitFieldMode( gIdToElement('gDialogForm'), gSMART_FIELD_MODE,
gINCREMENTAL_MODE );
```

If GWC JavaScript functions do not include the "g" prefix, the functions are reserved for internal usage. We recommend that you do not change these internal functions.

You have been provided with public APIs to use, such as "gGWCEvent".

Changes in HTML

Most generated HTML elements have an id attribute. The id values are also prefixed by "g".
For example:
dialogForm which references the <form> element is now named **gDialogForm,** and "workspace" becomes **gWorkspace**.

GWC JavaScript handles all elements defined in gDialogForm. Your HTML form must be defined as in the following example:

```
    <form action="page.html" id="gDialogForm" method=post
gwc:attributes="action document/URL">
      <input disabled=disabled id=gDBDate name=gDBDate type=hidden
value="MDY4/">
      ...
    </form>
```

The id and action attributes are required. The action attribute is then replaced by the application URL dynamically. gDbDate is used to handle date input for the calendar. So, if you have an application with the dateEdit widget, add the gDbDate line.

If you have written your own templates, ensure that you made these changes.

Changes in referencing a template

Template declaration has been simplified. You no longer need a .4st file. The template is referenced inside the application configuration using the style name.

Example

```
    <APPLICATION ...>
      <RESOURCE Id="mytemplate"
Source="FILE">/home/app/tpl.html</RESOURCE>
      ...
      <EXECUTION> ... </EXECUTION>
      <OUTPUT>
        <MAP Id="DUA_GWC">
          <THEME>
            <TEMPLATE Id="mystyle">$(mytemplate)</TEMPLATE>
            <TEMPLATE Id="style2">$(tpl2)</TEMPLATE>
          </THEME>
        </MAP>
      </OUTPUT>
    </APPLICATION>
```

In this example, the template identified as "mystyle" uses the resource "mytemplate". The resource "mytemplate" is an external file resource using the file "/home/app/tpl.html". This template is applied when a window is opened with the style "mystyle".

Changes in Template Expressions

A change has been made in the template expressions syntax when using conversion protocol. It now uses a parentheses format. See the conversion protocol chapter for more details.

Changes in as.xcf

The DTD of `as.xcf` has changed.

- Prior to version 1.30.1c, the `<THEME_COMPONENT>` tag was written as:

```
    <THEME_COMPONENT Id="cpn.theme.default.gwc">
      <TEMPLATE>$(res.theme.default.gwc.template)</TEMPLATE>
      <END>$(res.theme.default.html.end)</END>
      <ERROR>$(res.theme.default.html.error)</ERROR>
      <TIMEOUT>$(res.theme.default.html.timeout)</TIMEOUT>

<TRANSACTION_PENDING>$(res.theme.default.html.transaction)</TRANSACTION
_PENDING>
    </THEME_COMPONENT>
```

  As of version 1.30.1c, the `<THEME_COMPONENT>` tag is to be written as follows:

```
    <THEME_COMPONENT Id="cpn.theme.default.gwc">
      <TEMPLATE
Id="_default">$(res.theme.default.gwc.template)</TEMPLATE>
      <TEMPLATE Id="_end">$(res.theme.default.html.end)</TEMPLATE>
      <TEMPLATE Id="_error">$(res.theme.default.html.error)</TEMPLATE>
      <TEMPLATE
Id="_timeout">$(res.theme.default.html.timeout)</TEMPLATE>
      <TEMPLATE
Id="_transaction">$(res.theme.default.html.transaction)</TEMPLATE>
      <TEMPLATE
Id="_launch">$(res.theme.default.html.launch)</TEMPLATE>
    </THEME_COMPONENT>
```

- The tag `<USER_AGENT_2>` has been removed

- On Windows, '`/`' can now be used as the directory separator. As a result of this change, all platform-dependent resources have been updated using '`/`'.

Template CSS Reference

With Cascading Style Sheets (CSS), you can customize the look and feel of your application. The default theme provides an example of customization. Used in conjunction with the rendered HTML reference page, this section helps you interpret a CSS file.

Topics

CSS Syntax

CSS uses a specific syntax to define the style of HTML elements. This section presents the various syntax options.

Template CSS

This section lists available selectors by category (Containers, FormFields, Dialogs, and Others).

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

Note

The sections TABLE, MENU, and TOPMENU are followed by a link named "Samples". Follow this link to view a graphical explanation of each selector and what it controls.

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

CSS Reference Samples

In this section, graphical examples involving selectors for Input Array/Display Array, Menu, and TopMenu are provided.

For a complete list of selectors, refer to the section CSS Reference.

Input Array / Display Array Sample

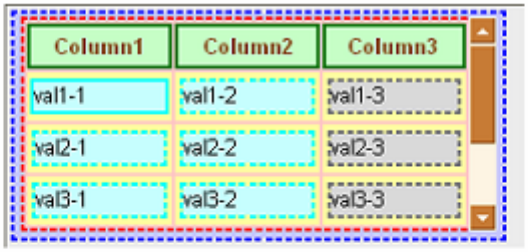
This sample shows the relationship between the CSS selectors for Table widgets and the elements defining the Table in the .PER file. Note that the third column of the table has its *NOENTRY* attribute set.

This sample shows the table in two states: *INPUT mode* and *DISPLAY mode*

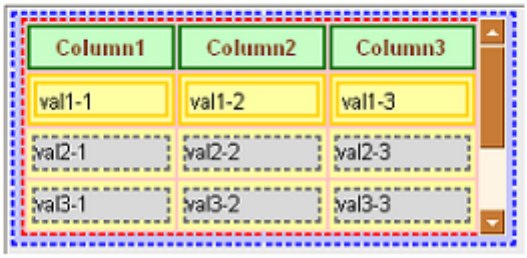
This sample was made by adding a border style and a background color style to the listed CSS selectors, e.g. `.gTableBox { border: 2px dashed blue; background-color: #CCCCFF; }`. The rendered HTML code listed is a 'light' version of the HTML generated by the GWC.

CSS Selectors	
.gTableBox	The DIV tag containing the table
.gTable	The table container
.gTable THEAD TR TH	A table column header
.gTable TR	A table row
.gTable .gCurrentRow *	Any descendant of the current row
.gTable TD *	Any descendant of a table cell
.gTable TD .gCurrentField	The current input field
.gTable .disabled	A disabled table cell

Rendering



input array



display array

.PER File (Input Array)

```
TABLE
{
  Column1   Column2   Column3
[ edt1    | edt2    | edt3    ]
[ edt1    | edt2    | edt3    ]
[ edt1    | edt2    | edt3    ]
}
END
ATTRIBUTES
EDIT edt1=formonly.edt1;
EDIT edt2=formonly.edt2;
EDIT edt3=formonly.edt3, NOENTRY;
END
```

Rendered HTML (Input Array)

```
<DIV class=gTableBox id=g63-div>
<TABLE class=gTable id=g63>
<THEAD>
 <TR>
  <TH><INPUT class="gTableHeader disabledTableHeader"
  value=Column1></TH>
  <TH><INPUT class="gTableHeader disabledTableHeader"
  value=Column2></TH>
  <TH><INPUT class="gTableHeader disabledTableHeader"
  value=Column3></TH>
 </TR>
</THEAD>
<TBODY>
 <TR>
 <TD><INPUT class="gEdit gCurrentField" value=val1-1></TD>
 <TD><INPUT class="gEdit " value=val1-2></TD>
 <TD><INPUT class="gEdit disabled" value=val1-3></TD>
 </TR>
 <TR>
 <TD><INPUT class="gEdit " value=val2-1></TD>
 <TD><INPUT class="gEdit " value=val2-2></TD>
 <TD><INPUT class="gEdit disabled" value=val2-3></TD>
 </TR>
 <TR>
 <TD><INPUT class="gEdit " value=val3-1></TD>
 <TD><INPUT class="gEdit " value=val3-2></TD>
 <TD><INPUT class="gEdit disabled" value=val3-3></TD>
 </TR>
</TBODY>
</TABLE>
</DIV>
```

.PER File (Display Array)

```
TABLE
{
  Column1      Column2      Column3
[ edt1       | edt2       | edt3       ]
[ edt1       | edt2       | edt3       ]
[ edt1       | edt2       | edt3       ]
}
END
ATTRIBUTES
EDIT edt1=formonly.edt1;
EDIT edt2=formonly.edt2;
EDIT edt3=formonly.edt3, NOENTRY;
END
```

Rendered HTML (Display Array)

```
<DIV class=gTableBox id=g63-div>
<TABLE class=gTable id=g63 style="BORDER-RIGHT: 17px solid">
<THEAD>
 <TR>
  <TH><INPUT class="gTableHeader " id=formonly_edt1 type=submit
  value=Column1 name=64></TH>
  <TH><INPUT class="gTableHeader " id=formonly_edt2 type=submit
  value=Column2 name=66></TH>
  <TH><INPUT class="gTableHeader " id=formonly_edt3 type=submit
  value=Column3 name=68></TH>
 </TR>
</THEAD>
<TBODY>
 <TR class=" gCurrentRow">
  <TD><INPUT class="gEdit disabled" value=val1-1></TD>
  <TD><INPUT class="gEdit disabled" value=val1-2></TD>
  <TD><INPUT class="gEdit disabled" value=val1-3></TD>
 </TR>
 <TR>
  <TD><INPUT class="gEdit disabled" value=val2-1></TD>
  <TD><INPUT class="gEdit disabled" value=val2-2></TD>
  <TD><INPUT class="gEdit disabled" value=val2-3></TD>
 </TR>
 <TR>
  <TD><INPUT class="gEdit disabled" value=val3-1></TD>
  <TD><INPUT class="gEdit disabled" value=val3-2></TD>
  <TD><INPUT class="gEdit disabled" value=val3-3></TD>
 </TR>
</TBODY>
</TABLE>
</DIV>
```

Menu Sample

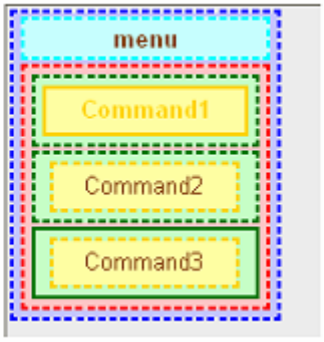
This sample shows the relationship between the CSS selectors for Menus and the elements defining the Menu in the .4GL file. The *Close* and *Help* menu actions are automatically added by the DVM and hidden by the GWC if they aren't found in the menu actions list. The third menu command has the mouse focus in the running application.

This sample was made by adding a border style and a background color style to the listed CSS selectors, e.g. `.gMenu { border: 2px dashed blue; background-color: #CCCCFF; }`. The rendered HTML code listed is a 'light' version of the HTML generated by the GWC.

CSS Selectors

.gMenu	The DIV tag containing the menu
.gMenu SPAN	The menu title
.gMenu UL	The menu items list
.gMenu LI	A menu item
.gMenu LI.hover	A menu command having the mouse cursor over it
.gMenu INPUT	A menu command
.gMenu LI.gCurrentAction INPUT	The current command
	Element not shown on screenshot

Rendering



.4GL File

```
MENU "menu"
  COMMAND "Command1"
  ...
  COMMAND "Command2"
  ...
  COMMAND "Command3"
  ...
END MENU
```

Rendered HTML

```
<DIV class=gMenu id=g63>
 <SPAN class=gMenuTitle>menu</SPAN>
 <UL>
  <LI class="gMenuAction gHidden">
  <INPUT class=gAction value=Close name=close>
  <LI class="gMenuAction gCurrentAction ">
  <INPUT class=gAction value=Command1 name=command1>
  <LI class="gMenuAction ">
  <INPUT class=gAction value=Command2 name=command2>
  <LI class="gMenuAction ">
  <INPUT class=gAction value=Command3 name=command3>
  <LI class="gMenuAction gHidden">
  <INPUT class=gAction value=Help name=help></LI>
 </UL>
</DIV>
```

TopMenu Sample

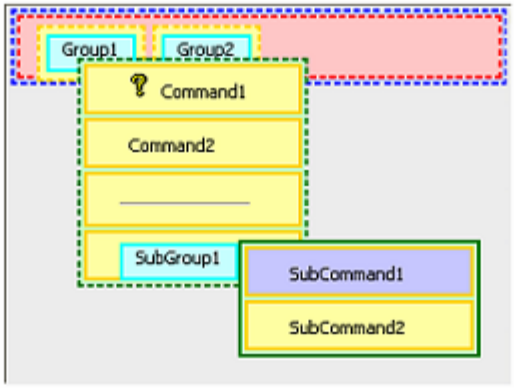
This sample shows the relationship between the CSS selectors for TopMenus and the elements defining the TopMenu in the .PER file. The command *SubCommand1* has the mouse focus in the running application.

This sample was made by adding a border style and a background color style to the listed CSS selectors, e.g. `.gTopMenu { border: 2px dashed blue; background-color: #CCCCFF; }`. The rendered HTML code listed is a 'light' version of the HTML generated by the GWC.

CSS Selectors

.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 list
.gTopMenu LI LI	A TopMenu group item
.gTopMenu UL UL UL	A sub-group's items list
.gTopMenu label	A TopMenu group's text
.gTopMenu .hover	A TopMenu group item having the mouse cursor over it
	Element not shown on screenshot

Rendering



.PER File

```
TOPMENU
  GROUP (TEXT="Group1")
    COMMAND command1 (TEXT="Command1", IMAGE="quest")
    COMMAND command2 (TEXT="Command2")
    SEPARATOR
    GROUP (TEXT="SubGroup1")
      COMMAND subcommand1 (TEXT="SubCommand1")
      COMMAND subcommand2 (TEXT="SubCommand2")
    END
  END
  GROUP (TEXT="Group2")
    COMMAND command3 (TEXT="Command3")
    COMMAND command4 (TEXT="Command4")
  END
END
```

Rendered HTML

```
<DIV class=gTopMenu id=g62>
<UL>
<LI>
<LABEL>Group1</LABEL>
<UL>
<LI>
<INPUT class=gAction src="/pic/quest" name=command1>
<INPUT class=gAction value=Command1 name=command1>
<LI>
<INPUT class=gAction value=Command2 name=command2>
<LI>
<HR>
<LI>
<LABEL>SubGroup1</LABEL>
<UL>
<LI>
<INPUT class=gAction value=SubCommand1 name=subcommand1>
<LI>
<INPUT class=gAction value=SubCommand2
name=subcommand2></LI>
</UL>
</LI>
</UL>
<LI>
<LABEL>Group2</LABEL>
<UL>
<LI>
<INPUT class=gAction value=Command3 name=command3>
<LI>
<INPUT class=gAction value=Command4 name=command4></LI>
</UL>
</LI>
</UL>
</DIV>
```

Template Language Reference

Genero Web Client provides a template language to create dynamic templates.

Topics

Genero Web Client namespace

A Genero Web Client template instruction is prefixed by "gwc". Genero Web Client processes the template instruction and generates new HTML code. This section presents the syntax for the Genero Web Client namespace.

Template instructions

Genero Web Client instruction specify the kind of operations you can perform. This section lists valid template instructions and valid syntax, and identifies the weight (processing priority) of each template instruction.

Template resources

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.

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

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:contentprotocol`	Specify the way an expression is processed.
`gwc:omit-tag`	Suppress the surrounding tag after instructions in the children tags have been processed.

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.

---

define instruction

The define instruction declares a local variable to be used in the HTML tag or the elements it contains. This variable is undefined outside of these tags.

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 condition is verified, the following Genero Web Client instruction is processed; otherwise the tag and its children are removed.

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

Repeat the current line for each element in the repeat condition.

Syntax

```
<tag gwc:repeat="elt eltList">
    stats
</tag>
```

Notes

1. *eltList* is the list of elements to loop on
2. *elt* is an element of eltList
3. *stats* are statements repeated if there are still elements in 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.

---

replace instruction

Replace the tag 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 tags 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="setAtt [; ...] " ...>
...
</tag>
```

Where *setAttr* is:

```
att expression
```

Notes

1. *att* is the attribute name
2. *expression* is the attribute value. Notice there is no equal sign (=) between the attribute name and its value.

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 between the HTML tags

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.

---

contentprotocol instruction

This instruction tells Genero Web Client that the content of the tag has to be processed by a specified parser.

Syntax

```
<tag gwc:contentprotocol="protocol">
expression
</tag>
```

Notes

1. *expression* is any valid expression
2. The protocol defined in the `gwc:contentprotocol` instruction is not the same as the one used for the conversion protocol in template expressions, except for the string protocol. In fact, only the string protocol has a special behavior within a `gwc:contentprotocol` instruction.
   Thus, having a tag like

   ```
   <DIV gwc:contentprotocol="bool" > expression </DIV>
   ```

3. behaves the same as the tag

   ```
   <DIV gwc:content="expression" ></DIV>
   ```

4. This means no conversion is done. On the other hand, the tag

   ```
   <DIV gwc:contentprotocol="string" > expression </DIV>
   ```

5. will produce the same result as the tag

   ```
   <DIV gwc:content="string:expression" ></DIV>
   ```

6. This can be useful in SCRIPT tags, to have 'well-formatted' scripts which use Genero Web Client template paths:

   ```
   <SCRIPT language="javascript" gwc:contentprotocol="string" >
   function displayFormfieldValue()
   {
     alert("Formfield edt1 value: ${formfield[edt1]/value}");
   }
   </SCRIPT>
   ```

Example

```
<div gwc:contentprotocol="string">action[accept]/text</div>
```

This displays action[accept]/text as a string (it leaves the text as it is) and does not have a Genero Web Client widget.

---

omit-tag instruction

Remove the surrounding tag after all children tags instructions have been processed

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.

Template resources

Template resources defined in a configuration file can be accessed using the following syntax.

Syntax

```
$( resource_identifier )
```

Note

1. *resource_identifier* is the value of the *id* attribute of the resource
2. template resources can be used anywhere in the parsed HTML document
3. template resources are merged into the document before the evaluation of the template instructions
4. some resources are defined and added to the configuration at run-time

Resource Identifier	Description
`application.id` `application.name`	Value of the id attribute of the application tag.
`application.start.uri`	Launching URL of the application.
`connector.uri`	Connector part of the URL.
`server.version`	Version of the running Genero Web Client.

Example

```
<RESOURCE Id="res.meta-tags" Source="INTERNAL"><![CDATA[
 <meta http-equiv="Pragma" content="no-cache">
 <meta http-equiv="Cache-Control" content="no-cache">
]]></RESOURCE>

<RESOURCE Id="res.restart" Source="INTERNAL"><![CDATA[
 <a href="$(application.start.uri)">Try again ...</a>
]]></RESOURCE>
```

```
<RESOURCE Id="res.theme.default.html.end" Source="INTERNAL"><![CDATA[
 <html>
  <head>
   $(res.meta-tags)
   <title>GWC - Version $(server.version) - Running
$(application.name)</title>
  </head>
  <body>
   Thank you. Please visit us again.
   <hr>
   $(res.restart)
  </body>
 </html>
]]></RESOURCE>
```

Template expressions

In this section, template expressions are discussed.

- expressions syntax
- conversion protocols
- string: protocol
- raw: protocol
- path: protocol

---

expressions syntax

Template expressions respect a defined syntax.

Syntax

expression is:

```
{ expression op expression | unary_op expression | ( expression ) |
conversion_protocol( expression ) | string: string_expression | raw:
expression | path: template_path | template_path | numeric | boolean |
string_value | expression_as_string in expression_as_string }
```

where *boolean* is:

```
{ TRUE | FALSE }
```

where *string_value* is:

```
' string '
```

where *op* is:

```
{ arithmetic_op | boolean_op | comparison_op }
```

where *arithmetic_op* is:

```
{ + | - | * | / | % }
```

where *boolean_op* is:

```
{ && | || }
```

where *comparison_op* is:

```
{ == | != | < | <= | > | >= }
```

where *unary_op* is:

```
{ + | - | ! }
```

where *conversion_protocol* is:

```
{ bool | expr | url | html | js }
```

where *string_expression* is:

```
{ [ string ] | [ ${ template_path } ] } [...]
```

Notes

1. *string* is a string value
2. *numeric* is a numeric value
3. *expression_as_string* is any template expression resulting in a string value
4. *template_path* is any valid template path
5. 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
   ```

6. 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

8. **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
```

9. 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
```

11. 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
```

---

conversion protocols

A conversion protocol converts the expression inside parentheses using the appropriate protocol:

```
bool(1+1) != false
```

The bool conversion protocol converts the value of its expression the same way the `gwc:condition` instruction tests its expression.

Examples

```
expr(1)                    # produces 1
expr('1234')               # produces 1234
expr('abcd')               # produces 0

bool(0)                    # produces 0
bool(1234)                 # produces 1
bool('abcd')               # produces 1
```

Conversion tables

`bool` and `expr` conversion protocols behave according to the following table:

Protocol	Expression value			
	0	other numeric value	empty string	not empty string
bool	0	1	0	1
expr	0	numeric value	0	0 unless the string is the representation of a numeric value.

`url` and `html` conversion protocols proceed character by character. Note that implicit HTML conversion is done in a `gwc:content` instruction and in a `gwc:replace` instruction if the resulting expression is a string. To avoid this conversion, or if you want to convert only a part of the expression, use the `html` operator in association with a `raw` operator.

Protocol	Character value					
	<	>	&	"	other ASCII	non ASCII
html	&lt;	&gt;	&amp;	&quot;	no changes	
url	see URL ASCII conversion table					URL encoding*

** non ASCII characters are encoded using the %hh syntax, hh being the hexadecimal value of the character. Encoding depends on the HTML page encoding.*

URL ASCII conversion table:

	Character value					
	A to Z	a to z	0 to 9	space	- _ . ! ~ * ' ( )	other values
url encoding	no changes			+	no changes	standard encoding*

** special ASCII characters are encoded using the %hh syntax, hh being the hexadecimal value of the character. Encoding depends on the HTML page encoding.*

With an ISO-8859-1 encoding for the HTML page:

```
url('été')                # produces %e9t%e9
```

With an UTF-8 encoding for the HTML page:

```
url('été')                # produces %c3%a9t%c3%a9
```

`js` conversion protocol escapes special characters by backslashing them:

	Character value			
	quote	**double quote**	**backslash**	**new line**
New character value	\'	\"	\\	\n

---

string: protocol

With the `string:` instruction, everything behind the colon sign is considered a string. Server paths are accessed by putting them inside the `${` and `}` tags.

Special characters like the quote (') character, the curly brace (}) character, etc., can be escaped inside a string using the backslash (\ ) character.

The following two examples result in the same values.

Example

```
string:Formfield edt1 value: ${formfield[edt1]/value}
'Formfield edt1 value: ' + formfield[edt1]/value
```

---

raw: protocol

The `raw:` instruction tells Genero Web Client to not use implicit HTML conversion once the expression after the colon sign has been processed. Implicit HTML conversion is done only in a `gwc:content` instruction or in a `gwc:replace` instruction and if the resulting expression is a string. This operator has to be used only as the first instruction of the expression to be effective.

Example

```
<span gwc:content="'<b>html</b>'" />          # produces
<span>&lt;b&gt;html&lt;/b&gt;</span>
```

```
<span gwc:content="raw:'<b>html</b>'" />     # produces
<span><b>html</b></span>
```

---

path: protocol

The `path:` instruction acts like the `string:` instruction. The expression after the colon sign is considered as a template path. The expressions `path:`*`template_path`* and *`template_path`* are equivalent.

Template paths

With template paths, you can access most elements in your application, as well as information about the Application Server.

- Application Server Paths
- Web Server Paths
- Genero Application Paths
- Other Paths

Application Server Paths

Server

Template Path	Description
`server/paths`	List of available paths.
`server/version`	Version of the server.

Web Server Paths

Document

Template Path	Description
`document/url`	URL of the next document to get. Used to set the action attribute of a form HTML tag.
`document/layoutData`	Generated layout data which determine the alignment of Genero elements.
`document/errors`	List of errors in the document.

Tip: You can use document/errors path when working with templates. This displays a message when an error occurs in a template instruction or expression. In development, you set this in the default template $FGLASDIR/tpl/generodefault.html.

Genero Application Paths

- Application path
- Window path
- Dialog path
- Menu path
- Action path
- Forms
- Containers
- Items
- TopMenu path
- ToolBar path

Application path

Template Path	Description
`application`	Access path to the application.
`application/text`	Application title (text attribute of the UserInterface node)
`application/actions`	Get the list of the application actions. Used in a repeat instruction to loop on each action of the application.

Window path

Template Path	Description
`window` `application/window`	Access path to the entire application window.
`window/text`	Title of the window. For example: • in the .per file: `LAYOUT (TEXT="mytitle")` • In the .4gl file: `OPEN WINDOW ... ATTRIBUTES (TEXT="mystyle")`

Dialog path

Template Path	Description
`dialog` *`window`*`/dialog`	Access path to the current dialog.
`dialog``/active`	True if the dialog is active.
`dialog``/action[`*`act`*`]`	Action widget that is named *act*.
`dialog``/actions`	Get the list of dialog actions. Used in a repeat instruction to loop on each action of the dialog.

Notes

1. *window* is any `window` access path
2. *dialog* can be any `dialog` access path
3. *act* is the action identifier
4. for any action path see action path section

Menu path

Template Path	Description
`menu`	Access path to the current menu.

window/menu	
menu/active	True if the menu is active.
menu/text	Text of the menu.
menu/style	Style of the menu. Can be 'default', 'dialog' or 'popup'.
menu/action[*act*]	Action widget that is named is *act*.
menu/actions	Get the list of menu actions. Used in a repeat instruction to loop on each action of the menu.

Notes

1. *window* is any window access path
2. *menu* can be any menu access path
3. *act* is the action identifier
4. for any action path see action path section

Action path

Template Path	Description
action[*act*] application/action[*act*] *dialog*/action[*act*] *menu*/action[*act*]	Access paths to default action elements, defined in 4ad file.
actions	Get the list of the application actions. Used in a repeat instruction to loop on each action of the application.
action/active	True if the action is active.
action/id	Action identifier, usually its name.
action/name	Action name.
action/text	Action displayed text.
action/comment	Comment for this action.
action/hidden	True if this action is hidden.

Notes

1. *act* is the action identifier
2. *dialog* is any dialog access path
3. *menu* is any menu access path
4. *action* is any action access path

Form path

Template Path	Description
`form` *`window`*`/form`	Access path to a visible form.
`form``/width`	Get the width in characters of the form.

Notes

1. *window* is any `window` access path
2. *form* is any `form` access path

Container path

Template Path	Description
`container[`*`cname`*`]` *`window`*`/container[`*`cname`*`]`	Access path to a container.
`container``/name`	Get the name of the container.

Notes

- *window* is any `window` access path
- *container* is any `container` access path
- *cname* is the container identifier

Within the context of containers, you can examine:

- Table path
- Scrollgrid path
- Folder path

Table path

Template Path	Description
`table[`*`tname`*`]` *`window`*`/table[`*`tname`*`]`	Access path to the table.
`table``/id`	Get the idref of a table.
`table``/active`	True if the table is active.
`table``/size`	RowCount, total number of rows in the table.
`table``/offset`	Get the current row index in the table.
`table``/pagesize`	Get the number of rows to be displayed. Usually corresponds to the number of the rows defined in .per file.
`table``/pages`	Get the list of table pages. Used in a repeat instruction to loop on each page of the table.
`table``/page[`*`pidx`*`]`	Get the table page. Value for pidx can only be next, previous or current. This path will NOT render the previous or the next table page. It always renders the current page. The next and previous pidx should only be used for test purposes, i.e. in a `gwc:condition` instruction.
`table``/page[`*`pidx`*`]/id`	Get the id of the page. This corresponds to the offset of the first row displayed on this page.
`table``/columns`	Get the list of header columns.
`table``/column[`*`cname`*`]`	Get the table header column.
`table``/column[`*`cname`*`]/id`	Get the identifier of header column.
`table``/column[`*`cname`*`]/name`	Get the name of header column.
`table``/column[`*`cname`*`]/text`	Get the title of header column.
`table``/rows`	Get the list of table rows to be displayed. Used in a repeat instruction to loop on each row of the table.
`rowElt``/cells`	Get the list of cells on row *rowElt* given by a previous *`table`*`/rows` instruction. Used in a repeat instruction to loop on each cells of the row..
`rowElt``/cell[`*`cname`*`]`	Get *cname* cell of the rows *rowElt*.
`rowElt``/cell[`*`cname`*`]/id`	Get the id of cell in column *cname* and row *rowElt*.
`rowElt``/cell[`*`cname`*`]/value`	Get the value of cell in column *cname* and row *rowElt*.

Notes

1. *window* is any `window` access path

2. *table* is any `table` access path
3. *tname* is the name of the table
4. *pidx* is the page index. The row offset begins from 0
5. *cname* is the column name
6. *rowElt* is an element of *`table/rows`*

Scrollgrid path

Template Path	Description
`scrollgrid[`*`sname`*`]` *`window`*`/scrollgrid[`*`sname`*`]`	Access path to the scrollgrid.
`scrollgrid``/id`	Get the idref of a scrollgrid.
`scrollgrid``/active`	True if the scrollgrid is active.
`scrollgrid``/size`	RowCount, total number of rows in the scrollgrid.
`scrollgrid``/offset`	Get the current row index in the scrollgrid.
`scrollgrid``/pagesize`	Get the number of rows to be displayed. Usually corresponds to the number of the rows defined in the .per file.
`scrollgrid``/pages`	Get the list of scrollgrid pages. Used in a repeat instruction to loop on each page of the scrollgrid.
`scrollgrid``/page[`*`pidx`*`]`	Get the scrollgrid page. Value for pidx can only be next, previous or current.
`scrollgrid``/page[`*`pidx`*`]/id`	Get the id of the page. This corresponds to the offset of the first row displayed on this page.
`scrollgrid``/rows`	Get the list of scrollgrid rows to be displayed. Used in a repeat instruction to loop on each row of the scrollgrid. Works only with matrix scrollgrid.
`rowElt``/cells`	Get the list of cells on row *rowElt* given by a previous *`scrollgrid`*`/rows` instruction. Used in a repeat instruction to loop on each cells of the row. Works only with matrix scrollgrid.
`rowElt``/cell[`*`cname`*`]`	Get *cname* cell of the rows *rowElt*. Works only with matrix scrollgrid.
`rowElt``/cell[`*`cname`*`]/id`	Get the id of cell in column *cname* and row *rowElt*. Works only with matrix scrollgrid.

rowElt`/cell[`*cname*`]/value`	Get the value of cell in column *cname* and row *rowElt*. Works only with matrix scrollgrid.
scrollgrid`/cells`	Get the list of formfield cells of the scrollgrid. Used in a repeat instruction to loop on each cell of the scrollgrid. Doesn't work with matrix scrollgrid.
scrollgrid`/cell[`*cellname*`]`	Get formfield cell *cellname* of the scrollgrid. Doesn't work with matrix scrollgrid.
scrollgrid`/cell[`*cellname*`]/id`	Get the id of formfield cell *cellname*. Doesn't work with matrix scrollgrid.
scrollgrid`/cell[`*cellname*`]/value`	Get the value of formfield cell *cellname*. Doesn't work with matrix scrollgrid.

Notes

1. *window* is any `window` access path
2. *scrollgrid* is any `scrollgrid` access path
3. *sname* is the name of the scrollgrid
4. *pidx* is the page index. The row offset begins from 0
5. *cname* is the column name
6. *rowElt* is an element of *scrollgrid*`/rows`

Folder path

Template Path	Description
`folder[`*fname*`]` *window*`/folder[`*fname*`]`	Access path to the folder.
folder`/pages`	Get the list of folder pages. Used in a repeat instruction to loop on each page of the folder.
folder`/page[`*pname*`]`	Get the folder page.
folder`/page[`*pname*`]/text`	Get the text of the page.

Notes

1. *window* is any `window` access path
2. *folder* is any `folder` access path
3. *fname* is the name of the folder
4. *pname* is the page name

Items

Within the context of items, you can examine:

- Formfield path
- Label path
- Button path
- Image path
- Error path
- Message path

FormField path

Template Path	Description
`formfield[`*`fname`*`]` *`window`*`/formfield[`*`fname`*`]`	Access path to formfield elements.
`formfield``/id`	Get the id of the formfield.
`formfield``/active`	True if the formfield is active.
`formfield``/hidden`	True if the formfield is hidden.
`formfield``/tabindex`	Get the tabindex of the formfield.
`formfield``/value`	Get the value of the formfield.
`formfield``/selectedtext`	Get the displayed text of the currently selected item of a combobox or a radiogroup.

Notes

1. *window* is any `window` access path
2. *formfield* is any `formfield` access path
3. *fname* is the formfield name

Label path

Template Path	Description

`label[`*`lname`*`]` *`window`*`/label[`*`lname`*`]`	Access path to static labels.
`label``/text`	Get the label text.

Notes

1. *window* is any `window` access path
2. *label* is any `label` access path
3. *lname* is the label name

Button path

Template Path	Description
`button[`*`bname`*`]` *`window`*`/button[`*`bname`*`]`	Access path to button.
`button``/id`	Get the button identifier.
`button``/image`	Get the button image path.
`button``/text`	Get the button text.

Notes

1. *window* is any `window` access path
2. *button* is any `button` access path
3. *bname* is the button name

Image path

Template Path	Description
`image[`*`iname`*`]` *`window`*`/image[`*`iname`*`]`	Access path to static images.
`image``/id`	Get the image identifier.
`image``/height`	Get the image height.
`image``/width`	Get the image width.
`image``/path`	Get the image path.

image/stretch	Get the image stretch attribute.

Notes

1. *window* is any window access path
2. *image* is any image access path
3. *iname* is the image name

Error path

Template Path	Description
error application/error	Access path to error message text .

Message path

Template Path	Description
message *window*/message	Access path to message text.

Notes

1. *window* is any window access path

TopMenu path

Template Path	Description
topmenu *window*/topmenu	Access path to the current form topmenu.
application/topmenu	Access path to the application level topmenu.

Notes

1. *window* is any `window` access path

ToolBar path

Template Path	Description
`toolbar` `window/toolbar`	Access path to the current form toolbar.
`application/toolbar`	Access path to the application level toolbar.

Notes

1. *window* is any `window` access path

Other Paths

- Attribute path
- Repeat path

Attribute path

Template Path	Description
`attribute[aname]`	Access to the value of the attribute *aname* of the current tag.

Notes

1. If *aname* is not a valid HTML tag attribute name or if the attribute is not defined, the returned value is an empty string.

Repeat path

Template Path	Description
`repeat/elt/length`	Get the length of the repeat sequence.
`repeat/elt/index`	Get the index of the current repeat element.
`repeat/elt/first`	Test if the current repeat element is the first element of the sequence.
`repeat/elt/last`	Test if the current repeat element is the last element of the sequence.
`repeat/elt/even`	Test if the current repeat element is an even element of the sequence.
`repeat/elt/odd`	Test if the current repeat element is an odd element of the sequence

Notes

1. *elt* is a repeat instruction element.

Template JavaScript API Reference

GWC provides a JavaScript API for you to use when scripting GWC JavaScript functionality in your HTML application / page. This API provides easy access to application events like changing a table offset, changing the value of a form field, or emulating a key press.

Overview

This section provides an overview of the JavaScript API provided by the GWC as well as notes about incremental and smart field mode.

Event Handler

This section discusses the event handler. The event handler is defined with the object `gGWCEvent`.

Events

This section covers those objects that handle Action, Key, Field, Table, ScrollGrid and Matrix events.

Javascript API Overview

GWC provides a JavaScript API to ease the use of the GWC JavaScript functionalities. This API provides easy access to application events like changing a table offset, changing the value of a form field, or emulating a key press.

Notes

1. Incremental mode depends on the call of `gInitFieldMode(gFIELD_MODE | gSMART_FIELD_MODE)`
2. While Smart Field mode works, it is not managed by the JavaScript API.

Event Handler

The event handler is defined with the object `gGWCEvent`. It provides two methods: a method to add events and a method to send them to the Genero Web Client.

Syntax

```
gGWCEvent().method
```

Method	Description
`Append(event1, event2,..., eventN)`	Adds one or more events to the events list
`Send()`	Sends all the events in the events list to the GWC

Examples

```
<input type=button onclick="gGWCEvent().Append(event).Send()">
<input type=button onclick="gGWCEvent().Append(event1, event2).Send()">
<input type=button
onclick="gGWCEvent().Append(event1).Append(event2).Send()">
<input type=button
onclick="gGWCEvent().Append(event1).Send().Append(event2).Send()">
```

Events

The GWC JavaScript API provides several objects to handle Action, Key, Field, Table, ScrollGrid and Matrix events.

- Action Events
- Key Events
- Field Events
- Table Events
- ScrollGrid Events
- Matrix Events

Action Events

Action events are managed by the `gAction` object. It has no method, the action name is simply provided when creating the event.

Syntax

```
gAction(actionName)
```

Notes

1. *actionName* must be an allowed action when sending this event to the GWC

Example

```
<input type=button
onclick="gGWCEvent().Append(gAction('accept')).Send()">
```

Key Events

Key events emulates key presses and are handled by the `gKey` object. It has no method, the accelerator key name is simply provided when creating the event.

Syntax

```
gKey(keyName [, shiftKey [, ctrlKey [, altKey ] ] ])
```

Notes

1. *keyName* is the accelerator key name
2. *shiftKey*, *ctrlKey* and *altKey* are boolean values indicating if the Shift, Control or Alt keys were pressed

Example

```
<input type=button onclick="gGWCEvent().Append(gKey('Tab',
true)).Send()">
```

Field Events

The `gField` object handles the form fields events. Two events are defined, one to give the focus to the field and one to set its value.

Syntax

```
gField(fieldId).event
```

Event	Description
`Focus()`	Gives the focus to the field
`Value(value)`	Sets the current value of the field

Notes

1. *fieldId* is the field identifier given by the template path `formfield[fieldName]/id`

Example

```
<input type="button" gwc:attributes="onclick
string:gGWCEvent().Append(gField(${FormField[ordernumber]/id}).Value).S
end()">
```

Table Events

Table events are handled by the `gTable` object which provides several events.

Syntax

```
gTable(tableId).event
```

Event	Description
`Select(localIndex)`	Selects the row at *localIndex* of the current page
`Offset(offset)`	Sets the offset of the table
`PageSize(pageSize)`	Sets the pagesize of the table
`Sort(columnName)`	Sorts the table by the given column

Notes

1. *tableId* is the table identifier given by the template path `table[tableName]/id`

Example

```
<input type=button
onclick="gGWCEvent().Append(gTable('ordertable').Offset('56')).Send()">
```

ScrollGrid Events

The only event handled by the `gScrollGrid` object is the offset event.

Syntax

```
gScrollGrid(scrollGridId).event
```

Event	Description
`Offset(offset)`	Sets the current offset of the scrollgrid

Notes

1. *scrollGridId* is the table identifier given by the template path `scrollgrid[scrollgridName]/id`

Example

```
<input type=button
onclick="gGWCEvent().Append(gScrollGrid('ordergrid').Offset('89')).Send
()">
```

Matrix Events

The `gMatrix` object handles only one event.

Syntax

```
gMatrix(matrixId).event
```

Event	Description
`Select(localIndex)`	Selects the row at *localIndex* of the current page

Notes

1. *matrixId* is the matrix identifier given by the template path `scrollgrid[scrollgridName]/id`

Example

```
<input type=button
onclick="gGWEvent().Append(gMatrix('ordergrid').Select('4')).Send()">
```

Rendered HTML

This section describes the relationship between a 4GL statement and the generated HTML. While details about rendered HTML is subject to changes as features evolve, this section is provided to help you understand the default rendering and how it may be customized. The default rendering includes JavaScript that may change the generated HTML.

The elements are organized by category:

- Containers: HBOX, VBOX, GROUP, FOLDER, PAGE, GRID, SCROLLGRID, TABLE
- Items type : Formfield, TextEdit, Edit, Button, ButtonEdit, ComboBox, CheckBox, DateEdit, Image, Label, RadioGroup
- Others : Menu, Dialog, TopMenu, ToolBar, Message, Error

For each element, you are shown:

- The generated HTML, describing the HTML code built by the Genero Web Client engine (no changes).
- The default rendering, with the default CSS and JavaScript applied. CSS modifies the element look. JavaScript can change the HTML structure and the look of an element.
- The list of available CSS styles.
- The **class** attributes used by JavaScript to set the behaviors and states of widgets.

Containers

- HBOX
- VBOX
- GROUP
- FOLDER
- PAGE
- GRID
- SCROLLGRID
- TABLE

HBOX Container

In per file

```
01 HBOX ahbox
02   ...
03 END
```

Generated HTML

Syntax

```
<table class="gHBox" id="name">
  <tr>
    <td>...</td> [...]
  </tr>
</table>
```

Notes

1. *name* is the `HBOX` name in the per file.

Example

```
01 <table class="gHBox" id="ahbox">
02   <tr>
03     <td>
04       <div class="gGrid" id="agrid">
05       ...
06       </div>
07     </td>
08     ...
09   </tr>
10 </table>
```

Classes

- *gHbox*: main hbox class
- *gHidden*: when attribute HIDDEN=USER
- *user style*: set by attribute STYLE

VBOX Container

In per file

```
01 VBOX avbox
02   acontainer [...]
03 END
```

where *acontainer* can be any container (grid, hbox, ...).

Generated HTML

Syntax

```
<table class="gVBox" id="name">
  <tr><td class="gVBoxLine">...</td></tr> [...]
</table>
```

Notes

1. *name* is the `VBOX` name in the per file.

Example

```
01 <table class="gVBox" id="avbox">
02   <tr>
03     <td class="gVBoxLine">
04       <div class="gGrid" id="agrid">
05       ...
06       </div>
07     </td>
08   </tr>
09   ...
10 </table>
```

Classes

- for vbox
  - *gVbox*: main vbox class
  - *gHidden*: when attribute HIDDEN=USER
- for vbox content
  - *gVboxLine*: one line of vbox content (a grid container for example)
  - *user style*: set by attribute STYLE

GROUP Container

In per file

```
01 GROUP gp1 (TEXT="Group1")
02 GRID g1
03 {
04  ...
05 }
06 END
07 END
```

Generated HTML

Syntax

```
<span class="gGroupDiv" id="group_name-div">
  <fieldset class="gGroup" id="group_name">
    <legend class="gGroupTitle" id="group_name-
title">group_title</legend>
    ...
  </fieldset>
</span>
```

Notes

1. *group_name* is the `GROUP` name in the per file.
2. *group_title* is the **text** value of the `GROUP`

Example

```
01 <span class="gGroupDiv" id="gp1">
02   <fieldset class="gGroup" id="gp1">
03     <legend class="gGroupTitle" id="gp1-title">
04     Group1
05     </legend>
06     <div class="grid" id="g1">
07     ...
08     </div>
09   </fieldset>
10 </span>
```

Classes

- for group container
  - *gGroupDiv*: main group container class
  - *gHidden*: when attribute HIDDEN=USER
- for group content
  - *gGroup*: main group content class

    - *user style*: set by attribute STYLE
- for group title
    - *gGroupTitle*: main group title class

FOLDER Container

In per file

```
01 FOLDER f1
02   ...
03 END
```

Generated HTML

Syntax

```
<div class="gFolder" id="name">
  ...
</div>
```

Notes

1. *name* is the `FOLDER` name in the per file.

Example

```
01 <div class="gFolder" id="f1">
01 ...
03 </div>
```

Classes

- *gFolder*: main folder class
- *gHidden*: when attribute HIDDEN=USER
- *user style*: set by attribute STYLE

PAGE Container

In per file

```
01 PAGE p1 (TEXT="Page1")
02 ...
03 END
```

Generated HTML

Syntax

```
<span class="gPageHeader" id="page_name-header">
  <input class="gAction" type="submit" value="page_title">
</span>
<div class="gPage" id="page_name">
  ...
</div>
```

Notes

1. *page_name* is the `PAGE` name in the per file.
2. *page_title* is the text value of the `PAGE`
3. additional classes can be: selectedPageHeader,

Example

```
01 <span class="gPageHeader" id="p1-header">
02   <input class="gAction" type="submit" value="Page1">
03 </span>
04 <div class="gPage" id="p1">
05   ...
06 </div>
```

Default rendering

HTML	Without Rendering	Default Rendering
none	page 1 Here a text page 2	page 1 page 2 Here a text

with JavaScript rendering:

```
01 <SPAN class="gPageHeader gSelectedPageHeader" id=p1-header
_actionName="undefined">
```

```
02 <LABEL>page 1</LABEL>
03 <INPUT class=gAction type=submit value="page 1">
04 </SPAN>
05 <DIV class="gPage gSelectedPage" id=p1>
06 ...
07 </DIV>
```

The default rendering adds a label and hides the input button. The effect is to see a clickable text for selecting a folder page. On some user agents, a disabled input button cannot receive the focus. You cannot click on a button to go into a folder page.

For more details see the FolderWrapper in $FGLASDIR/fjs/defaultTheme/genero.js

Classes

- for page header
  - *gPageHeader*: main page header class
  - *gHidden*: when attribute HIDDEN=USER
- for page
  - *gPage*: main page class
  - *user style*: set by attribute STYLE

GRID Container

In per file

```
01 GRID g1
02 {
03  [f001           ]
04 ...
05 }
06 END
```

Generated HTML

Syntax

```
<span class="gGrid" id="grid_name">
  aline
</span>
```

where *aline* is, generated for each line in the grid:

```
<div class="gGridLine">
  ...
</div>
```

Notes

1. *grid_name* is the `GRID` name in the per file.

Example

```
01 <span class="gGrid" id="g1">
02   <div class="gGridLine">
03     <span class="gFormFieldBox" id="field1-span">
04       <input class="gEdit" id="field1" type="text">
05     </span>
06   </div>
07 <span>
```

Classes

- for the grid
  - *gGrid*: main grid container class
  - *gHidden*: when attribute HIDDEN=USER
- for each grid line
  - *gGridLine*

SCROLLGRID Container

In per file

```
01 SCROLLGRID s1
02 {
03   ...
04 }
05 END
```

Generated HTML

Syntax

```
<span class="gScrollGridBox" id="sgname-div">
  <span class="gScrollGrid" id="sgname">
    ...
    <span class=gHLineBox id=gline-span><hr id="gline"></span> [...]
  </span>
```

```
  <input type=hidden name=gOffset value="offset">
  <input type=hidden name=gPageSize value="pageSize">
  <input type=hidden name=gSize value="asize">
</span>
```

Notes

1. *sgname* is the scrollgrid identifier
2. *gline* is a separator of the record pattern
3. *offset* is the record position in the array
4. *pSize* is the number of lines in the per line
5. *asize* is the array size

Example

```
01 <span class="gScrollGridBox" id="s1-div">
02   <span class="gScrollGrid" id="s1">
03    ...
04   </span>
05 </span>
```

Classes

- for scrollgrid container
  - *gScrollGridBox*: main scrollgrid container class
  - *gHidden*: when attribute HIDDEN=USER
  - *user style*: set by attribute STYLE
- for scrollgrid content
  - *gScrollGrid*: main scrollgrid class
  - *gEnabledScroll*: if scrolling is available
- for scrollgrid separator
  - *gHLineBox*: separator container class

TABLE Container

In per file

```
01 <T t0         >
02   D1  D2  D3
03  [s1 |s2 |s3 ]
```

Generated HTML

Syntax

```
<span class="gTableBox" id="tname-div">
<table class="gTable" id="tname">
  tableSelect [...]
  <thead>
    <tr>
      tableTitle [...]
    </tr>
  </thead>
  <tr>
    tableColumn [...]
  </tr>
</table>
<input type=hidden name=gOffset value="offset">
<input type=hidden name=gPageSize value="pageSize">
<input type=hidden name=gSize value="asize">
</span>
```

where tableSelect is:

```
<col class="gColSelect" id="cname-col">
```

and tableTitle is:

```
<th>
<input class="gTableHeader" id="cname" name="idref" type="submit"
value="ctitle">
</th>
```

and tableColumn is:

```
<td>
  afield
</td>
```

Notes

1. *rowstate* can be currentRow if this row is the current one in a display array
2. *cname* is the formfield identifier
3. *idref* is the node id in the AUI tree
4. *ctitle* is the column title
5. *afield* is any field representation
6. *offset* is the record position in the array
7. *pSize* is the number of lines in the per line
8. *asize* is the array size
9. A column is added to handle row selection :
   in tableSelect:

```
<col class="gColSelect" id="cname-col-select">
```

in tableColumn:

```
<TD><INPUT class="gTableSelect" type="radio" value="row"
name="tableid"></TD>
```

Example

```
01 <span class="gTableBox" id="t0-div">
02   <table class="gTable" id="t0">
03     ...
04     <col id="s1-col">
05    ...
06   <thead>
07     <tr>
08       <th>
09            <input class="gTableHeader" id="s1" name="90"
type="submit" value="D1">
10        </th>
11        ...
12      </tr>
13    </thead>
14    <tbody>
15      <tr>
16        ...
17        <td>
18            <input class="edit" name="112" size="10" type="text"
value="">
19        </td>
20        ...
21      </tr>
22      ...
23    </tbody>
24   </table>
25 </span>
```

Classes

- for table container
  - *gTableBox*: main table container class
  - *gHidden*: when attribute HIDDEN=USER
- for table
  - *gTable*: main table class
  - *gEnabledScroll*: when scrollbar is available
  - *gCurrentTable*: if is the current table
  - *user style*: set by attribute STYLE
- for table column
  - *gTableHeader*: default style for a table header
  - *gSortAsc*: if the column is in ascendant sort
  - *gSortDesc*: if the column is in descendant sort

    - *gHidden*: when HIDDEN=USER (set on <col> element)
- for table row
    - *gCurrentRow*: if the row is the current one (set on <tr> element)
- for scrollbar
    - *gOffset*
    - *gSize*
    - *gPageSize*

Items type

- Formfield
- TextEdit
- Edit
- Button
- ButtonEdit
- ComboBox
- CheckBox
- DateEdit
- Image
- Label
- RadioGroup

FormField

Generated HTML

Syntax

```
<span class="gFormFieldBox" id="fname-span">
  afield
</span>
```

Notes

1. *fname* is the formfield identifier
2. *afield* is the formfield representation

This element is used to set the field position. It becomes useless when this field is in a TABLE. Its position is determined by the table cell <td>.

Classes

- *gFormField*: field container main class

TEXTEDIT

TextEdit, the multi-line edit field, is represented as a textarea.

In per file

```
01 TEXTEDIT f01 = formonly.text;
```

Generated HTML

Syntax

```
<textarea class="gTextEdit" id="txtName">text</textarea>
```

Notes

1. *txtName* is the name of the formfield TextEdit in the per file
2. *text* is the value of the formfield TextEdit

Example

```
01 <textarea class="gTextEdit" id="text">Here is a text</textarea>
```

Classes

- *gTextEdit*: textEdit main class
- *user style*: set by attribute STYLE
- *gNotNull*: set by attribute NOT NULL
- *gRequired*: set by attribute REQUIRED
- *gVerify*: set by attribute VERIFY
- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *g<field_trigger>*: set by input triggers (gOnChange, ...)
- *gAutoNext*: set by attribute AUTONEXT
- *gShiftUp*: set by attribute UPSHIFT
- *gShiftDown*: set by attribute DOWNSHIFT
- *gScrollbarHorizontal*: set by attribute SCROLLBARS=HORIZONTAL

- *gScrollbarVertical*: set by attribute SCROLLBARS=VERTICAL
- *gStretchX*: set by attribute STRETCH=X
- *gStretchY*: set by attribute STRETCH=Y
- *gWantTabs*: set by attribute WANTTABS
- *gWantReturns*: set if attribute WANTNORETURNS not set

EDIT Item Type

In per file

```
01 EDIT f01 = formonly.state;
```

Generated HTML

Syntax

```
<input class="gEdit" id="ename" type="text" value="text">
```

Notes

1. *ename* is the formfield identifier
2. *text* is the formfield value

Example

```
01  <input class="gEdit" id="state" type="text" value="Description">
```

Classes

- *gEdit*: edit main class
- *user style*: set by attribute STYLE
- *gNotNull*: set by attribute NOT NULL
- *gRequired*: set by attribute REQUIRED
- *gVerify*: set by attribute VERIFY
- *gInclude*: set by attribute INCLUDE
- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *gJustifyLeft*: set by attribute JUSTIFY=LEFT
- *gJustifyRight*: set by attribute JUSTIFY=RIGHT
- *gJustifyCenter*: set by attribute JUSTIFY=CENTER
- *g<field_trigger>*: set by input triggers (gOnChange, ...)
- *gAutoNext*: set by attribute AUTONEXT

- *gShiftUp*: set by attribute UPSHIFT
- *gShiftDown*: set by attribute DOWNSHIFT

BUTTON Item Type

In per file

```
01 BUTTON f01 : print, TEXT="Print Report", IMAGE="printer";
```

Generated HTML

Syntax

```
<span class="gButtonBox" id="idref-span">
[
<input class="gAction" name="bname" src="img" title="btitle"
type="image">
]
<input class="gAction" name="bname" title="btitle" type="submit"
value="btext">
</span>
```

Notes

1. *idref* is the node id in the AUI tree
2. *bname* is the formfield identifier
3. *img* is the path to the buttonedit image
4. *btitle* is the button comment
5. *btext* is the value of the TEXT attribute in per file
6. button does not necessarily have an image

Example

```
01 <span class="gButtonBox" id="idRef90-span">
02  <input alt="print" class="gAction" name="print"
src="/pic/printer.png" title="Print" type="image">
03  <input class="gAction" name="print" title="Print" type="submit"
value="Print Report">
04 </span>
```

Classes

- for button container
  - *gButtonBox*: button container main class

    - *gHidden*: when attribute HIDDEN=USER
- for button and image
    - *gAction*: main class

BUTTONEDIT Item Type

In per file

```
01 BUTTONEDIT f01 = FORMONLY.f01, IMAGE="smiley";
```

Generated HTML

Syntax

```
<input class="gButtonEdit" id="fname" name="idref" type="text">
<input class="gAction" src="img" type="image">
```

Notes

1. *fname* is the formfield identifier
2. *idref* is the node id in the AUI tree
3. *img* is the path to the buttonedit image

Example

```
01 <span class="gFormFieldBox" id="f01-span">
02  <input class="gButtonEdit" id="f01" name="88" size="18"
type="text">
03  <input class="gAction" src="/pic/smiley.png" type="image">
04 </span>
```

Classes

- *gButtonEdit*: buttonedit main class
- *user style*: set by attribute STYLE
- *gNotNull*: set by attribute NOT NULL
- *gRequired*: set by attribute REQUIRED
- *gVerify*: set by attribute VERIFY
- *gInclude*: set by attribute INCLUDE
- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *gJustifyLeft*: set by attribute JUSTIFY=LEFT
- *gJustifyRight*: set by attribute JUSTIFY=RIGHT

- *gJustifyCenter*: set by attribute JUSTIFY=CENTER
- *g<field_trigger>*: set by input triggers (gOnChange, ...)
- *gAutoNext*: set by attribute AUTONEXT
- *gShiftUp*: set by attribute UPSHIFT
- *gShiftDown*: set by attribute DOWNSHIFT
- *gAction*: image main class

COMBOBOX Item Type

In per file

```
01 COMBOBOX f01 = formonly.city,
ITEMS=((1,"Paris"),(2,"Madrid"),(3,"London")), DEFAULT=2;
```

Generated HTML

Syntax

```
<select class="gComboBox" id="fname" name="idref">
    opt [...]
</select>
```

where opt is:

```
<option value="item_val">
  item_text
</option>
```

Notes

1. *fname* is the formfield identifier
2. *idref* is the node id in the AUI tree
3. *item_val* is the real value of a combobox item
4. *item_text* is the displayed text of a combobox item

Example

```
01 <span class="gFormFieldBox" id="city-span">
02   <select class="gComboBox gCurrentField gTypeString" id="city"
name="89">
03     <option value="">
04     </option>
05     <option value="1">
06    Paris
```

```
07    </option>
08     <option selected="selected" value="2">
09     Madrid
10     </option>
11     <option value="3">
12     London
13     </option>
14   </select>
15 </span>
```

Default rendering

In a Construct dialog, with the attribute QUERYEDITABLE, the user can enter a value in the combobox.

GDC	Without Rendering	Default Rendering
		

in per file:

```
01 combobox f03 = formonly.comp, QUERYEDITABLE,
02   items=(("Century Pro Shop","Century"),("Bay
Sports","Sports"),("Kids Korner","Kids"));
```

with default rendering:

```
01 <SELECT class="gComboBox gQuery " id=comp tabIndex=3 multiple size=1
name=91>
02 <OPTION value=""></OPTION>
03 <OPTION value="Century Pro Shop">Century</OPTION>
04 <OPTION value="Bay Sports">Sports</OPTION>
05 <OPTION value="Kids Korner" selected>Kids</OPTION>
06 </SELECT>
07 <SPAN class=queryZone>
08 <INPUT class=" " style="POSITION: relative" value="Kids
Korner|Century">
09 </SPAN>
```

An HTML combobox is not editable. To simulate QUERYEDITABLE, an input field beside the combobox is added by JavaScript. The user can then enter any value in this input field.

An HTML combobox allows multi selection, so you can select several values for the combobox with the shift or control key.

Classes

- *gComboBox*: combobox main class
- *user style*: set by attribute STYLE
- *gNotNull*: set by attribute NOT NULL
- *gRequired*: set by attribute REQUIRED
- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *g<field_trigger>*: set by input triggers (gOnChange, ...)

CHECKBOX Item Type

In per file

```
01 CHECKBOX f01 = formonly.check, TEXT="Active", VALUECHECKED="Y",
VALUEUNCHECKED="N", DEFAULT="Y";
```

Generated HTML

Syntax

```
<select class="gCheckBox" id="cname" name="idref">
  opt [...]
</select>
<label for="cname">
ctext
</label>
```

where option is:

```
<option value="item_val">
item_val
</option>
```

Notes

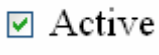
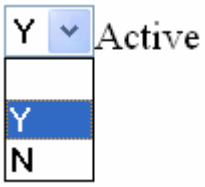
1. *cname* is the formfield identifier
2. *idref* is the node id in the AUI tree
3. *ctext* is the TEXT attribute of the checkbox
4. *item_val* is the checked or unchecked value for the checkbox. A null value is added if the current selected value is null

Example

```
01   <select class="gCheckBox gCurrentField gTypeChar" id="check"
name="91">
02     <option value="">
03     </option>
04     <option selected="selected" value="Y">
05       Y
06     </option>
07     <option value="N">
08       N
09     </option>
10   </select>
11   <label for="check">
12     Active
13   </label>
```

Default Rendering

There is no checkbox with 3 states (null, checked, unchecked) in HTML. The adopted solution is to use a combobox reshaped in checkbox.

HTML	Without Rendering	Default Rendering
		

Example

```
01 <INPUT class="checkBox currentField typeChar checkedState" id=91>
```

The combobox is replaced by an input . It contains a background image that simulates the checkbox states: cross (null), v (checked) or nothing (unchecked). Events handlers like click and double-click are added to this input to allow the states changes.

For more details see the CheckboxWrapper function in $FGLASDIR/web/defaultTheme/genero.js file.

Classes

- *gCheckBox*: checkbox main class
- *user style*: set by attribute STYLE
- *gNotNull*: set by attribute NOT NULL
- *gRequired*: set by attribute REQUIRED

- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *g<field_trigger>*: set by input triggers (gOnChange, ...)
- *gNullState*: if the checkbox has a NULL value
- *gCheckedState*: if the checkbox is checked
- *gUncheckedState*: if the checkbox is unchecked

.

DATEEDIT Item Type

In per file

```
01 DATEEDIT f01 = formonly.dte;
```

Generated HTML

Syntax

```
<input class="gDateEdit" id="dname" name="idref" type="text"
value="val">
</input>
```

Notes

1. *dname* is the formfield identifier
2. *idref* is the node id in the AUI tree
3. *val* is the date time value

Example

```
01 <span class="gFormFieldBox" id="dte-span">
02  <input class="gDateEdit gCurrentField" id="dte" name="92"
type="text">
03 </span>
```

Default rendering

The DATEEDIT widget is a date picker.

HTML	Without Rendering	Default Rendering
none	dateedit	

With default rendering, a calendar image is added to each dateedit in input state. Clicking this image creates a calendar table if it does not already exist. The calendar interaction is handled by the JavaScript function "Calendar" in genero.js. The look is set by "calendar" classes in genero.css.

For more details see the DateeditWrapper function in $FGLASDIR/web/defaultTheme/genero.js file.

Classes

- *gDateEdit*: dateedit main class
- *user style*: set by attribute STYLE
- *gRequired*: set by attribute REQUIRED
- *gInclude*: set by attribute INCLUDE
- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *gJustifyLeft*: set by attribute JUSTIFY=LEFT
- *gJustifyRight*: set by attribute JUSTIFY=RIGHT
- *gJustifyCenter*: set by attribute JUSTIFY=CENTER
- *g<field_trigger>*: set by input triggers (gOnChange, ...)

IMAGE Item Type

In per file

```
01 IMAGE f01 = formonly.f01, PIXELHEIGHT=300, PIXELWIDTH=400,
STRETCH=BOTH;
02 IMAGE img : logo, IMAGE="smiley";
```

Generated HTML

Syntax

```
<span class="tclass" id="iname-span">
  <img class="gImage" id="iname" src="img">
</span>
```

Notes

1. *iname* is the image identifier
2. *idref* is the node id in the AUI tree
3. *img* is the path to the image
4. *tclass* illustrates the images type, a formfield or a static image

Example

```
01 <span class="gFormFieldBox" id="f01-span">
02  <img class="gImage" id="f01" src="/pic/smiley.png">
03 </span>
04 ...
05 <span class="gImageBox" id="logo-span">
06  <img class="gImage" id="logo" src="/pic/smiley.png">
07 </span>
```

Classes

- for static image container
  - *gImageBox*: image container main class
  - *gHidden*: when attribute HIDDEN=USER
- for image
  - *gImage*: main image class
  - *gStretchX*: set by attribute PIXELWIDTH
  - *gStretchY*: set by attribute PIXELHEIGHT
  - *gAutoscale*: set by attribute AUTOSCALE
  - *user style*: set by attribute STYLE

LABEL Item Type

In per file

```
01 LABEL f01 = formonly.desc;
02 LABEL lab : label1, TEXT="Hello";
```

Generated HTML

Syntax

```
<label class="gLabel" id="lname">
  text
</label>
```

Notes

1. *lname* is the label identifier
2. *text* is the TEXT attribute of a static label or the value of the formfield

Example

```
01 <span class="gFormFieldBox" id="desc-span">
02   <label class="gLabel" id="desc">
03    Description
04   </label>
05 </span>
06 ...
07 <span class="gLabelBox" id="label1-span">
08   <label id="label1">
09     Hello
10   </label>
11 </span>
```

Classes

- for static label
  - *gLabelBox*: label container main class
  - *gHidden*: when attribute HIDDEN=USER
- for formfield label
  - *gLabel*: main formfield label class
- for label
  - *user style*: set by attribute STYLE
  - *gHidden*: when attribute HIDDEN=USER
  - *gJustifyLeft*: when attribute JUSTIFY=LEFT
  - *gJustifyRight*: when attribute JUSTIFY=RIGHT
  - *gJustifyCenter*: when attribute JUSTIFY=CENTER

RADIOGROUP Item Type

In per file

```
01 RADIOGROUP f01 = formonly.f01,
ITEMS=((1,"Beginner"),(2,"Normal"),(3,"Expert")),DEFAULT="3";
```

Generated HTML

Syntax

```
<select class="gRadioGroup" id="rname" name="idref">
  opt [...]
</select>
```

where opt is:

```
<option value="item_val">
  item_text
</option>
```

Notes

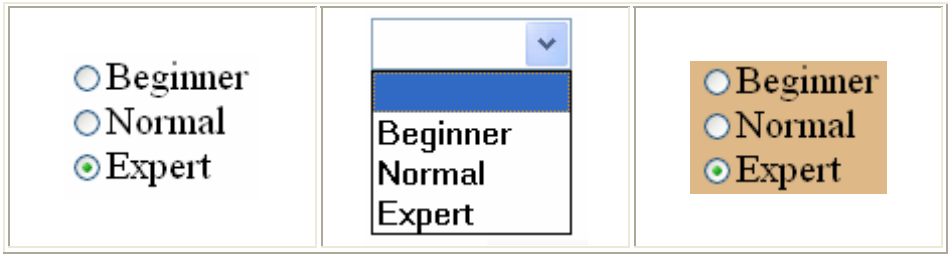
1. *rname* is the radiogroup identifier
2. *item_val* is the real value of the radiogroup
3. *item_text* is the displayed text of the radiogroup

Example

```
01 <span class="gFormFieldBox" id="f01-span">
02   <select class="gRadioGroup" id="f01" name="95">
03     <option value="">
04     </option>
05     <option value="1">
06       Beginner
07     </option>
08     <option value="2">
09       Normal
10     </option>
11     <option selected="selected" value="3">
12       Expert
13     </option>
14   </select>
15 </span>
```

Default rendering

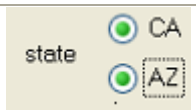
HTML	Without Rendering	Default Rendering




```
01 <SPAN class="radioGroup currentField typeString" title="">
02   <DIV><INPUT tabIndex=-1 type=radio value=1>Beginner</DIV>
03   <DIV><INPUT tabIndex=-1 type=radio value=2>Normal</DIV>
04   <DIV><INPUT tabIndex=-1 type=radio CHECKED value=3>Expert</DIV>
05 </SPAN>
```

The combobox is reshaped to radio group buttons.

In a construct dialog, radiogroup has multiple choices.

GDC	Without Rendering	Default Rendering
		

In per file

```
01 radiogroup f06 = formonly.stat, items=(("CA","CA"),("AZ","AZ")) ;
```

With default rendering

```
01 <INPUT tabIndex=-1 type=radio CHECKED value=CA>CA
02 <INPUT tabIndex=-1 type=radio CHECKED value=AZ>AZ
03 <SPAN class=gQueryZone>
04   <INPUT class=" current" style="POSITION: relative">
05 </SPAN>
```

The generated combobox is reshaped to radio buttons. Beside the radio buttons an input field is added. The user can enter a request which is not limited to radio button choices.

Classes

- *gRadioGroup*: radiogroup main classs
- *user style*: set by attribute STYLE
- *gNotNull*: set by attribute NOT NULL
- *gRequired*: set by attribute REQUIRED
- *gInclude*: set by attribute INCLUDE
- *gQuery*: in construct
- *gNoEntry*: set by attribute NOENTRY
- *g<field_trigger>*: set by input triggers (gOnChange, ...)

- *gOrientationHorizontal*: set by attribute ORIENTATION=HORIZONTAL
- *gOrientationVertical*: set by attribute ORIENTATION=VERTICAL

Others

- Menu
- Dialog
- TopMenu
- ToolBar
- Message
- Error

Menu

In 4gl file

```
01 MENU "Main"
02   ATTRIBUTE (image="smiley")
03   ON ACTION act
04   ...
05 END MENU
```

where in default.4ad, you have:

```
01 <ActionDefault name="act" text="The menu action" image="smiley"/>
```

Generated HTML

Syntax

```
<DIV class="gMenu">
  <IMG src="mImg">
  <SPAN class=gMenuTitle>mTitle</SPAN>
  <UL>
    act [...]
  </UL>
</DIV>
```

where act is:

```
<LI class="gMenuAction">
  [ <INPUT class=gAction type=image src="img" name=actName> ]
  <INPUT class=gAction type=submit value="actText" name=actName>
```

Notes

1. *mTitle* is the menu title
2. *mImg* is the menu image
3. *img* is image value
4. *actName* is the action name
5. *actText* is the action text

Example

```
01 <DIV class="gMenu" title="Menu comment">
02   <IMG src="/pic/ssmiley">
03   <SPAN class=gMenuTitle>Main</SPAN>
04   <UL>
05     <LI class="gMenuAction"><INPUT class=gAction type=image
src="/pic/smiley" name=act>
06         <INPUT class=gAction type=submit value="The menu action"
name=act>
07     ...
08    </UL>
09 </DIV>
```

Classes

- menu container
  - *gMenu*: menu container main class
  - *gStyle<user style>*: set by attribute STYLE
  - *gStyleDefault*: set by attribute STYLE="default"
  - *gStyleDialog*: set by attribute STYLE="dialog"
  - *gStylePopup*: set by attribute STYLE="popup"
- menu title
  - *gMenuTitle*: menu title main class
- menu action
  - *gAction*: action and command main class
  - *gHidden*: when attribute HIDDEN=USER
  - *gCurrentAction*: if the action is the current one

Dialog

Generated HTML

Syntax

```
<DIV class=gDialog>
  <UL>
    <LI class=gDialogAction><INPUT class=gAction type=submit
value=actText name=actName> [...]
  </UL>
</DIV>
```

Notes

1. *actText* is the action text
2. *actName* is the action name

Example

```
01 <DIV class=gDialog>
02   <UL>
03     <LI class=gDialogAction><INPUT class=gAction type=submit
value=OK name=accept>
04       <LI class=gDialogAction><INPUT class=gAction type=submit
value=Cancel name=cancel>
05   </UL>
06 </DIV>
```

Classes

- for dialog container
  - *gDialog*: dialog container main class
- for action container
  - *gDialogAction*: action container class
  - *gHidden*: when attribute HIDDEN=USER
- for action
  - *gAction*: action main class

TopMenu

In .per file

```
01 TOPMENU
02   GROUP (TEXT="Form")
03     COMMAND help (TEXT="Help", IMAGE="quest")
04     SEPARATOR (TAG="tm_separator")
05     COMMAND quit (TEXT="Quit")
06   END
07 END
```

Generated HTML

Syntax

```
<DIV class=gTopMenu>
  tGroup [...]
</DIV>
```

where tGroup is:

```
<UL>
  <LI><LABEL>gTitle</LABEL>
  <UL>
    tCommand | tGroup | tSeparator [...]
  </UL>
</UL>
```

where *tCommand* is:

```
<LI>
  <INPUT class=gAction type=image src=cImg name=cName>
  <INPUT class=gAction type=submit value=cTitle name=cName>
```

where *tSeparator* is:

```
<LI><HR>
```

Notes

1. *cImg* is the command image
2. *cName* is the command identifier
3. *cTitle* is the command text

Example

```
01 <DIV class=gTopMenu>
02   <UL>
03     <LI><LABEL>Form</LABEL>
04       <UL>
05         <LI><INPUT class=gAction type=image alt=help
src="/pic/quest" name=help>
06               <INPUT class=gAction type=submit value=Help name=help>
07         <LI><HR>
08       <LI><INPUT class=gAction type=submit value=Quit name=quit>
09     </UL>
10   </UL>
11 </DIV>
```

Classes

- *gTopMenu*: topmenu container main class
- *gAction*: topmenu command main class

ToolBar

In .per file

```
01 TOOLBAR
02   ITEM action1 (TEXT="Action1", IMAGE="accept")
03   SEPARATOR
04   ITEM quit (TEXT="Exit", image="quit")
05 END
```

Generated HTML

Syntax

```
<DIV class=gToolBar>
  <UL>
    tItem | tSeparator [...]
  </UL>
</DIV>
```

where *tItem* is:

```
<LI>
  <INPUT class=gAction type=image src=actImg name=actName>
  <INPUT class=gAction type=submit value=actTitle name=actName>
```

where *tSeparator* is:

```
<HR>
```

Notes

1. *actImg* is the action image
2. *actTitle* is the action title
3. *actName* is the action identifier

Example

```
01 <DIV class=gToolBar>
02   <UL>
03     <LI><INPUT class=gAction type=image src="/pic/accept"
name=action1>
04     <INPUT class=gAction type=submit value=Action1 name=action1>
05     <LI><HR>
06     <LI><INPUT class=gAction type=image src="/pic/quit" name=quit>
07     <INPUT class=gAction value=Exit name=quit>
08   </UL>
09 </DIV>
```

Classes

- *gToolBar*: toolbar container main class
- *gAction*: toolbar action main class

Message

In 4gl file

```
01 MESSAGE "This is a message"
```

Generated HTML

Syntax

```
<P class=gMessage id=gMessage>text</P>
```

Notes

1. *text* is the message value

Example

```
01 <P class=gMessage id=gMessage>This is a message</P>
```

Classes

- *gMessage*: main message class

Error

In 4gl file

```
01 ERROR "This is an error"
```

Generated HTML

Syntax

```
<P class=gError id=gError>text</P>
```

Notes

1. *text* is the error value

Example

```
01 <P class=gError id=gError>This is an error</P>
```

Classes

- *gError*: main error class

Template Tutorial

This tutorial explains how to customize the rendering of an application by using CSS and Genero Web Client instructions and template paths.

Tutorial overview

4GL and Web applications do not have the same widgets, however Genero Web Client takes advantage of the best of both of these worlds. You can choose how to represent a 4GL widget in a Web application and how a Web widget reacts. You control the design and behavior of a widget. For example, an array can be displayed the same as would using the Genero Desktop Client, or it can be presented in a more web-like fashion with links.

This tutorial is based on a simple application that manages identity cards. Using this application, a user can add a new card, list existing cards, enter search criteria and search for one or more cards, or delete a card. The demo application does not use a database; the card data is saved in a XML file.

The files supporting this tutorial:

- The tutorial application source files are located in the **$FGLASDIR/demo/tutorial/src** directory
- The configuration files are located in the **$FGLASDIR/demo/tutorial/app** directory.
- The HTML files and image files are located in the **$FGLASDIR/demo/tutorial/web** directory.

To view the application for a specific tutorial step, go to the demos page and click on the link for the tutorial step that interests you. For information on accessing the demos page, refer to the section Quick Start >> Launch Demos.

Reference for the Application Server configuration and to the Genero Web Client template Language can be found in the Reference section of this manual.
Reference for the Application Deployment can be found in the Miscellaneous section of this manual.

Tutorial Topic Summary

The following topics are covered in this tutorial:

- How to configure an application
  - Where to find configuration files
  - How to access a database
  - How to use images
  - How to use arguments
  - How to configure a connector
- How to customize an application
  - Using CSS
  - Using templates
  - Using Javascript API

Step 0: Using the built-in rendering

Built-in rendering is done when an application is started without using a custom template. When a custom template is not provided, the Genero Web Client uses the default HTML template **generodefault.html**.

In order to start an application, it must be defined for the Genero Web Client. Information for accessing the application can be specified in the application server configuration file (default **as.xcf**) or in a separate, XML-based, application-specific configuration file.

Note: All application configuration files used in this tutorial are found in the *demo/tutorial/app* directory of the Genero Web Client.

The following XML provides the information needed to access the identity card management application.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <APPLICATION Id="tutorial" Parent="demo-tut-abstract">
03   <EXECUTION>
04     <PATH>$(res.path.demo.dem-tut)/src</PATH>
05     <MODULE>card.42r</MODULE>
06   </EXECUTION>
07 </APPLICATION>
```

Notes

1. Line 2: The `APPLICATION` tag specifies the Id of the application and tells Genero Web Client that this application inherits the configuration settings of the parent application *demo-tut-abstract*.
2. Line 4: In the `EXECUTION` tag, the `PATH` tag defines the application's module directory.
3. Line 5: In the `EXECUTION` tag, the `MODULE` tag specifies the main module of the application.

In this example, the parent application is an *abstract application*, defined in the **as.xcf** file:

```
01 <APPLICATION Id="demo-tut-abstract" Parent="defaultgwc"
Abstract="TRUE">
02   <RESOURCE Id="res.path.demo.dem-tut"
Source="INTERNAL">$(res.path.as.demo)/tutorial</RESOURCE>
03   <PICTURE>
04     <PATH>$(connector.uri)/tutorial/img</PATH>
05   </PICTURE>
06 </APPLICATION>
```

Notes

1. Line 1: This abstract application inherits the configuration of the default Genero Web Client application.
2. Line 2: This abstract application defines a internal `RESOURCE` pointing to the tutorial directory.
3. Line 3-5: The `PICTURE` path tells the web server where to look for images.

Some aliases are added to the `INTERFACE_TO_CONNECTOR` tag:

```
01 <ALIAS
Id="/tutorial/img">$(res.path.as.demo)/tutorial/web/tutorial/img</ALIAS
>
02 <ALIAS
Id="/tutorial/inc">$(res.path.as.demo)/tutorial/web/tutorial/inc</ALIAS
>
```

Finally, a `GROUP` is defined, identifying the directory that contains the XML-based, application-specific configuration file:

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

With this information entered into the configuration file, the application is ready to be launched. The application can be accessed using the following syntax:

```
http://server:port/wa/r/group/application
```

If you are logged on to the same machine on which the Genero Web Client is installed, the URL will be:

```
http://localhost:6394/wa/r/tut-demo/tutorial
```

Step 1: Customize the rendering

In this step, the rendering of the application is customized by adding a custom CSS file and modifying the default template.

To modify the default template, start by making a copy of the **generodefault.html** file (located in the Genero Web Client template directory) and rename the copy.

Next, open the new file and make any changes that are necessary.

Example **tutorialStep1.html** in the directory *demo/tutorial/web/tutorial*:

```
01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
03   <head>
04     $(res.meta-tags)
05     <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
06     <title gwc:content="string:${window/text}">Template page</title>
07     <script language=javascript
src="$(connector.uri)/fjs/uaapi/webBrowser.js"></script>
08     <script language=javascript
src="$(connector.uri)/fjs/asapi/application.js"></script>
09     <script language=javascript
src="$(connector.uri)/fjs/asapi/wrappers.js"></script>
10     <link rel="stylesheet"
href="$(connector.uri)/fjs/defaultTheme/genero.css" type="text/css" />
11     <script language=javascript
src="$(connector.uri)/fjs/defaultTheme/genero.js"></script>
12     <link rel="stylesheet"
href="$(connector.uri)/tutorial/inc/tutorialStep1.css" type="text/css"
/>
13     <style type="text/css" >
14       #gForm { margin-right: 20px; height:auto; width:auto; }
```

```
15       #gFormTable { width: 100%; table-layout: fixed; }
16     </style>
17   </head>
18   <body>
19     <form action="..." id="gDialogForm" method=post
gwc:attributes="action document/URL">
20       <div gwc:replace="application/intermediatetrigger" />
21       <div gwc:replace="application/dbdate" />
22       <table width="100%" height="100%" border="0" cellpadding="0"
cellspacing="0">
23         <tr>
24           <td colspan=2>
25             <div gwc:condition="application/topmenu"
gwc:replace="application/topmenu" /><div gwc:condition="topmenu"
gwc:replace="topmenu" />
26             <div gwc:condition="application/toolbar"
gwc:replace="application/toolbar" /><div gwc:condition="toolbar"
gwc:replace="toolbar" />
27           </td>
28         </tr>
29         <tr height="100%">
30           <td width="100%" style="vertical-align:top;"><table
gwc:condition="form" id="gFormTable"><tr><td><div id="gForm-div"
gwc:content="form" /></td></tr></table></td>
31           <td valign="top" id="gPanel"><div gwc:condition="menu"
gwc:replace="menu" /><div gwc:condition="dialog" gwc:replace="dialog"
/></td>
32         </tr>
33         <tr>
34       <td colspan=2 id="gInfo"><div gwc:condition="message"
gwc:content="message" /><div gwc:condition="error" gwc:content="error"
/></td>
35         </tr>
36       </table>
37     </form>
38     <span class="footer">&copy;2004 Four J's Development
Tools</span>
39     <script defer language=javascript><!--
40       var gLayoutData = ${document/layoutData};
41       gInitFieldMode( gIdToElement('gDialogForm'),
gSMART_FIELD_MODE, gINCREMENTAL_MODE ); // [ gFIELD_MODE |
gSMART_FIELD_MODE ], [ gINCREMENTAL_MODE | gFULL_MODE ]
42     //--></script>
43   </body>
44 </html>
```

Notes

1. Line 12: A link is added to the custom style sheet *tutorialStep1.css*. This CSS file overrides some of the genero.css styles and defines a new style for the footer of the page.
2. Line 19-37: These lines render the entire application window. The `FORM` tag is used to submit data to the Genero Web Client. This form must have the id *gDialogForm*. Use the `gwc:attributes` instruction to replace the value of the

action attribute of the tag with the correct URL using the `document/url` template path.

3. Line 21: A *gDBDate* `INPUT` is defined, since the form contains a calendar widget.
4. Line 26: The template path `application/toolbar` is used to render the application level toolbar.
5. Line 38: A footer is added to the page with corporate information.

The contents of the CSS file:

```
01 .gToolBar {
02   background: #BBBBEE;
03   border-bottom-color: gold;
04 }
05 .gToolBar LI {
06   border-color: white;
07 }
08 .gToolBar .hover {
09   background-color: gold;
10 }
11 .gToolBar .hover * {
12   color: blue;
13 }
14 .gToolBar .pressed {
15   background-color: gold;
16 }
17 .footer {
18   font-size: x-small;
19   position: absolute;
20   bottom: 0px;
21 }
22 BODY {
23   background-color: #EEEEFF;
24 }
25 .gMenu,
26 .disabled,
27 .gButtonBox {
28   background-color:#BBBBEE;
29 }
30 .gDialog UL,
31 .gDialog LI {
32   border-color: #BBBBEE;
33 }
```

The next step is to change our application's XML-based configuration file and specify that the Genero Web Client use the new template file:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <APPLICATION Id="tutorial_step1" Parent="demo-tut-abstract">
03   <RESOURCE Id="res.template.tutorial"
Source="FILE">$(res.path.demo.dem-
tut)/web/tutorial/tutorialStep1.html</RESOURCE>
04   <EXECUTION>
```

```
05      <PATH>$(res.path.demo.dem-tut)/src</PATH>
06      <MODULE>card.42r</MODULE>
07    </EXECUTION>
08    <OUTPUT>
09     <MAP Id="DUA_GWC">
10      <THEME>
11        <TEMPLATE Id="_default">$(res.template.tutorial)</TEMPLATE>
12      </THEME>
13     </MAP>
14    </OUTPUT>
15 </APPLICATION>
```

Notes

1. Line 2: This file is a copy of the **tutorial.xml** file. This is done for the purpose of this demo; the application Id is changed to ensure the application displayed is for the current step of the tutorial.
2. Line 3: This line defines a `RESOURCE` for the new template page
3. Line 08-14: This line overrides the default `OUTPUT` template by instructing Genero Web Client to use the resource defined in line 3 when rendering a window that uses no style or an unreferenced style.

Fig 1. Built-in rendering

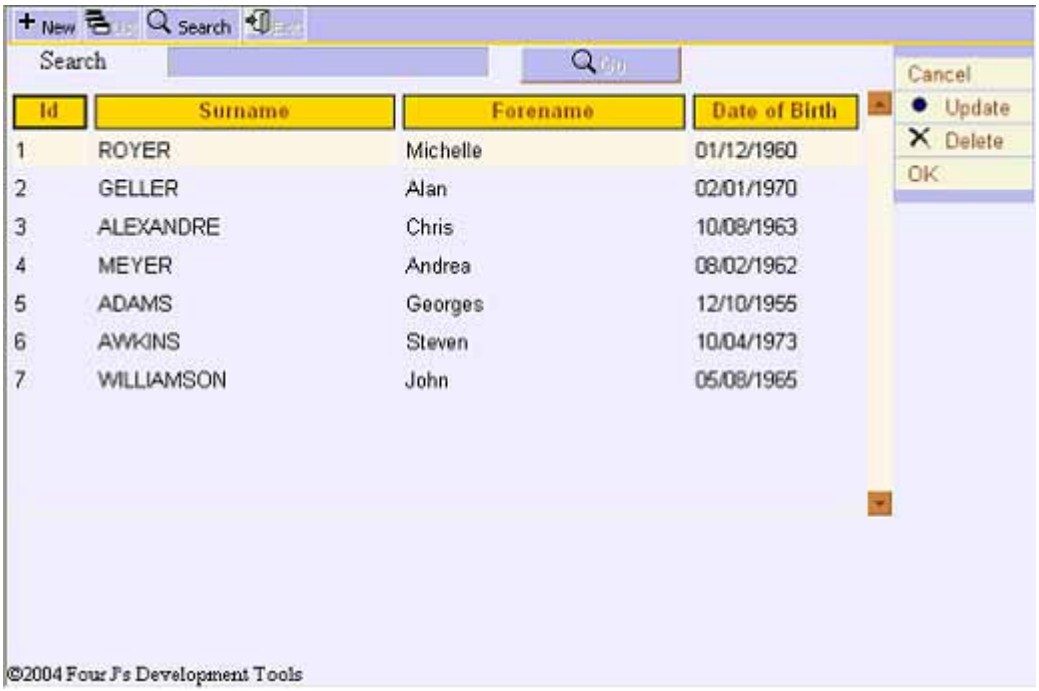


Fig 2. Customized rendering

Step 2: Use basic template paths

In this step, a new template is defined. This new template is to be used when adding a new card.

To declare the new template, we must modify the application XML file.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <APPLICATION Id="tutorial_step2" Parent="demo-tut-abstract">
03   <RESOURCE Id="res.template.tutorial"
Source="FILE">$(res.path.demo.dem-
tut)/web/tutorial/tutorialStep1.html</RESOURCE>
04   <RESOURCE Id="res.template.tutorial.addcard"
Source="FILE">$(res.path.demo.dem-
tut)/web/tutorial/tutorialStep2.html</RESOURCE>
05   ...
06   <OUTPUT>
07    <MAP Id="DUA_GWC">
08     <THEME>
09       <TEMPLATE Id="_default">$(res.template.tutorial)</TEMPLATE>
10       <TEMPLATE
Id="addcard">$(res.template.tutorial.addcard)</TEMPLATE>
11     </THEME>
```

```
12    </MAP>
13   </OUTPUT>
14 </APPLICATION>
```

Notes

1. Line 4: This line defines the resource to the template file.
2. Line 10: This line adds a new association between the window style *addcard* and the template file.

The new template file is the same as the file *tutorialStep1.html* except for the `BODY` tag content. The built-in rendering is changed with template-oriented rendering:

```
01 <form action="..." id="gDialogForm" method=post
gwc:attributes="action document/URL">
02   <div gwc:replace="application/intermediatetrigger" />
03   <div gwc:replace="application/dbdate" />
04   <div gwc:replace="application/toolbar" />
05   <div id="gForm" width="100%">
06     <table>
07      <tr>
08       <td width="55%">
09       <fieldset><legend>Identity</legend>
10        <table width="100%">
11         <tr>
12          <td rowspan="2" valign="top" width="25%" style="white-
space:nowrap;">
13           <I>Title</I><span gwc:replace="formfield[title]"/>
14          </td>
15          <td width="15%"><I>Surname</I></td>
16          <td width="60%"><span
gwc:replace="formfield[surname]"/></td>
17         </tr>
18         <tr>
19          <td width="15%"><I>Forename</I></td>
20          <td width="60%"><span
gwc:replace="formfield[forename]"/></td>
21         </tr>
22        </table>
23       </fieldset>
24      </td>
25      <td rowspan="2" valign="top" width="45%">
26      <fieldset><legend>Address</legend>
27        <I>Address</I><span gwc:replace="formfield[address]"/><br>
28        <I>Country</I><span gwc:replace="formfield[country]"/>
29      </fieldset>
30      </td>
31     </tr>
32     <tr>
33      <td>
34       <fieldset>
35       <table>
36        <tr>
```

```
37          <td><I>Date of Birth</I></td><td><span
gwc:replace="formfield[dbirth]"/></td>
38         </tr>
39         <tr>
40          <td><I>Marital Status</I></td><td><span
gwc:replace="formfield[marital]"/></td>
41         </tr>
42        </table>
43        </fieldset>
44       </td>
45      </tr>
46     </table>
47     <div align="right">
48     <span class="gButtonBox" gwc:content="action[accept]" /></span>
49     <span class="gButtonBox" gwc:content="action[cancel]" /></span>
50     </div>
51   </div>
52 </form>
```

Notes

1. Line 4: There is no topmenu defined in the application, so just the toolbar path is left.
2. Line 5: The *gForm* `DIV` tag will contain all widgets and will be used by the JavaScript scripts.
3. Line 6-46: This table holds the formfields used to input data in the application.
4. Line 13: A `gwc:replace` instruction is used with a `formfield` template path to tell Genero Web Client to render the formfield *title* of the application here.
5. Line 48-49: The `gwc:content` instructions are used with `action` template paths to render the *accept* and *cancel* actions.
6. As the layout is entirely custom-generated, the layout data information is not necessary anymore. The line declaring the gLayoutData variable is removed.

The following pictures show the application page when the 'New' action is clicked. This page uses both built-in rendering and template rendering. At this point, when using the template rendering, no error message is displayed if you leave required fields empty. Displaying of the error message will be done in the next tutorial step.

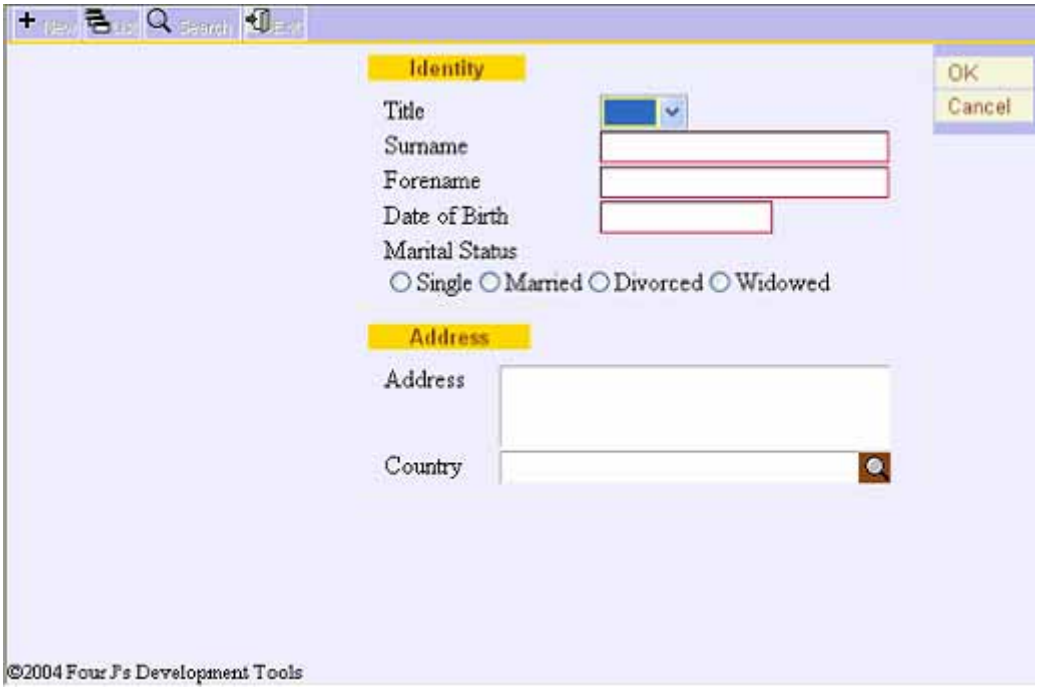


Fig 1. Built-in rendering

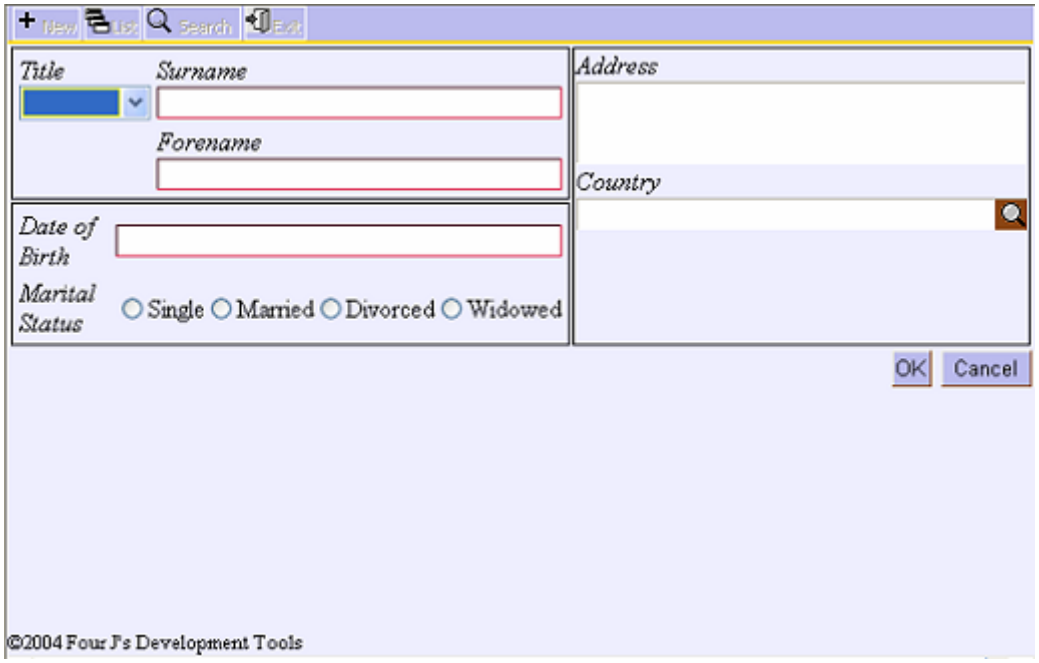


Fig 2. Customized Form rendering

Step 3: Displaying application messages and errors

At the end of the last section of the tutorial, the application is configured such that if you leave a required field empty, no error message displays. To display an error message and provide guidance to the user, template instructions are added that display the application's messages and error messages. These template instructions are added to the customized template file.

```
01 <div gwc:condition="application/error"
gwc:replace="application/error" />
02 <div gwc:condition="message" gwc:content="message" />
```

Notes

1. Line 1-2: The `gwc:condition` instruction tells Genero Web Client to display the messages only if they are defined. In this case, we display them using the `application/error` and `message` template paths.

We also modify the *tutorialStep1.css* file to prevent a message from overlapping the footer of the page:

```
01 #gError {
02   bottom: 20px;
03 }
04 #gMessage {
05   bottom: 20px;
06 }
```

After making these additions, an error message displays when a required field is left empty.

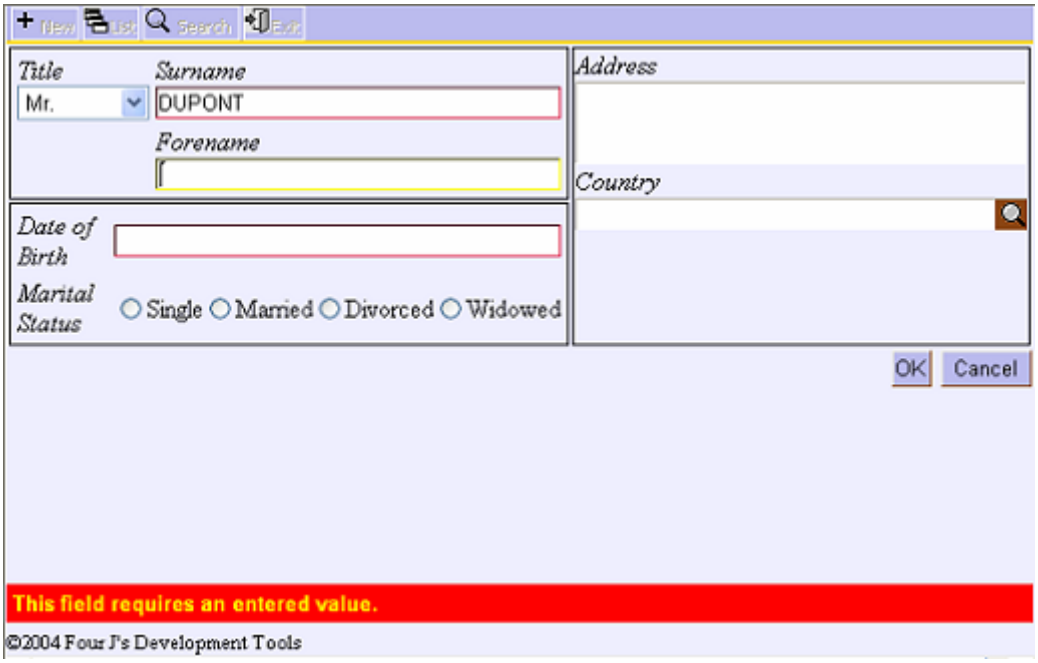


Fig 1. Error rendering

Step 4: Use advanced Genero Web Client instructions

For this next step in the tutorial, the *tutorialStep1.html* template file is changed to have a template-oriented rendering. For the purpose of the demo, use the file **tutorialStep4.html**.

As in Step 3, replace the built-in rendering lines in the file with:

```
01 <div gwc:define="searchActive formfield[search] &&
formfield[search]/active;
02                  tableActive  table[t1] && table[t1]/active;
03                  cardList     searchActive || tableActive"
04      gwc:omit-tag="true">
05   <div gwc:condition="cardList" gwc:omit-tag="true">
06     <form action="..." id="gDialogForm" method=post
gwc:attributes="action document/URL">
06       <div gwc:replace="application/intermediatetrigger" />
07       <div gwc:replace="application/toolbar" />
08       <div id="gForm">
09         <div gwc:replace="container[g1]" />
10         <div gwc:condition="table[t1]/size == 0"><em>No result
found.</em></div>
11         <div class=gTableBox gwc:define="t table[t1]"
gwc:condition="t/size" style="width:100%;">
```

```
12           <table width="100%" gwc:attributes="id t/id; class
'gTable' + (!searchActive && ' gEnabledScroll' || '')">
13            <colgroup>
14             <col id="t1-col-select" />
15             <col gwc:repeat="col t/columns" gwc:attributes="id
col/name + '-col'"></col>
16            </colgroup>
17            <thead>
18            <tr>
19             <th style="display:none;"><!-- Empty header for table
select --></th>
20             <th gwc:repeat="col t/columns"
gwc:attributes=&quotstyle repeat/col/first && 'display:none;'" >
21               <input class=gTableHeader type=submit
gwc:attributes="id col/name; value col/text; name col/id; disabled
searchActive">
22             </th>
23            </tr>
24            </thead>
25            <tbody>
26            <tr gwc:repeat="row t/rows">
27             <td style="display:none;"><input class=gTableSelect
type=radio gwc:attributes="value repeat/row/index; name t/id; checked
repeat/row/first; disabled searchActive"></td>
28             <td gwc:repeat="cell row/cells" gwc:attributes="style
repeat/cell/first && 'display:none;'" ><span gwc:replace="cell/value"
/></td>
29            </tr>
30            </tbody>
31           </table>
32           <input disabled type=hidden name=gOffset
gwc:attributes="value t/offset" >
33           <input disabled type=hidden name=gSize
gwc:attributes="value t/size" >
34           <input disabled type=hidden name=gPageSize
gwc:attributes="value t/PageSize" >
35         </div>
36         <div align="right">
37           <span class="gButtonBox"
gwc:condition="dialog/action[cancel]" gwc:content="action[cancel]" />
38           <span class="gButtonBox"
gwc:condition="dialog/action[cardupdate]"
gwc:content="action[cardupdate]" />
39           <span class="gButtonBox"
gwc:condition="dialog/action[carddelete]"
gwc:content="action[carddelete]" />
40         </div>
41       </div>
42     </form>
43   </div>
44   <div gwc:condition="!cardList" gwc:omit-tag="true">
45     <form action="..." id="gDialogForm" method=post
gwc:attributes="action document/URL">
46       <div gwc:replace="application/intermediatetrigger" />
47       <div gwc:replace="application/dbdate" />
48       <table  border="0" cellpadding="0" cellspacing="0">
49         <tr>
```

```
50           <td colspan=2>
51             <div gwc:condition="application/topmenu"
gwc:replace="application/topmenu" /><div gwc:condition="topmenu"
gwc:replace="topmenu" />
52             <div gwc:condition="application/toolbar"
gwc:replace="application/toolbar" /><div gwc:condition="toolbar"
gwc:replace="toolbar" />
53           </td>
54         </tr>
55         <tr>
56           <td width="100%"><div gwc:condition="form"
gwc:replace="form" /></td>
57           <td valign="top" id="gPanel"><div gwc:condition="menu"
gwc:replace="menu" /><div gwc:condition="dialog" gwc:replace="dialog"
/></td>
58         </tr>
59         <tr>
60           <td colspan=2 id="gInfo"><div gwc:condition="message"
gwc:content="message" /><div gwc:condition="error" gwc:content="error"
/></td>
61         </tr>
62       </table>
63     </form>
64   </div>
65 </div>
```

Notes

1. Line 1-4: Some variables are defined using the `gwc:define` instruction. *searchActive* will be true if the formfield *search* exists and is active; the same for *tableActive* and the table *t1*. *cardList* will be true if either of the previous two variables is true. The `gwc:omit-tag` instruction tells Genero Web Client not to render this tag.
2. Line 5: The content of this tag is rendered when the application is looking for a card or displaying the table.
3. Line 9: The `container` template path is used to render the grid *g1* containing the search form in the application PER file.
4. Line 11: A variable *t* representing the table *t1* is defined. This variable will be used later in the inner tags. The table is only displayed if it is not empty.
5. Line 12: An *id* attribute is added to the tag with the id of the table. It's used by the JavaScript wrappers, which also need the class *gTable*. Add *gEnableScroll* to the tag class if not in search mode. This disables the table scrollbar.
6. Line 20-22: A `gwc:repeat` instruction is used to set up the table headers. Once again, the proper attributes of an input tag are set in order to have a wrapped header (i.e. for sorting the table). The input is disabled if in search mode. Notice the use of a `gwc:attributes` instruction to omit displaying the first column.
7. Line 26-29: Another `gwc:repeat` instruction is used to display the table rows. The first cell of each row is used by the table wrapper to handle row selection. The row selection is disabled if in search mode. Here too a condition is added that will not render the first column of the table.
8. Line 32-34: These inputs are used by the table wrapper to handle row selection.

9. Line 37-39: These lines display the form actions (if they exist).
10. Line 44-64: When only displaying the application top window, the built-in rendering is used.

The listing page of employee cards now uses a customized template:

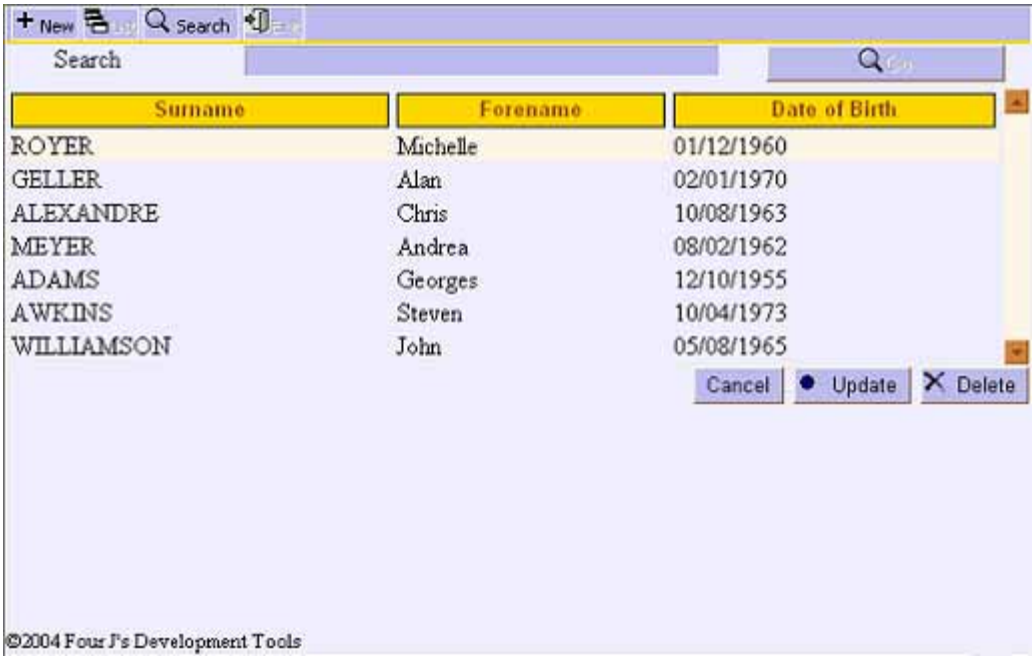


Fig 1. Table rendering using gwc:repeat instruction

Step 5: Use Genero Web Client JavaScript API

The Genero Web Client JavaScript API provides easy access to actions used to interact with Genero Web Client.

In this step, the way the *cardupdate* and *carddelete* actions are rendered and handled will be changed. To change these actions, the template file must be modified:

```
01    ...
02    <tr gwc:repeat="row t/rows">
03     <td style="display:none;"><input class=gTableSelect type=radio
gwc:attributes="value repeat/row/index; name t/id; checked
repeat/row/first; disabled searchActive"></td>
04     <td gwc:repeat="cell row/cells" gwc:attributes="style
repeat/cell/first && 'display:none;'" ><span gwc:replace="cell/value"
/></td>
05     <td width="60px" gwc:condition="row/cell[id]/value">
06       <input type="image" src="$(connector.uri)/tutorial/img/circle"
```

```
07              gwc:attributes="onclick
'javascript:gGWCEvent().Append(gTable(\'${t/id}\').Select(${repeat/row/
index}), gAction(\'cardupdate\')).Send();';
08                              disabled !dialog/action[cardupdate]">
09      <input type="image" src="$(connector.uri)/tutorial/img/delete"
10              gwc:attributes="onclick
'javascript:gGWCEvent().Append(gTable(\'${t/id}\').Select(${repeat/row/
index}), gAction(\'carddelete\')).Send();';
11                              disabled !dialog/action[carddelete]">
12     </td>
13    </tr>
14    ...
15 <div align="right">
16   <span class="gButtonBox" gwc:condition="dialog/action[cancel]"
gwc:content="action[cancel]" />
17 </div>
```

Notes

1. Line 5-12: Add a cell to each row. The cell contains two images. Define the *onclick* event of these images as being JavaScript code. Use the Genero Web Client JavaScript API to prepare two events before sending them to the Genero Web Client.
    - The first event is the current row selection using the `Select` method of the `gTable` object.
    - The second event is the action *cardupdate* or *carddelete*
2. Line 16: Remove the *cardupdate* and *carddelete* action buttons.

The listing page now has the 'Update' and 'Delete' actions integrated in the table.

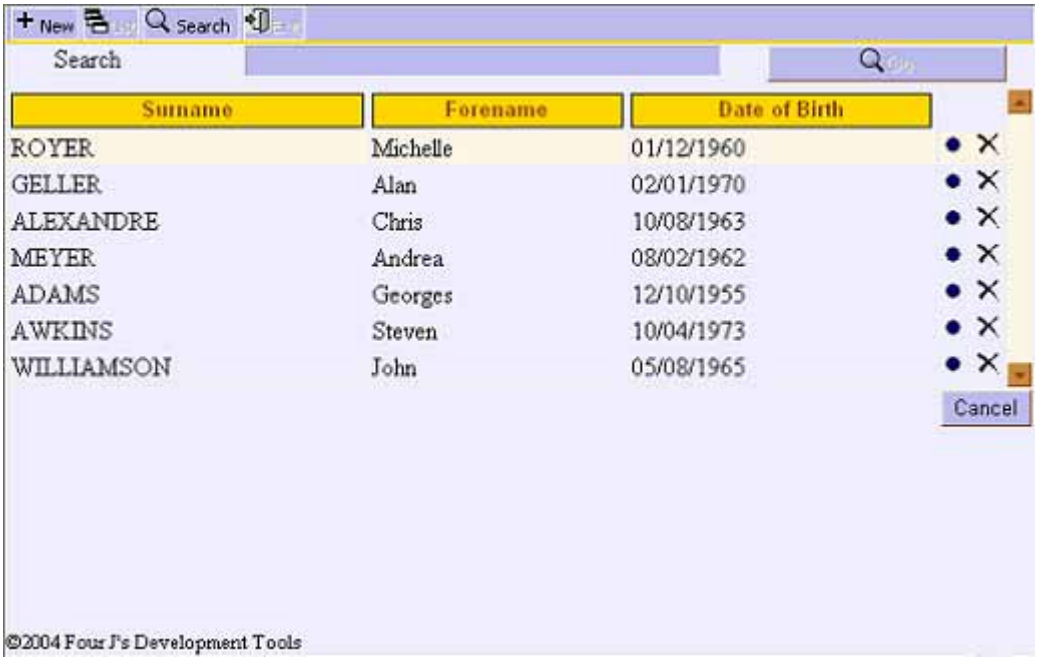


Fig 1. Actions using Genero Web Client JavaScript API